UNIVERSITY OF BELGRADE
FACULTY OF MATHEMATICS

Mirjana M. Maljković

PREDICTION OF ALPHABETS OF LOCAL PROTEIN STRUCTURES USING DATA MINING METHODS

Doctoral Dissertation

Belgrade, 2021.

UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Mirjana M. Maljković

PREDVIĐANJE ALFABETA LOKALNE STRUKTURE PROTEINA PRIMENOM METODA ISTRAŽIVANJA PODATAKA

doktorska disertacija

Beograd, 2021.

Advisor:

dr Nenad Mitić, full professor
University of Belgrade, Faculty of Mathematics

Committee:

dr Saša Malkov, associate professor
University of Belgrade, Faculty of Mathematics

dr Jovana Kovačević, assistant professor
University of Belgrade, Faculty of Mathematics

dr Alexandre de Brevern, senior researcher
Université de Paris, INSERM UMR_S 1134, DSIMB, Université de la Réunion, INTS 6, rue Alexandre Cabanel 75015 Paris, France

Date of the defense: _____

Mentor:

dr Nenad Mitić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Saša Malkov, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Jovana Kovačević, docent
Univerzitet u Beogradu, Matematički fakultet

dr Alexandre de Brevern, viši istraživač
Université de Paris, INSERM UMR_S 1134, DSIMB, Université de la Réunion, INTS 6, rue Alexandre Cabanel 75015 Paris, France

Datum odbrane: _____

Dissertation title: Prediction of alphabets of local protein structures using data mining methods

Abstract: Proteins are linear biological polymers composed of amino acids whose structure and function are determined by the number and order of amino acids. The structure of the protein has three levels: primary, secondary and tertiary (three-dimensional, 3D) structure. Since the experimental determination of protein 3D structure is expensive and time-consuming, it is important to develop predictors of protein 3D structure properties from the amino acid sequence (primary structure), such as 3D structure of the protein backbone. The 3D structure of the backbone can be described using prototypes of local protein structure, i.e. prototypes of protein fragments with a length of few amino acids. A set of local structure prototypes determines the library of local protein structures, also called the structural alphabet. A structural alphabet is defined as a set of N prototypes of L amino acid length. The subject of this dissertation is the development of models for the prediction of structural alphabet prototypes for a given amino acid sequence using different data mining approaches. As one of the most known, structural alphabet Protein Blocks (PBs) was used in one part of the doctorial research. Structural alphabet PBs consists of 16 prototypes that are defined using fragments of 5 consecutive amino acids. The amino acid sequence is combined with the structural properties of a protein that can be determined based on amino acid sequence (occurrence of repeats in the amino acid sequence) and results of predictors of protein structural properties (backbone angles, secondary structures, occurrence of disordered regions, accessible surface area of amino acids) as an input to the prediction model of structural alphabet prototypes. Besides the development of models for prediction of prototypes of existing structural alphabet, the analysis of the capability of developing new structural alphabets is researched by applying the TwoStep clustering algorithm and construction of models for the prediction of prototypes of new structural alphabets. Several structural alphabets, which differ in the length of prototypes and the number of prototypes, have been constructed and analyzed. Fragments of the large number of proteins, whose structure is experimentally determined, were used to construct the new structural alphabets.

Keywords: data mining, structural alphabet, prediction model, Protein Blocks

Research area: computer science

Research sub-area: data mining, bioinformatics

Naslov disertacije: Predviđanje alfabeta lokalne strukture proteina primenom metoda istraživanja podataka

Rezime: Proteini su linearni biološki polimeri sastavljeni od aminokiselina čiji broj i redosled određuju strukturu i funkciju proteina. Struktura proteina je definisana sa tri nivoa: primarnom, sekundarnom i tercijarnom (trodimenzionalnom, 3D) strukturom. Pošto je eksperimentalno određivanje 3D strukture proteina skupo i vremenski zahtevno, postoji potreba za razvojem programa koji na osnovu aminokiselinske sekvence (primarne strukture) predviđaju osobine 3D strukture, kao što je 3D struktura glavnog lanca proteina (eng. backbone). 3D struktura glavnog lanca proteina može da se opiše korišćenjem prototipova lokalne strukture proteina, tj. delova proteina od nekoliko uzastopnih aminokiselina. Skup definisanih prototipova lokalne strukture čini biblioteku lokalnih struktura proteina, koja se još naziva i strukturni alfabet (eng. structural alphabet). Svaki strukturni alfabet je definisan kao skup od N prototipova dužine L aminokiselina. Predmet ove disertacije je pravljenje modela za predviđanje prototipova strukturnog alfabeta za zadatu aminokiselinsku sekvencu primenom različitih algoritama istraživanja podataka. Kao jedan od najpoznatijih, strukturni alfabet *Protenski blokovi* (eng. Protein Blocks) je korišćen u jednom delu istraživanja u okviru disertacije. Strukturni alfabet *Proteinski blokovi* se sastoji od 16 prototipova koji su napravljeni na osnovu delova proteina od 5 uzastopnih aminokiselina. Kao ulaz u model za predviđanje prototipova strukturnog alfabeta koriste se strukturne osobine proteina koje mogu da se odrede na osnovu aminokiselinske sekvence (lokacija ponavljajuće niske u aminokiselinskoj sekvenci) i rezultati predviđanja nekih strukturnih osobina proteina (uglovi glavnog lanca, sekundarne strukture, pojavljivanje neuređenih regiona, pristupačna površina). Pored razvoja modela za predviđanje prototipova postojećeg strukturnog alfabeta, u radu je izvršena i analiza mogućnosti razvoja novih strukturnih alfabeta primenom algoritma klasterovanja *TwoStep* i pravljenje modela za predviđanje prototipova novih strukturnih alfabeta. Radi analize, napravljeno je više strukturnih alfabeta sa različitim brojem prototipova i različite dužine prototipova. Za istraživanje novih strukturni alfabeta korišćeni su delovi velikog broja proteina čija je struktura eksperimentalno određena.

Ključne reči: istraživanje podataka, strukturni alfabeti, model za predviđanje, Proteinski blokovi

Naučna oblast: računarstvo

Uža naučna oblast: istraživanje podataka, bioinformatika

Acknowledgements

I would first like to thank my advisor, dr Nenad Mitić, for patient support and guidance throughout my doctorial studies and the writing of this dissertation. I would like to acknowledge dr Alexandre G. de Brevern and dr Miloš Beljanski for their wonderful collaboration and invaluable expertise during the research. I am deeply grateful to dr Saša Malkov and dr Jovana Kovačević for their insightful comments and suggestions.

Contents

List of Figures

List of Tables

1 Introduction

Data mining is a field of computer science whose methods (classification, clustering, association rules) discover significant patterns in a dataset. The availability of accurate data is the basis of successful research in the field of data mining. The significant developments in computer science in recent decades have enabled storage and easy access to a large amount of data. The availability of data from various fields has enabled the usage of data mining methods , and it has led to great scientific and industrial progress.

Bioinformatics is the interdisciplinary field that combines knowledge from several scientific areas (mainly from mathematics, computer science and biology) with the aim to develop or improve the algorithms for solving intrinsically complex biology problems. One such problem is the prediction of 3D protein structure from an amino acid sequence.

Since the experimental determination of the protein 3D structure is expensive and time-consuming, the development of an accurate predictor of protein 3D structure from an amino acid sequence is crucial for many structure-based studies, such as protein function prediction or drug design. A common approach for solving this problem is to use predictors of different protein structure properties and then combine their results to obtain a more accurate prediction of the protein 3D structure [6].

For the approximation of protein backbone, prototypes of the local 3D protein structure conformations can be used. Prototypes are determined based on 3D protein structure conformations of fragments of consecutive amino acids in a protein sequence. The set of defined prototypes composes the library of local protein structures, also called the structural alphabet (SA). The whole protein backbone can be described by concatenating the prototypes which correspond to consecutive fragments extracted from the beginning to the end of a protein sequence.

Several research teams have developed different structural alphabets using the

experimentally determined 3D structure of proteins. Developed structural alphabets differ in the length of prototypes (expressed in the number of consecutive amino acids), the properties used to describe the protein backbone (atom coordinates, distances, backbone angles), methods used to define prototypes and the number of obtained prototypes. One of the most used structural alphabets is Protein Blocks which consists of 16 prototypes described with backbone angles for five consecutive amino acids [15]. Besides the approximation, structural alphabets can be used for the prediction of the protein backbone. The predictor of prototypes of structural alphabets can be used as one step in a system of protein structure properties predictors for the prediction of the full 3D structure of a protein or in bioinformatics studies based on structural alphabets, such as protein structure analysis and comparison of proteins using structural alphabet prototypes [69]. Structural alphabets and predictors of structural alphabet prototypes from an amino acid sequence can be constructed using data mining methods.

Clustering algorithms are used for finding the appropriate prototypes of the local 3D protein structure conformations for approximation of protein backbone and classification algorithms are used for the development of predictors of prototypes from an amino acid sequence.

The dissertation includes a description of the development of accurate predictors of prototypes of structural alphabet Protein Blocks using data mining methods. In order to create a base for the development of new predictors of local protein structures, the capability of developing a new structural alphabet using a clustering algorithm that was not used in previous research studies was also analyzed in the thesis.

The text is organized as follows:

- Chapter 2 describes approaches for the preparation of dataset and data mining methods used in the research. Overview of classification algorithms used for the development of predictors for structural alphabet Protein Blocks is given, as well as an overview of clustering algorithms used for construction of structural alphabet Protein Blocks and sets of prototypes of local protein structures obtained in the second part of the research.

- Chapter 3 describes the structure of proteins and existing structural alphabets. Structural alphabet Protein Blocks and existing predictors of its prototypes are described in details.

- Chapter 4 describes the process of building the new models for prediction of Protein Blocks, obtained as a result of the research. Also, amino acid sequence information used as input for prediction models is described.

  The performance of new models for the prediction of Protein Blocks developed using several classification algorithms and the results of comparison of the obtained Protein Blocks prediction models are analyzed.

  The performance of the developed model is compared with the performance of existing predictors of the structural alphabet Protein Blocks. The results described in this part of the dissertation are submitted for review in [64].

- Chapter 5 contains results of the analysis of the capability of developing a new structural alphabet using a clustering algorithm, in order to use it as the basis for the development of new predictors of local protein structures.

- Chapter 6 represents the conclusion about the results obtained in the research and plans for future work.

2 Data mining methods

2.1 Data mining

Data mining can be described as a process of discovering useful patterns in a dataset. Different data mining methods could be used for discovering patterns in a dataset like classification, clustering and association rules. Various algorithms were developed for each data mining method. Figure 2.1 shows the general process of data mining. The process usually begins with getting familiar with the data. In order to use information obtained from different sources, data has to be merged and cleaned from useless attributes or instances. Each data mining algorithm requires a specific format of input data; so before applying the desired algorithm, the data has to be transformed into an appropriate form. The result of an applied data mining algorithm on a transformed data is called a model. Before using the obtained model in the real world, the model has to be evaluated. Each data mining method has specific quality measures which can be used to evaluate the model. The data mining process is rarely linear. It is usually iterative because we often need to go back one or more steps after evaluating the model in order to improve the results.

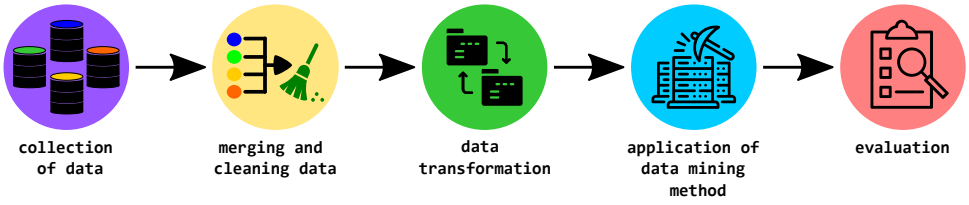

Figure 2.1: General steps in data mining process [1]

[1]Some icons are from https://thenounproject.com

The requirement of a good data mining process is the availability of files with quality data for the research. The significant development in computer science in recent decades has enabled easier storage and access to a large amount of data from various fields, such as medicine and biology. The availability of data has enabled the application of data mining methods in various fields, and it has led to great scientific and industrial progress.

2.2 Format of dataset

Datasets consist of instances (objects). Instances are described using attributes (properties). An attribute type can be categorical or numerical. The datasets used for data mining can be stored in different formats. Based on the format of the dataset, different data mining algorithms can be applied to the dataset. Two commonly used data formats for the description of amino acid sequences in bioinformatics are fixed-length sliding window format and sequence format [44, 42].

Let $S = E_1E_2...E_i...E_N$ be a sequence of N elements and each element at position j ($j = 1, 2, ...N$) is described with k attributes $(Attr_j^1, Attr_j^2, ..., Attr_j^k)$.

In the fixed-length sliding window format data, a fragment of $2 * L + 1$ consecutive elements in a sequence presents one instance of a dataset (where $2 * L + 1$ is from 1 to N). Fragment of $2 * L + 1$ consecutive elements is also called a sliding window of length $2 * L + 1$. An instance in a dataset corresponds to a central element in a fragment. In addition to the attributes of the central element, the attributes of L elements on its left side, and attributes of L elements on its right side, are extracted too, in order to present information about the neighbours of the central element. Besides the attributes of an individual element (element level attributes), the instance may also contain attributes that are related to more than one element in a fragment (attributes at the level of a fragment). Figure 2.2 illustrates the fixed-length sliding window data format. In Figure, $E_{x-L}E_{x-L-1}...E_x...E_{x+L-1}E_{x+L}$ presents fragment with central element at position x in a sequence. Each element is described with k attributes. $Attr_i^j$ is j-th element level attribute of element at position i. Besides element level attributes, m fragment level attributes are also shown. $FAttr_x^j$ is a j-th fragment level attribute. Thus, a fragment is described with $k * (2 * L + 1) + m$ attributes: $(Attr_{x-L}^1, Attr_{x-L}^2, ..., Attr_{x-L}^k, ..., Attr_x^1, Attr_x^2, ..., Attr_x^k, ..., Attr_{x+L}^1, Attr_{x+L}^2, ..., Attr_{x+L}^k, FAttr_x^1, FAttr_x^2, ..., FAttr_x^m)$.

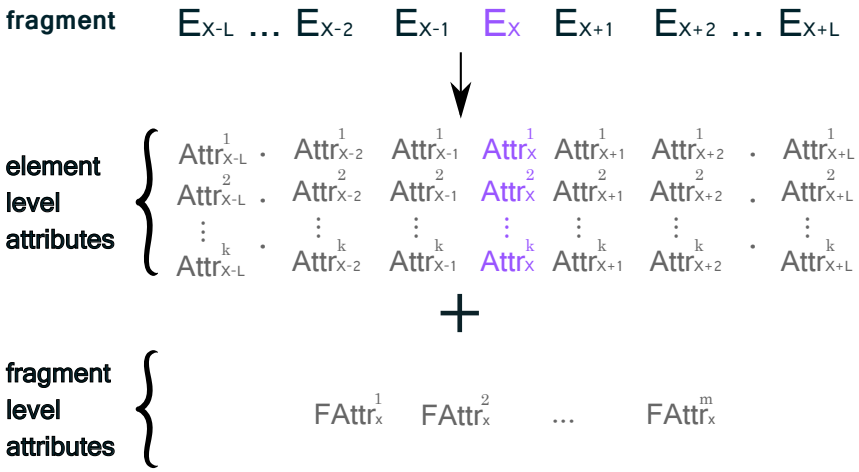

Figure 2.2: Fixed-length sliding window data format

One instance in a sequence data format presents the whole sequence. Figure 2.3 illustrates the sequence format data. Sequence consists of N elements. Since each element is described with k properties, for the description of a sequence is used $N*k$ attributes: $(Attr_1^1, Attr_1^2, ..., Attr_1^k, ..., Attr_i^1, Attr_i^2, ..., Attr_i^k, ..., Attr_N^1, Attr_N^2, ..., Attr_N^k)$.

Fixed-length sliding window data format enables the description of short-range relationships, i.e. only relationships between $2 * L + 1$ elements described with an instance. Sequence format describes the short-range relationships between elements in a sequence (elements in a neighbourhood), as well as the long-range relationships (relationship of elements with distant positions in the sequence).

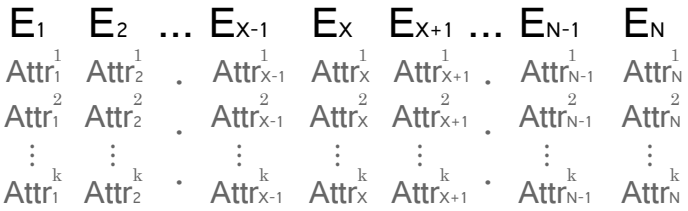

Figure 2.3: Sequence data format

2.3 Classification

Let (X, Y) be a set of attributes of a dataset, where $X = (X_1, X_2, ...X_n)$ is a set of input attributes, and Y is the target attribute. Classification is a process of learning the relationship between the input attributes and the target attribute. A

		predicted class			
		C_1	C_2	**...**	C_n
actual class	C_1	C_{11}	C_{12}	...	C_{1n}
	C_2	C_{21}	C_{22}	...	C_{2n}
	...	...	...	...	...
	C_n	C_{n1}	C_{n2}	...	C_{nn}

Table 2.1: Confusion matrix for a multiclass classification problem

target attribute has to be a categorical attribute with two or more distinct values, which are called *classes*. There is no type restriction for the X attributes. The result of classification is a *model*, which can be described as function $f : X \to Y$.

A classification model can be used to predict a class for an instance with an unknown class. For example, classification can be used to predict if a patient has a particular disease. To build such a model, it would be necessary to have health data of patients who have the disease, as well as data of patients who do not have the disease. The obtained model can then be used to predict whether a new patient has the disease or not based on his or her health data.

Evaluation measure

Various measures can be used for the evaluation of a classification model. The confusion matrix (see Table 2.1) shows the results of classification for each pair of classes. Let n be the number of classes in a dataset and let C_i denote the i-th class. The confusion matrix is a square matrix with dimension $n \times n$. Each row or column is labelled with one class. The labels of rows present the true classes of instances, and labels of the columns present predicted classes of instances. C_{ij} is the number of instances of class C_i classified as class C_j by a model.

Based on the confusion matrix, the following measures of model performance can be calculated:

- **Accuracy** is the fraction of instances accurately predicted by a model and is calculated by:

$$Accuracy = \frac{\sum_{i=1}^{n} C_{ii}}{\sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij}} \tag{2.1}$$

  Accuracy is used for a general evaluation of a model.

- **Precision, recall** and $\boldsymbol{F_1}$ are used for a model performance per class.
  - Precision of a class C_i is a fraction of the number of instances of class C_i that the model has correctly classified among instances that the model has classified with class C_i:

$$\text{Precision for class } C_i = \frac{C_{ii}}{\sum_{j=1}^{n} C_{ji}} \tag{2.2}$$

  - Recall of a class C_i is a fraction of instances of class C_i that the model has correctly classified:

$$\text{Recall for class } C_i = \frac{C_{ii}}{\sum_{j=1}^{n} C_{ij}} \tag{2.3}$$

  - F_1 is the harmonic mean of precision and recall:

$$F_1 = \frac{2 * r * p}{r + p} = \frac{2}{\frac{1}{r} + \frac{1}{p}} \tag{2.4}$$

More information about presented evaluation measures can be found in [79, 3].

In bioinformatics, for accuracy is commonly used label Q_k where k is the number of classes in a dataset. For example, accuracy for the prediction of Protein Blocks is labelled as Q_{16}, while the accuracy for the prediction of 3-states secondary structures is labelled as Q_3.

Dataset partition

Evaluation of the classification model using the instances on which it was trained sometimes can give an incorrect estimate of how a model will behave on unseen instances during the model building phase. A problem that may occur in classification is overfitting a model to instances used for model building. Overfitted model classifies well instances used for building a model but classifies poorly the unseen instances. To prevent the overfitting, the dataset may be split into two parts: the training part and the test part. The training part is used for building (training) model, while the test part is used for the final evaluation of the model. The training part is usually larger than the test part (typically ratio of training and test part is 70:30). Partition of the dataset should be stratified by classes.

Stratified partition ensures that classes are represented in the same percentage in each part obtained by partition.

For most classification algorithms, it is necessary to adjust the algorithm's parameters in order to get the optimal model for the used dataset. If the training part is used for training the models with different parameter values and the same test part for evaluation of each of the obtained models, then the behaviour of models on instances in a test part is taken into account during the selection of optimal parameter values. This should not happen, since the test part should be used for the final test of the optimal model. To overcame this problem, data can be split into three parts: training part, validation part and test part. The training part is used for building models with different values for algorithm parameters. The validation part is used to estimate how the obtained models will behave on the unseen data. Based on the estimated behaviour of the models on the validation part, the optimal model is chosen. The test part is used for the final evaluation of the optimal model. The final model can be built using optimal parameters on the instances from the training and validation parts.

The consequences of splitting a dataset into three parts are: a smaller number of instances is used for training a model in the phase of parameter adjustment, and the obtained results are more sensitive to the partition of instances on parts. If the dataset is small, the training part may not be representative. To overcome this problem, a technique called cross-validation can be used to estimate the performance of models trained with different parameter values in order to find the optimal values of parameters. In cross-validation, a dataset is first split into the training part and test part. In the phase of adjusting the parameters, the training part is split in k equal-sized subsets called *folds*. The following procedure is performed for each combination of parameter values: k different models are trained with the same parameter values for the algorithm; each model is trained using $k-1$ folds, and validated on the remaining one fold; each model uses a different fold for validation. To evaluate the performance of a model with used parameter values, each obtained model is applied on a corresponding fold for validation and the performance measure (usually accuracy) is calculated. The average value of calculated evaluation measures on folds for validation is the estimated performance of a model with used parameter values. In cross-validation, each instance is part of the training dataset for estimating the performance of a model with the used parameter values ($k-1$ times), and it is also used for validation (once). Using the

algorithm's optimal parameter values (combination of parameter values with the best-estimated performance), the final model is trained using the whole training part. The final evaluation is done on the test part. The pseudocode of the process of training an optimal model using cross-validation is shown in Algorithm 1. For simplicity, accuracy is used as an evaluation measure.

Algorithm 1: Procedure for model selection using cross-validation

Input:
DS: dataset;
$params$: set of different combinations of values for parameters of algorithm;
k: number of folds;
Split DS to $train\_part$ and $test\_part$
Split $train\_part$ to k equal folds: $folds = [fold_1, fold_2, ..., fold_k]$
$params\_validation = \{\}$
foreach $param \in params$ **do**
  $avg\_acc = 0$
  for $i = 1$ *to* k **do**
    $folds\_for\_train = folds \setminus fold_i$
    $model$=train_model($folds\_for\_train$, $param$)
    acc= apply_model($model$, $fold_i$)
    $avg\_acc = avg\_acc + acc$
  end
  $avg\_acc = avg\_acc/k$
  $params\_validation[param] = avg\_acc$
end
$optimal\_param$ = key of params_validation with maximal value
$final\_model$=train_model($train\_part$, $optimal\_param$)
$final\_evaluation\_acc$= apply_model($final\_model$, $test\_part$)

More information about presented techniques for data partition can be found in [79, 3].

Ensembles

During the model-building phase, the decision boundaries between classes are learned based on the training dataset [3]. Each algorithm uses certain assumptions about the decision boundaries between classes. Also, different models can be obtained by using the same algorithm, but different training datasets. Causes of model's errors in classification of instances can be:

- *bias* - a difference of learned decision boundaries from true decision boundaries between classes caused by assumptions of relationships between attributes and classes;
- *variance* - sensitivity on changing the decision boundaries due to variation in training data.

During the training phase of a model, a bias-variance trade-off should be found.

In order to reduce bias and/or variance in a classification, ensemble techniques can be used. Ensemble techniques use multiple classification models for the classification of an instance. Classification models in the ensemble can be build using different classification algorithms on the same training data or with the same algorithm but different training data or on the same training data but with different weights of instances. Outputs of used models are combined for the final prediction of the instance's class. Ensemble techniques that can be used are bagging and boosting [3].

The bagging technique [8] was designed to reduce the model variance. It is also called bootstrapped aggregating because it uses the *bootstrapping* approach for dataset preparation for building classification models of an ensemble. For training each of the ensemble models, instances are sampled uniformly from a training set with replacement. Sampled datasets are of the same size as the training dataset. Each sampled dataset contains on average 63.2% of instances from the training dataset, while some instances appear multiple times. Class with majority votes from classification models is usually chosen as ensemble prediction.

Boosting technique [39] can be used to reduce the model bias. Each classification model is trained using the same classification algorithm and the whole training dataset, but weights are assigned to instances in a training dataset. The first model is trained using the same weight ($\frac{1}{N}$, where N is the number of instances) for all instances. Before training the succeeding model, the weights of instances that are misclassified with the previous model are increased in order to build a model that will better classify them, i.e. correct the errors of the previously made model. Classification algorithm can be modified to use weights of instances, or bootstrap sampling of the training data can be used while weights are determined as probabilities of choosing an instance.

Cost-matrix

Cost-sensitive learning technique [79] can be used for assigning the cost to misclassifications of instances of one class with other classes. Cost-sensitive learning is based on using a cost-matrix. Cost-matrix W is a square matrix with non-negative elements. It has a same form like a confusion matrix (see Table 2.1). Element $W(i, j)$ of the matrix is the cost (or weight) of classifying an instance into class j if its true class is i. By default, the cost of classifying an instance to a true class is 0, while classifying it with the wrong class is 1. In the case of unbalanced classes, a model with default cost-matrix might misclassify less frequent classes while achieving good accuracy. For example, in a binary classification problem, if 90% of instances belong to one class, while the rest of instances belong to another class, the model which classifies poorly the instances of the less frequent class will have good accuracy.

The cost of a model is calculated by the formula

$$Cost = \sum_{i=1}^{n} \sum_{j=1}^{n} C(i, j) * W(i, j) \tag{2.5}$$

where n is the number of classes and C is the confusion matrix. If a user defines the cost-matrix, it is taken into account during the model building phase. The classification algorithm tries to avoid misclassifications with a high error cost. The goal is to obtain a model with the lowest possible cost.

Decision trees

The decision tree [79] is a classification technique for which a classification model has a form of a tree with two types of nodes:

- *non-terminal* nodes that have one or none parent, and two or more children. The non-terminal node has an associated attribute test question, which can have two or more answers. Each branch from parent to child corresponds to one answer to the parent's attribute question. Test questions are usually in a form: What is the value of the test attribute? The purpose of the test attribute is to separate instances that belong to different classes into subsets. The root node is a non-terminal node with no parent, while other non-terminal nodes in a tree have a parent.

- *leaves* are terminal nodes; they have one parent and none children. Each leaf is associated with one class from a dataset on which a classification model was built.

The classification of an instance using the decision tree is like answering the questionnaire using the instance values of attributes. Starting from the root, an instance goes down the branch which corresponds to the instance's value of a test attribute. If an instance goes to an internal node, the procedure is repeated, and if it goes to a leaf node, the test instance is classified with the class assigned to that leaf. Each path from a root to a leaf is one rule for the classification. Figure 2.4 shows an example of decision tree. Non-terminal nodes are coloured in orange, and leaf nodes in yellow. The root is a node with attribute X as a test attribute. One rule for classification is: If instance has value c in attribute X and m or n as value of attribute Z, classify it with class C_2.

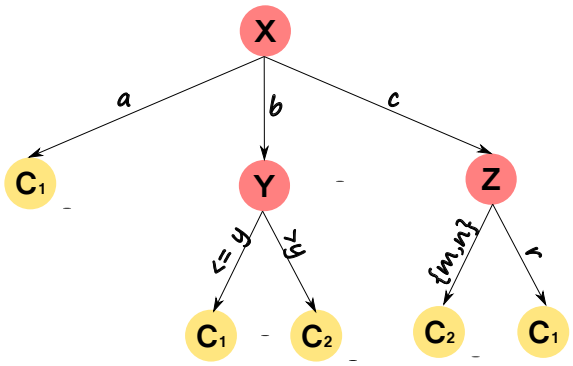

Figure 2.4: Example of decision tree

If a test attribute is categorical, the number of branches can be from two to the number of unique values in a test attribute. If the number of branches is smaller than the number of unique values in a test attribute, values are grouped and one branch corresponds to one or a group of values. In Figure 2.4, branches of a root correspond to one value in a test attribute X, while root's rightmost child with test attribute Z has a branch with grouped values (m and n). Values of a continuous attribute are divided into two or more intervals and a branch is assigned to each interval.

Various algorithms for building decision trees are proposed. They usually build the decision tree using local decisions about the one best attribute for dividing the instances. The aim of the division is to get children as clean as possible. A node is clean if all its instances belong to the same class. The base steps in building a decision tree are:

1. Start from a root node that holds all training instances.
2. (a) If a node is clean or contains only instances with the same values for all attributes, define it as a leaf node and assign to it the majority class.
   (b) Else, find the best attribute for dividing the training instances into as clean children-nodes as possible.
3. Repeat step 2 for all non-terminal nodes.

Impurity measures are used to determine how clean the node is. Some of the impurity measures [79] that are determined based on the frequency of classes in the node t are:

- Gini index

$$\text{Gini Index}(t) = 1 - \sum_{j=1}^{n} [p(j|t)]^2 \tag{2.6}$$

- Entropy

$$\text{Entropy}(t) = -\sum_{j=1}^{n} p(j|t) * \log_2 p(j|t) \tag{2.7}$$

where n is the number of classes and $p(j|t)$ is the frequency of class j in the node t.

In order to choose the best attribute for the node splitting, each attribute is tested. Let the parent-node have N instances which are divided into l children-nodes, and a child i has a N_i instances ($i = 1, 2, ..., l$). The difference between the

impurity of the parent-node and his children-nodes, called gain [79], can be used to determine the best attribute. Gain is defined as:

$$\Delta = I(parent\_node) - I(children\_nodes) = \\ I(parent\_node) - \sum_{i=1}^{l} \frac{N_i}{N} * I(child_i\_node) \tag{2.8}$$

where I is the used measure for the impurity of a node.

Overfitted models are usually complex. The complexity is reflected in the depth of decision trees, and the number of leaves. Complex trees have a large depth and a large number of leaves. Also, in overfitted decision tree models, the leaves often have a small number of instances. To prevent obtaining overfitted models, i.e. building too complex model, two approaches commonly used in decision tree algorithms are:

- Prepruning the tree, i.e. defining criteria when to stop splitting the nodes, such as maximum depth of a tree, minimal number of instances that have to be in a non-terminal node so it can be split, the minimal number of instances that have to be in a leaf in order to split its parent, minimal gain that has to be between the node and its children in order to split a node, etc.
- Post-pruning the full-grown tree. Using a bottom-up approach, the tree is pruned by replacing weak subtrees of non-terminal nodes if this increases the gain. A subtree is replaced with a leaf or the most frequently used branch. The majority class of the instances is assigned to a new leaf. Different strategies for post-pruning have been proposed [79].

C4.5/C5.0

Quinlan proposed C4.5 algorithm [71] as a successor of the ID3[70] algorithm. C4.5 uses entropy as an impurity measure for node splitting. The disadvantage of entropy is that it favours categorical attributes with a large number of values. To overcome this problem, the gain ratio is used as a split criterion. The gain ratio takes into account the number of children-nodes when choosing the best attribute. If the parent-node has N instances which are devided into l children-nodes, where a child i has N_i instances ($i = 1, 2, ..., l$), gain ratio is defined as:

$$\Delta_{ratio} = \frac{\Delta}{-\sum_{i=1}^{l} \frac{N_i}{N} * \log_2 \frac{N_i}{N}} \tag{2.9}$$

where Δ is gain (see Equation 2.8). Node with a test based on categorical attribute can be split using (a) one branch for one value or (b) using one branch for a group of values. Node with a test based on numerical attribute is split with two branches; one branch corresponds to condition $A \leq x$, while the other corresponds to condition $A > x$, where x is the best threshold for splitting the values of attribute A. To overcame overfitting, C4.5 uses a method called pessimistic pruning in which a subtree can be replaced with (a) a leaf labelled with the most frequent class or (b) with the most frequently used branch.

The probability of error in a leaf is estimated by using the upper limit of confidence limits ($U_{CF}(N, E)$) for the binomial distribution for a given confidence level CF, where N is the number of instances in a leaf, and E is the number of incorrectly classified instances in a leaf. The predicted number of incorrectly classified instances for a leaf with N instances is $U_{CF}(N, E) * N$. The predicted number of incorrectly classified instances by a subtree is a sum of the predicted number of incorrectly classified instances by its branches. During the processing of the subtree in the pruning phase, the estimated error is calculated for the whole subtree as the sum of the estimated errors of its branches, and for the leaf which would be obtained by the subtree's pruning. If the number of estimated errors after pruning is smaller or the same, the subtree is pruned to a leaf. Also, the contribution to the estimated error reduction is checked if the subtree is replaced with one of its branches. If replacing a subtree with a branch or leaf contributes to the reduction of the estimated error, a subtree is replaced with a branch or leaf, depending on what contributes the most. Pruning is done from leaves to the root.

Quinlan also proposed algorithm C5.0, the successor of algorithm C4.5. New functionalities are added in C5.0, such as the usage of boosting technique and cost-matrix. Detailed information about the implementation of C5.0 algorithm in software SPSS Modeler can be found in [47].

CART

CART (Classification And Regression Trees) [9, 35] was developed by Breiman and his team. In CART, the target attribute can be categorical or numerical. In classification, CART can use Gini criterion, Entropy criterion or Twoing criterion [35]. Non-terminal nodes are split using two branches. For numerical input attributes, the best threshold for splitting is sought among the attribute values. Values of a categorical attribute are grouped into two parts and one branch is

assigned to each part. Categorical values are grouped in a way that gives the greatest increase in impurity. During the model building, a decision tree is built to the full size and then post-pruning is applied using the minimal cost-complexity algorithm. Cost-complexity measure for the decision tree T is defined as

$$R_\alpha(T) = R(T) + \alpha|T| \tag{2.10}$$

where $R(T)$ is the cost of the tree calculated using the training dataset, $|T|$ is the number of leaves, and α is a penalty for each leaf. In the pruning phase, the α initially has a value of 0, and then in each step, the α is increased. When α increases, the weakest node is cut off in order to reduce $R_\alpha(T)$. The process is executed in iterations until a decision tree containing only the root is obtained. A sequence of pruned decision trees, obtained in each iteration, is saved. The first element of the sequence is a fully grown tree (for $\alpha = 0$), and the last element is a tree containing only the root (for the greatest α). The decision tree that has the best cost-complexity on a validation set among the trees in the sequence is chosen as the optimal tree.

In the implementation of the CART algorithm, the user can specify values for pre-pruning parameters which define when to stop growing the decision tree, such as:

- *maximum tree depth* - the tree can grow until it reaches the specified maximum tree depth;
- *minimal instances for spliting* - minimal number of instances which are required to be in a node so it can be split;
- *minimal instances in a leaf* - minimal number of instances which are required to be in each leaf.

CART algorithm includes a mechanism for class balancing when deciding which class will be assigned to the node by comparing the frequency of each class in a node with the frequency of a class in a root.

Cost-matrix can be specified and is incorporated in an assignment of a class to a node, as well as in the process of building the model if a Gini index is used as an impurity measure.

Detailed information about the original definition of an algorithm can be found in [9]. Information about the implementation of CART algorithm in software SPSS

Modeler can be found in [47] and detailed information about the implementation in Python library Scikit-learn can be found in [23].

CHAID

CHAID (CHi-squared Automatic Interaction Detection) [55] is a method that uses the statistical significance test as a criterion rchangewhen choosingfor selection of the attribute for node splitting during the tree growing phase. Chi-squared test is used for classification. The algorithm uses only categorical attributes, so continuous attributes need to be transformed into ordinal attributes.

In the statistical significance test for attribute X, the null hypothesis of independence of attribute X and target attribute Y is tested using a contingency table in which values of attribute X are rows and classes in target attribute Y are columns.

Before the selection of the best attribute for node splitting, similar values of each attribute X are merged. The similarity of attribute values is defined based on their statistical difference with respect to the target attribute. The result obtained by merging the values of an attribute is called a compound value.

Until the number of different values in the attribute is greater than two and as long as there are similar values according to the statistical test do:

- Find the most similar pair of values of attribute X with respect to the target attribute, i.e. according to the p-value. If the obtained largest p-value is greater than the specified threshold for merging, merge the values into a new group (new compound value). If an attribute X is ordinal, only contiguous values can be merged, while for nominal attributes any two values can be merged.

- If the new compound value consists of three or more original values, find the best binary split according to the statistical test, i.e. with the smallest p-value. If the p-value is less than the specified threshold for splitting, split the compound value into two parts.

If the number of instances of any of the obtained (compound) attribute values is less than the specified threshold, that value is merged with the most similar (compound) value according to the statistical test.

After merging the values for each attribute according to the statistical test, the best attribute for splitting the node based on the adjusted p-value of the statistical test is selected. The adjusted p-value is calculated as the product of the p-value and Bonferroni multiplier. The Bonferroni multiplier calculates the number of ways in which n values can be split into r groups. For nominal attribute it is calculated by:

$$B = \sum_{i=0}^{r-1} (-1)^i \frac{(r-i)^n}{i!(r-i)!} \tag{2.11}$$

One branch is assigned to each value obtained after merging similar original values of the chosen attribute for splitting the node.

Exhaustive CHAID [47] is an improvement of algorithm CHAID. During the merging values of an attribute, exhaustive CHAID merge values of the attribute until only two compound values are left. Then it chooses the best split from the history of merging the groups of values and computes an adjusted p-value for it.

In the implementation of CHAID and Exhaustive CHAID algorithms, users can specify values for prepruning parameters which define when to stop growing the decision tree, same as for CART. In the implementation can be used techniques for boosting, bagging and cost-sensitive learning.

More details about the implementation of algorithm CHAID and Exhaustive CHAID can be read in [47].

SPRINT

SPRINT (Scalable PaRallelizable INduction and decision Trees) [78] is a decision-tree based algorithm that was designed with aims: (a) to overcome problems of decision-tree based algorithms which required that the whole or part of the training dataset be stored in memory and (b) to be easily parallelized. To accomplish these aims, SPRINT uses special data structures: one-time sorted attribute lists and class histograms. Those data structures are used for finding the best splitting of the node for each attribute. If the attribute lists can not fit in memory, they are placed on a disk. In the growing tree phase, SPRINT uses the Gini index as an impurity measure and binary splits of nodes. Minimum Description Length principle is used for the pruning.

Ensemble methods

Random Forests is an ensemble method that uses decision trees for building classification models. Random Forests can be used to reduce model variance. For training the decision tree models, a special vector of random values from a fixed probability distribution is generated in order to obtain less correlated models. Based on how the random values are used, different Random Forests approaches exist [79]:

- *Forest-random input selection (Forest-RI).* In the tree-growing phase for each model, F attributes are randomly selected from a set of attributes in a process of splitting the node. The best attribute for node splitting is determined by processing only F extracted attributes. Each decision tree is constructed without pruning in order to reduce bias. If F is small, the obtained trees are less correlated, but the less accurate trees are obtained. As a compromise, F is usually determined as $F = \log_2(k) + 1$, where k is the number of attributes in a dataset.

- *Forest-random linear combinations (Forest-RC)* is used when the number of attributes in a dataset is small. In order to increase the number of attributes, for each node, F new attributes are created as a linear combination of L randomly selected original attributes. The coefficients of original attributes in linear combinations are randomly generated from range $[-1, 1]$ using a uniform distribution. The best attribute for node splitting is chosen among the created F new attributes.

- At each node, the attribute for splitting is randomly chosen among the best F attributes.

XGBoost Tree is an ensemble method that uses gradient boosting technique with a decision tree algorithm (usually CART) as a base[12].

Information about the implementation of Random Forests and XGBoost Tree in software SPSS Modeler can be found in [47] and information about the implementation of Random Forests in Python library Scikit-learn can be found in [23].

Artificial neural networks

The artificial neural network classification model simulates the process by which the human brain learns[79, 3]. In short, the human nervous system consists of cells called neurons. Information is transmitted between neurons by special connections called synapses. The human brain learns by changing the strength of the synaptic connection between neurons caused by external stimuli.

Analogous to the human brain, artificial neural networks consist of calculation units called neurons. In general, each neuron has inputs, a calculation function, and an output[79]. The inputs receive data from other neurons and a weight is assigned to each input. Using the assigned weights , input values and the associated function, called activation function, the result is calculated and passed on to other neurons. Neurons are divided into layers. Neurons in the input layer receive instance data from the dataset and pass values to other neurons. The output layer contains one or more neurons that calculate the predicted values for the instance given as input. A neural network can contain zero or more hidden layers between the input and output layer. A hidden layer consists of one or more neurons called hidden neurons. Neural networks with hidden layers are called multilayer neural networks.

Based on how the neurons are connected in the hidden layers, neural networks can be divided into[79, 41]:

- *feed-forward* neural networks in which neurons of one layer are connected with neurons of the next layer, i.e. networks whose connections are acyclic. An example of a feed-forward neural network is a Multilayer Perceptron.

- *recurrent* networks in which neurons of one layer are connected with neurons from the same layer as with neurons from previous and next layers, i.e. networks with cyclic (recurrent) connections.

The feed-forward neural network allows only the flow and processing of information from input to output, without the possibility of returning the obtained output to the model and use of it in the following predictions. Loops in a recurrent neural network allow the use of information from previously read data in addition to the data currently being processed [19]. A recurrent neural network is suitable for sequence classification problem [42]. Elements of a sequence are processed one by one by a recurrent neural network, and the prediction of the element currently

being processed depends on the information of previously read elements within the sequence.

Multilayer Perceptron

The multilayer Perceptron is suitable for classification instances in a dataset with the fixed-length sliding window format. In the input layer, one neuron corresponds to one attribute. As activation function of neurons, the following functions are commonly used [79, 37]:

- Identity

$$f(x) = x \tag{2.12}$$

- Logistic sigmoid

$$f(x) = 1/(1 + e^{-x}) \tag{2.13}$$

- Hyperbolic tangent (tanh)

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.14}$$

- Rectified linear unit (relu)

$$f(x) = max(0, x) \tag{2.15}$$

- Scaled Exponential Linear Unit (SELU)

$$f(x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.16}$$

  where $\lambda = 1.05070098$ and $\alpha = 1.67326324$

Neuron with the assigned activation function f and d inputs $x = (x_1, x_2, ..., x_d)$ with corresponding weights $w = (w_1, w_2, ..., w_d)$ and bias b, calculates the output value by formula:

$$z = f(x * w + b) = f(\sum_{i=1}^{d} x_i * w_i + b) \tag{2.17}$$

For the multiclass classification problem with n classes, to each class c_i $(i = 1, 2, ..., n)$ is assigned a neuron which output z_{c_i} is an input to a output neuron. *softmax* function is commonly used as activation function of the output neuron:

$$\hat{y} = softmax(z_{c_1}, z_{c_2}, ..., ...z_{c_n}) = (\frac{exp^{z_{c_1}}}{\sum_{j=1}^{n} exp^{z_{c_j}}}, \frac{exp^{z_{c_2}}}{\sum_{j=1}^{n} exp^{z_{c_j}}}, ... \frac{exp^{z_{c_n}}}{\sum_{j=1}^{n} exp^{z_{c_j}}}) \tag{2.18}$$

softmax function transforms values to a vector which values are interpreted as the probabilities of categorical classes. The values of the vector are in the range [0,1] and their sum is 1. Class corresponding to the maximum probability value is assigned to an input instance. Illustration of a Multilayer Perceptron is shown in Figure 2.5.

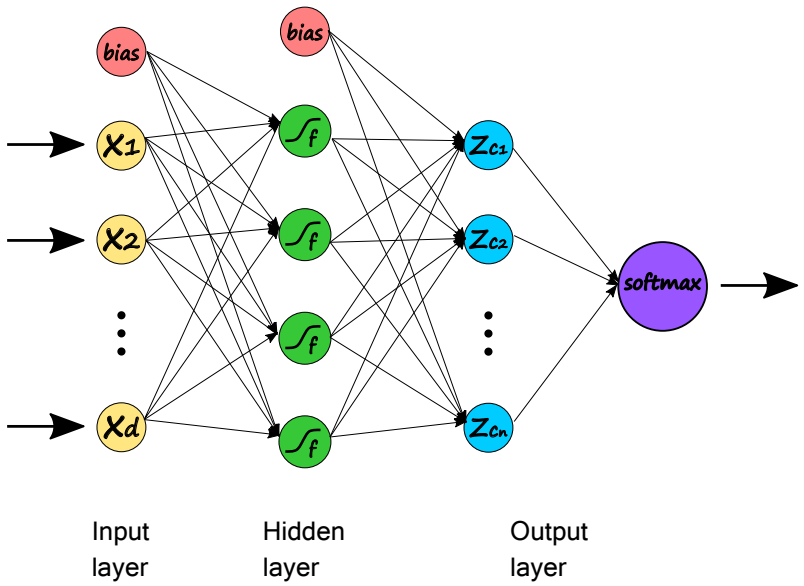

Figure 2.5: Illustration of structure of Multilayer Perceptron

When a model is trained using neural networks, the cost (loss) function must be defined. A model training aims to minimize the cost function by adjusting the weights in a neural network. In classification problems with multiple classes, cross-entropy loss is used as cost function:

$$Cross - entropy = -\frac{1}{N} \sum_{i=1}^{N} y_i * log(\hat{y}_i) \tag{2.19}$$

where N is the number of instances in a dataset, y_i is a vector of true probabilities of classes for an instance i, and $\hat{y}_i$ is a vector of estimated probabilities of classes by model. y is a vector with a value 1 for the true class of an instance and 0 for other classes.

Different gradient-based algorithms can be used for the optimization of the cost function, such as Adam [56]. For the computation of the weights, the back-propagation technique can be used.Each instance is processed in two phases in backpropagation technique[79]:

- *forward phase* computes $\hat{y}$ for instance being processed using previously obtained weights.
- *backward phase* updates the weights in order to improve the $\hat{y}$ for instance being processed. The weights are updated from the last layer to the first layer. Errors of neurons in layer $i+1$ are used to estimate errors and weight corrections of neurons in a layer i.

Detailed information about the implementation of Multilayer Perceptron in software SPSS Modeler can be found in [47] and information about the implementation of Multilayer Perceptron in Python library Scikit-learn can be found in [23].

LSTM-Bidirectional recurrent neural networks

Recurrent neural networks (RNNs) are suitable for sequence classification problem [42]. RNNs use the datasets in sequence data format for building models. Target values associated with instances are sequences of class labels. Each element in an input sequence is associated with one class label in a target sequence. In RNNs terminology, the element in an instance is called *timestep*. The elements from the input sequence are processed one by one. Depending on the architecture, the class of the element being processed can be determined in the same step, or after a delay of several steps, i.e. after processing the succeeding few elements. For each element e^t for $t = 1, ..., n$ in a sequence, the equations used in calculations in the forward phase can be summarized with [41]:

$$a^t = b + Wh^{t-1} + Ue^t \tag{2.20}$$

$$h^t = f(a^t) \tag{2.21}$$

$$o^t = c + Vh^t \tag{2.22}$$

$$\hat{y} = softmax(o^t) \tag{2.23}$$

where a is input to hidden nodes, h is output of hidden nodes (h^0 is initial state), o is input to output node, b and c are bias vectors, U is the weight matrix for connections between input nodes and hidden nodes, W is the weight matrix for connections between hidden nodes and V is the weight matrix for connections between hidden and output nodes. f denotes the activation function of hidden nodes, and $softmax$ is used as the activation function of the output node.

Recurrent connections enable the influence of elements on positions from 1 to $i-1$ in a sequence on the decision of the assigned class for an element on a position i in a sequence. If the information of following elements (on the position from $i+1$ to n) in a sequence can be useful for the prediction of a class label for element i, bidirectional recurrent neural network (BRNN) [77] can be used. BRNN provides information about all past and future elements in a sequence for predicting the class label at position i by using two recurrent hidden layers: (a) forward layer for processing elements in a sequence in the forward direction, i.e. from position 1 to position n, (b) and backward layer for processing elements in a sequence in the backward direction, i.e. from position n to position 1.

Nodes in the input layer are connected with neurons in the forward layer and neurons in the backward layer. Neurons from forward and backward layers are not connected. Neurons in a forward layer are connected with the neurons in the output layer, and neurons in a backward layer are connected with the neurons in the output layer. Neurons in the output layer combine the outputs from neurons in forward and backward layers.

The general structure of BRNN unfolded for three elements in a sequence is illustrated in Figure 2.6. A forward layer is presented with a blue rectangle and a backward layer is presented with a green rectangle.

Summarized steps in training the BRNN are proposed in [77]:

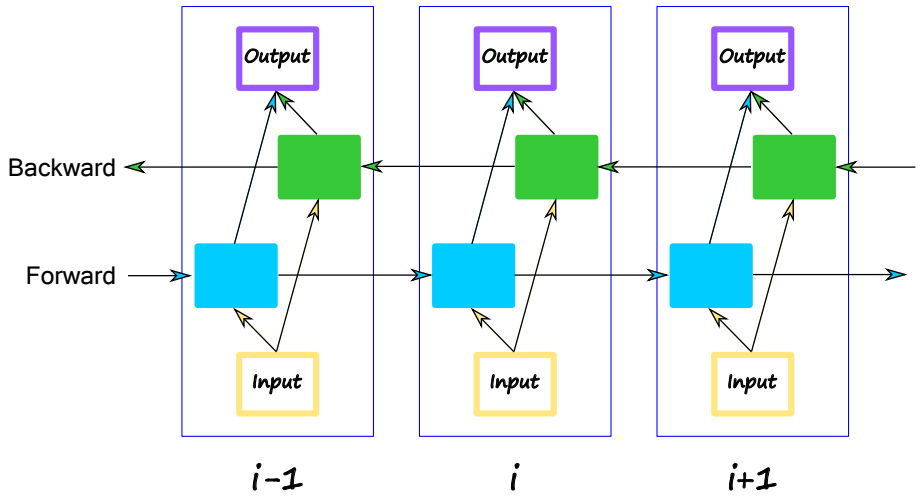

Figure 2.6: General structure of BRNN unfolded for three elements

1. Forward phase: Run all elements through the BRNN and calculate the output values.
   - Do forward phase for a forward layer for elements on a position from 1 to n.
   - Do forward phase for a backward layer for elements on a position from n to 1.
   - Do forward phase for the output layer.
2. Backward phase: Calculate the part of the objective function derivative for each element.
   - Do backward phase for the output layer.
   - Do backward phase for a forward layer for elements on a position from 1 to n.
   - Do backward phase for a backward layer for elements on a position from n to 1.
3. Update weights phase.

When propagating over many elements in a sequence, the problem which can occur is that gradients tend to either vanish or explode [41]. As a result, learning dependencies between distant elements in a sequence is difficult. Long Short-Term

Memory (LSTM) [45] architecture was proposed to overcome this problem. LSTM network is like RNN/BRNN, except that the LSTM network contains memory blocks instead of classical neurons in hidden layers.

Each memory block contains one or more self-connected memory cells and three units called gates (input, output and forget gates) which enable the storage and use of information on previously processed elements. An illustration of the memory block with one memory cell is shown in Figure 2.7.

Input, output and forget gates are summation units that receive data from and outside the block. Each gate has activation function h . The activation function is chosen so that the gate activations are between 0 and 1, where 0 denotes a closed gate and 1 an open gate. Cell input and output have activation gates g and h, respectively (see Figure 2.7). A cell doesn't have an activation function. Multiplication units (small black circles) of input and output multiply the input and output of the cell, while the multiplication unit of forget gate multiplies the cell previous state. Output from the block is the result of the output gate multiplication unit.

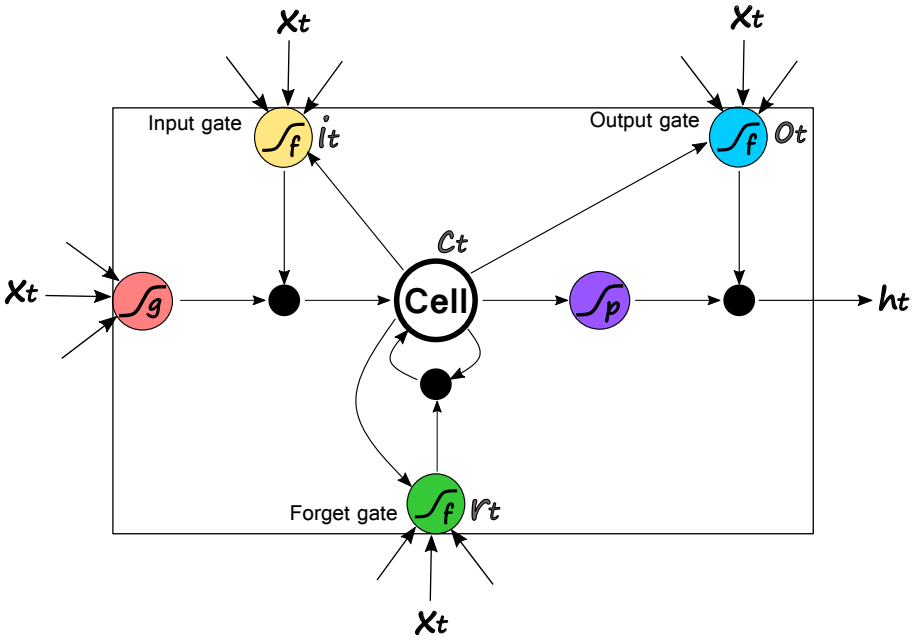

Figure 2.7: Memory block with one memory cell

Corresponding equations used in forward pass for element t (for $t = 1, 2, ..., n$) are [63, 40] :

$$i_t = f(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.24}$$

$$r_t = f(W_{xr}x_t + W_{hr}h_{t-1} + W_{cr}c_{t-1} + b_r) \tag{2.25}$$

$$c_t = r_t c_{t-1} + i_t g(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.26}$$

$$o_t = f(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.27}$$

$$h_t = o_t p(c_t) \tag{2.28}$$

where x_t is current input vector, f is activation function of gates, i, r, o and c are input gate, forget gate, output gate and cell activation vectors, respectively; h is hidden vector (output of a block) and b are bias vectors. W are weight matrices, where W_{kj} are weights of connections between k and j. g is a cell input activation function, and p is cell output activation function.

Accuracy of a RNN is calculated as:

$$Accuracy = 1 - \frac{1}{L}\sum_{i=1}^{n} hd(\hat{y}_i, y_i) \tag{2.29}$$

where n is the number of instances in a dataset, y_i is a target sequence of i-th instance, $\hat{y}_i$ is predicted sequence for i-th instance, hd is a hamming distance and L is the sum of lengths of target sequences ($L = \sum_{i=1}^{n} |y_i|$).

2.4 Clustering

Let a dataset D contains instances described by n attributes $X = (X_1, X_2, ..., X_n)$. In general, the clustering is the grouping of the instances from dataset D based only on the attributes X with the aim that the instances of one group are as similar to each other as possible and as different from instances of other groups as possible. One group of instances obtained by clustering is called a cluster.

Numerous clustering algorithms have been proposed. Some of them can be divided into the following groups based on their properties [79]:

- *Representative or prototype-based algorithms*: each cluster is described using a representative point. The representative point of a cluster is usually presented as the point with properties determined using cluster instances or is a selected instance from the cluster. Algorithms of this type require an iterative pass through a dataset in order to better determine the representative of a cluster. An instance is assigned to the cluster whose representative point is the most similar to it.

- *Hierarchical clustering algorithms*: a result of clustering is described with a hierarchical tree. Each tree leaf presents an individual instance, and the root of a tree contains all instances in a dataset, i.e. it presents one cluster with all instances. Each non-terminal node contains all instances from its child-nodes. There are two approaches for building a hierarchical tree: bottom-up and top-down. In a bottom-up approach, initially, each instance is a separate cluster. In each step, the two most similar clusters are merged. In the end, all instances are in one cluster (root-node). In a top-down approach, initially, all instances are in one cluster (root-node). In each step, a cluster with the least similar instances is divided into two clusters. In the end, each instance is a separate cluster. Usually, the number of desired clusters, that will be the result of clustering, is determined after the construction of the hierarchical tree.

- *Density-based algorithms*: the goal is to find dense regions of instances that are separated by space without instances or with sparse regions of instances. Each found dense region is a cluster.

- *Graph-based algorithms*: the dataset is presented as a graph. Instances are presented as nodes, and connections among instances are presented as links between nodes. A cluster is a group of nodes that are mutualy interconnected and not connected with other nodes.

Kohonen Self-Organizing Feature Map

Kohonen Self-Organizing Feature Map (SOFM or SOM) [57] is an algorithm based on neural networks and each cluster is described with a prototype. As a result of clustering, a neural network with a fully connected input and output layer is obtained. In the output layer, one node presents one centroid of a cluster

described as a weight vector of input nodes. The activation of an output node is the proximity of an instance and corresponding centroid. Nodes in the output layer are presented in a form of a lower-dimensional lattice (usually 2-dimensional). Each node is labelled with an n-tuple which presents the node's coordinates in a lattice. SOM maintenances the topology between input instances and neurons in an output layer, i.e. input instances that are similar will be assigned to nearby neurons in the lattice.

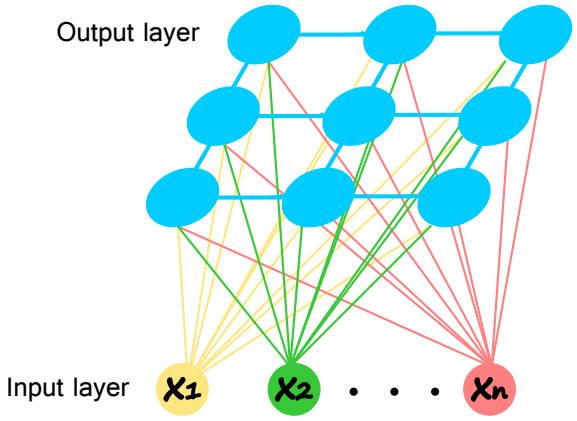

Figure 2.8: Example of Kohonen SOM

The main steps in the training phase of SOM are:

- Initialization of weights/centroids of neural network.
- Do in cycles until centroids do not change or the stop criteria (i.e. maximum number of cycles) is met.
  - For each instance i in a dataset, do:
    - Find the most similar centroid c to the instance i.
    - Update the weights of c in order to move it closer to i.
    - Update the weights of centroids that are nearby the centroid c based on the specified threshold about the neighbourhood.
- Return the obtained centroids of clusters.

For training the model using Kohonen SOM, following parameters should be defined:

- how to calculate proximity between an instance and centroids. Euclidean distance is comonly used, and calculated as:

$$Euclidean(i, c) = \sqrt{\sum_{j=1}^{n}(i_j - c_j)^2} \tag{2.30}$$

  where i is an instance, c is a centroid, and n is the number of input attributes by which the dataset is described.

- how is defined the neighbourhood of a centroid, i.e. nearby centroids which are updated along with the closest centroid. Chebychev distance is commonly used for the definition of a neighbourhood and is calculated as the maximum distance on any dimension of a grid on which centroids are placed.

- neighbourhood size. Centroids on the distance less than defined neighbourhood size from the closest centroid of an instance are updated, besides the closest centroid.

- η learning rate parameter which determines how closest centroids will be moved toward the instance being processed. The weights of the centroid are change based on the formula:

$$\delta_c = \eta(c - i) \tag{2.31}$$

  where c is a centroid and i is an instance being processed. η usually decreases as the number of completed cycles increases.

IBM SPSS Modeler supports Kohonen SOM[47].

TwoStep

Chiu and his team [13] developed a scalable algorithm that efficiently handles dataset with both numerical and categorical attributes. The clustering procedure has two phases:

- Pre-clustering of instances in order to find dense regions in terms of summary statistics. Finding dense regions of instances is done in one pass through dataset and information of them is saved as summary statistics called *cluster features* in a data structure called *CF-tree*. *CF-tree* is constructed in this phase. The leaves of *CF-tree* represent the dense regions. The non-leaf nodes are used as signposts to rout the instance to the dense region to which it belongs according to the properties. Each node is described with *cluster features* that contain the number of instances described by node, mean and variance of each numerical attribute, and counts for values in categorical attributes.

- Hierarchical clustering is performed on obtained dense regions in the pre-clustering phase. Obtained dense regions are initial clusters. The two most similar clusters are merged in each step until all dense regions are in one cluster. The decision which two clusters will merge in one step of hierarchical clustering is based only on the *cluster features*.

The distance measure used to determine the distance between two clusters is based on the log-likelihood function. Let a_i $(i = 1, 2, ..., w)$ be categorical attribute and b_j $(j = 1, 2, ..., q)$ be numerical attribut of a dataset, and C_l $(l = 1, 2, ...k)$ is a cluster. It is assumed that the numerical attributes are within-cluster C_l independent normal distributed. Mean of a numerical attribute b_j within cluster C_l is μ_{lj}, and its variance is σ^2_{lj}. Also, it is assumed that categorical attributes are indepentend and have normal distribution within cluster C_l.

The distance between two clusters C_r and C_s is defined as:

$$d(C_r, C_s) = \xi_r + \xi_s - \xi_{\langle r,s \rangle} \tag{2.32}$$

where $\langle r, s \rangle$ is an id of cluster obtained by merging clusters r and s and $\xi_g(g = r, s, \langle r, s \rangle)$ is calculated as

$$\xi_g = -n_g(\sum_{j=1}^{q} \frac{1}{2} \log(\hat{\sigma}_j^2 + \hat{\sigma}_{gj}^2) + \sum_{i=1}^{w} \hat{E}_{gi} \tag{2.33}$$

where n_g is the number of instances in a cluster g, $\hat{\sigma}_j^2$ is the estimated variance of the j-th numerical attribute for all instances, $\hat{\sigma}_{gj}^2$ is the estimated variance of the

j-th numerical attribute for instances in the cluster g, and $\hat{E}_{gi}$ is calculated as

$$\hat{E}_{gi} = -\sum_{v=1}^{m_v} \frac{n_{giv}}{n_g} \log \frac{n_{giv}}{n_g} \tag{2.34}$$

where n_{giv} is the number of instances in cluster g with the v-th value of the i-th categorical attribute.

TwoStep uses a two-phase procedure to automatically determine the best number of clusters when the minimal and maximal number of preferred clusters is defined.

In the first phase, Bayesian Information Criterion (BIC) is used for a coarse estimate of the optimal number of clusters. Since the models with small BIC are considered to be good models, in the first step is calculated the decrease of BIC for each merging step of hierarchical clustering. As the coarse number of clusters is chosen the k when a decrease in BIC starts to diminish. In the second phase, as a criterion for the best number of clusters is used the ratio change in distance, calculated for each merging step. Merging starts from the estimated number of clusters in the first phase (from k). The final number of clusters is chosen when a big difference in the ratio for two consecutive merges occurs. Instead of BIC, other criteria can be used.

IBM SPSS Modeler supports clustering with TwoStep[47].

3 Structural alphabets

3.1 Protein structure

Proteins are organic molecules that have significant and numerous roles in the functioning of an organism or cell. Some of the protein roles are participation in the cell structure , regulation of metabolism, protection of the organism from foreign substances, biological catalysis, transportation of molecules through cell membranes or blood, etc. By chemical composition, proteins are linear biological polymers composed of amino acids linked together by peptide bonds. The number and order of amino acids determine the protein structure and function [4, 14].
In general, an amino acid (see Figure 3.1) consists of amino group ($-NH_2$), carboxyl group ($-COOH$) and R group (residue) attached to the same central carbon atom (C_α).

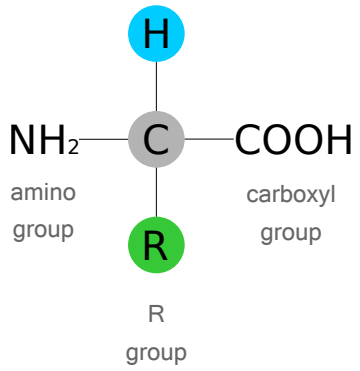

Figure 3.1: General structure of an amino acid

Proteins are commonly composed of twenty different amino acids, which have unique R groups. Depending on the composition of the R group, each amino acid has unique physicochemical properties, such as the polarity of the R group (non-polar R group, uncharged polar R group and charged polar R group), aromaticity

and bulk. Each amino acid has a corresponding three-letter and one-letter name. The list of twenty amino acids is shown in Table 3.1.

Amino acid	**Abbreviations**		**Molecular formula**	**Linear formula**
Alanine	Ala	A	$C_3H_7NO_2$	$CH_3 - CH(NH_2) - COOH$
Arginine	Arg	R	$C_6H_{14}N_4O_2$	$HN = C(NH_2) - NH - (CH_2)_3 - CH(NH_2) - COOH$
Asparagine	Asn	N	$C_4H_8N_2O_3$	$H_2N - CO - CH_2 - CH(NH_2) - COOH$
Aspartic acid	Asp	D	$C_4H_7NO_4$	$HOOC - CH_2 - CH(NH_2) - COOH$
Cysteine	Cys	C	$C_3H_7NO_2S$	$HS - CH_2 - CH(NH_2) - COOH$
Glutamine	Gln	Q	$C_5H_{10}N_2O_3$	$H_2N - CO - (CH_2)_2 - CH(NH_2) - COOH$
Glutamic acid	Glu	E	$C_5H_9NO_4$	$HOOC - (CH_2)_2 - CH(NH_2) - COOH$
Glycine	Gly	G	$C_2H_5NO_2$	$NH_2 - CH_2 - COOH$
Histidine	His	H	$C_6H_9N_3O_2$	$NH - CH = N - CH = C - CH_2 - CH(NH_2) - COOH$
Isoleucine	Ile	I	$C_6H_{13}NO_2$	$CH_3 - CH_2 - CH(CH_3) - CH(NH_2) - COOH$
Leucine	Leu	L	$C_6H_{13}NO_2$	$(CH_3)_2 - CH - CH_2 - CH(NH_2) - COOH$
Lysine	Lys	K	$C_6H_{14}N_2O_2$	$H_2N - (CH_2)_4 - CH(NH_2) - COOH$
Methionine	Met	M	$C_5H_{11}NO_2S$	$CH_3 - S - (CH_2)_2 - CH(NH_2) - COOH$
Phenylalanine	Phe	F	$C_9H_{11}NO_2$	$Ph - CH_2 - CH(NH2_) - COOH$
Proline	Pro	P	$C_5H_9NO_2$	$NH - (CH_2)_3 - CH - COOH$
Serine	Ser	S	$C_3H_7NO_3$	$HO - CH_2 - CH(NH_2) - COOH$
Threonine	Thr	T	$C_4H_9NO_3$	$CH_3 - CH(OH) - CH(NH_2) - COOH$
Tryptophan	Trp	W	$C_{11}H_{12}N_2O_2$	$Ph - NH - CH = C - CH_2 - CH(NH_2) - COOH$
Tyrosine	Tyr	Y	$C_9H_{11}NO_3$	$HO - Ph - CH_2 - CH(NH_2) - COOH$
Valine	Val	V	$C_5H_{11}NO_2$	$(CH_3)_2 - CH - CH(NH_2) - COOH$

Table 3.1: Molecular and linear formulas of 20 amino acids [1]

A polypeptide chain is formed by building a peptide bond ($C - N$) between two successive amino acids (see Figure 3.2). In a peptide bond, carbon from the carboxyl group of one amino acid shares electrons with nitrogen from the amino group of successive amino acid. When a peptide bond is formed, a water molecule is released. The amino acid sequence without residual parts is called *backbone* or *main chain*. The ends of the backbone are called N-terminus and C-terminus. N-terminus refers to the end with a free amino group, while the C-terminus refers to the end with a free carboxyl group.

The structure of a protein chain is described on three levels as primary, secondary and tertiary structure.

Protein primary structure is determined by a sequence of amino acids linked by peptide bonds. It is also called an amino acid sequence and is described as a sequence of one-letter amino acid names in the form $A_1A_2, ..., A_N$ where N is the number of amino acids in a chain. Amino acids in a sequence are counted from N-terminus to C-terminus.

Protein secondary structure is deifned by amino acid conformations in a region of a polypeptide chain. Its states are defined based on spatial patterns of backbone

[1]Source is http://www.imgt.org/IMGTeducation/Aide-memoire/_UK/aminoacids/formuleAA

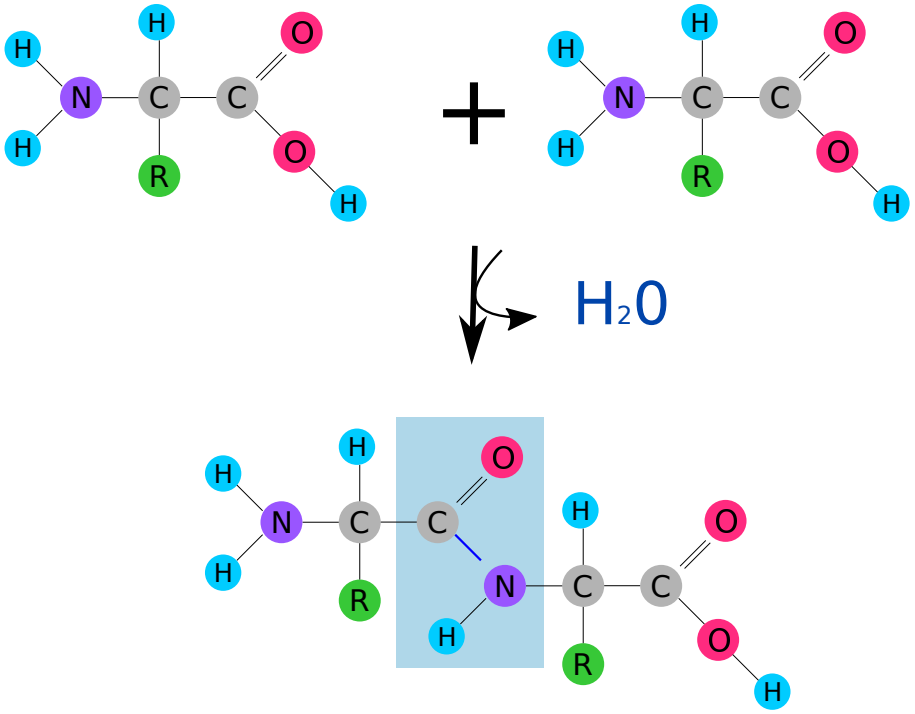

Figure 3.2: Peptide bond ($C-N$) between two successive amino acids

dihedral angles (ϕ and ψ) and the hydrogen-bonding patterns observed between carbonyl and amide groups in the peptide backbone. Conformation of protein backbone can be described with two dihedral angles (ϕ and ψ) per amino acid. Rotation around $N-C\alpha$ bond is represented by dihedral angle ϕ, and rotation around $C\alpha-C$ bond is represented by dihedral angle ψ (see Figure 3.3). Dihedral angle ϕ for amino acid at position i in a sequence is determined by coordinates of atoms $C_{i-1}, N_i, C_{\alpha i}, C_i$, while dihedral angle ψ is determined by coordinates of atoms $N_i, C_{\alpha i}, C_i, N_{i+1}$.

The standard way for visualization of dihedral angles ϕ and ψ of amino acids in a protein is by a Ramachandran plot [72]. In Figure 3.4 is the Ramachandran plot published by Lovell et al. in [33] and it shows the ϕ and ψ angles for 97,368 amino acids.

Secondary structure is described with eight states. Dictionary of Secondary Structure of Proteins (DSSP) program [53] can be used to assign one of the eight secondary structure states (3_{10}-helix (G), α-helix (H), π-helix (I), β-bridge (B), β-strand (E), bent (S), β-turn (T) and loop or irregular state (-)) to amino acids in a protein chain. In the studies, secondary structure is often described with three

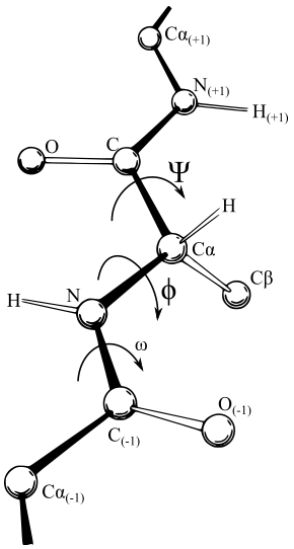

Figure 3.3: Illustration of dihedral angles ϕ and ψ [74]

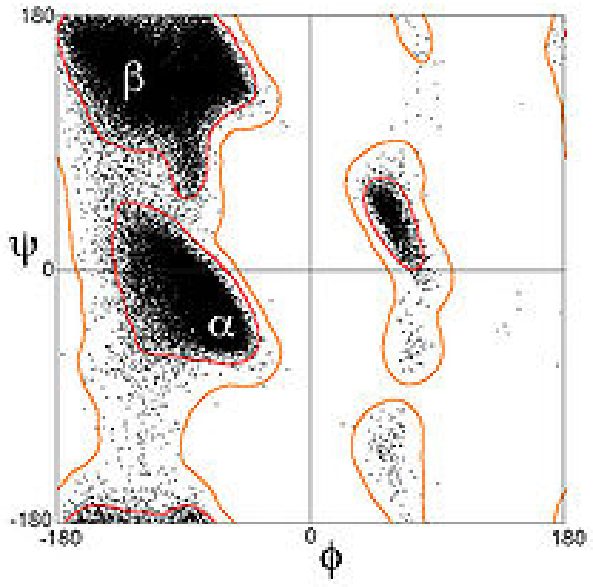

Figure 3.4: Ramachandran plot [33]

instead of eight states: two regular and most common states α-helix and β-sheet [18], and one non-regular state coil. In Figure 3.4, amino acids belonging to α-helix are recognized as cluster around $\phi = -57°$ and $\psi = -47°$, while amino acids belonging to β-sheet are recognized as cluster around $\phi = -130°$ and $\psi = +140°$.

The three-dimensional or tertiary structure (3D) of a protein is defined by the coordinates of the amino acid atoms of a polypeptide chain. Protein may consist

of one or more peptide chains. The complex 3D structure of all protein peptide chains is called quaternary structure. Protein Data Bank (PDB) [7] contains experimentally determined protein 3D structures.

Determination of the 3D structure of a protein from a secondary structure is not easy. The initial problem is the absence of the strict rules which define each of the secondary structure states [69]. A variety of programs based on different methods have been developed for the assignment of secondary structure states to the amino acids in a protein. As a consequence, the results of different programs for assignment the secondary structure states to amino acids in a protein can be very diverging.

To overcome the difficulties with secondary structure, a new approach for the description of the 3D structure of the backbone has been introduced [69]. The 3D structure of the backbone can be described using prototypes that approximate the local folds that occur in the structure of the protein. Prototypes are determined based on fragments of consecutive amino acids in polypeptide chains whose 3D structure is known. The local structure prototypes compose the library of local protein structures, also called the structural alphabet (SA).

Labels, which are usually letters, are assigned to prototypes of the structural alphabet. In terms of SA, the 3D structure of a protein backbone is represented as a sequence of prototype labels. For each L successive amino acids (fragment of length L) in an amino acid sequence from the N-terminus to the C-terminus, a corresponding prototype is determined. For most SAs, prototypes are assigned to overlapping fragments. Representation of the protein chain using SA prototypes can be viewed as a kind of a compression of the 3D structure of a protein chain backbone.

The first complex SA was developed by Unger et al. [82] in 1989, and they demonstrated that protein structures can be approximated by concatenating prototypes of the local structure. In the following years, more than 15 SAs have been developed to approximate local protein structures. SAs differ in the properties used to describe the protein backbone (coordinates, distances, torsion angles), methods used to define them (K-mean clustering algorithms, Kohonen maps, artificial neural networks, ...) and the number of amino acids in a fragment. Each SA is defined as a set of N prototypes of L amino acid length. For each SA, it is important that its prototypes enable precise reconstruction of the protein structure.

With the increase in the number of prototypes of the structural alphabet, approximation of the protein structure is better, but the accurate prediction of protein structure in terms of structural alphabet prototypes is more complicated. A detailed overview of the structural alphabets is given in [69].

A short description of some of the well-known structural alphabets is given in section 3.2 Overview of existing structural alphabets. A more detailed overview of SA Protein Blocks is given in section 3.3 Protein Blocks.

3.2 Overview of existing structural alphabets

Building Blocks

Unger and his team [82] were the first researchers who developed a complex structural library Building Blocks from proteins with experimental determined structures in 1989. They suggested that a structural library can be used to analyse and predict protein structure. Fragments of 6 consecutive amino acids were used for the definition of prototypes. Their study used a set of 82 proteins determined with X-ray. Overlapping fragments were extracted from only 4 proteins. Extracted fragments were used to determine prototypes. In order to calculate the distance between two fragments, the authors used the best molecular fit (BMF) algorithm by Nyburg [68] or Kabsch [52] for aligning fragments. After aligning, they calculated normalised root mean square deviation (*rms deviation*) of $C\alpha$ atomic positions. *rms deviation* was calculated by the equation:

$$\text{rms } C\alpha \text{ atomic positions} = \sqrt{\frac{\sum_{i=1}^{n} \delta_i^2}{n-2}} \tag{3.1}$$

where δ_i is a distance between two C_α atoms on i-th position in fragments.

Prototypes were selected using a variant of KNN clustering and division of obtained clusters into subclusters in which the maximum distance between members is 1Å. The total number of obtained subclusters was 103. The central fragments of obtained subclusters are prototypes, called building blocks. The usefulness of the obtained building blocks was verified by assigning the closest building blocks for each fragment in a set of 82 proteins. It was found that at least one block exists for 76% of fragments at a distance of 1Å, and for 92% of fragments at a distance of 1.25Å. For the first 60 amino acids of 71 proteins, the average *rms*

of $C\alpha$ atomic positions distance between the original proteins and reconstructed proteins was 7.3Å. Analysis of the relationship between secondary structures and building blocks showed that many blocks are associated with helices, sheets and turns, while some are associated with secondary structures' connections.

Hierarchical ascending clustering using distances

Rooman and co-workers [75] described automatic procedure based on hierarchical ascending clustering on fragments of fixed length between 4 and 7. To describe a fragment, they used distances between $C\alpha$ backbone atoms in a fragment. Dataset for clustering contained 75 proteins. As a distance measure between fragments, authors used *rms deviation* of the inter-$C\alpha$ distances calculated by the equation:

$$\text{rms inter-}C\alpha\text{ distances} = \sqrt{\frac{2}{n*(n-1)}\sum_{i=1}^{n-1}\sum_{j=i}^{n}(d_{ij}^{x}-d_{ij}^{y})^2} \tag{3.2}$$

where n is the length of a fragment, x and y are fragments, d_{ij}^{x} and d_{ij}^{y} are the Euclidean distances between C_α atoms of amino acid i and j in fragments x and y.

At first, the authors developed SAs with four prototypes using the clustering procedure and fragments of fixed length from 4 to 7 amino acids to compare them with secondary structure classes. Fragment with a minimal *rms deviation* in a cluster, considering all other fragments in the same cluster, was chosen as cluster prototype. In the analysis of obtained SAs using different fragment lengths, it was noticed that obtained clusters per SA had similar secondary structures of fragments: the first cluster was associated with helix, second with β-strand, third with all secondary structures except helix, and fourth with coil and turns. Later, the clustering procedure was used to demonstrate that structural families of fragments can be found. As an example, they described structural families obtained using a combination of clustering procedure and criteria on backbone dihedral angles on fragments of length 7. The number of fragments in a cluster was used as a criterion for cutting the hierarchical tree. The used threshold was 50 members. The obtained clusters were divided by using dihedral angles to obtain subclusters of fragments with dihedral angles in the same domains of the Ramachandran plot [86]. Some of the obtained prototypes corresponded to known secondary structures, while some prototypes presented new structural families.

Kohonen feature map with 100 structural motifs

Schuchhardt and co-workers [76] applied a self-organizing Kohonen map to identify local structural motifs. The study was performed on 9 amino acids long fragments from 136 proteins. Dihedral angles were used to describe fragments. As a distance measure between two fragments i_1 and i_2, root mean square deviation on angular values (*rmsda*) was used. *rmsda* was calculated by the equation:

$$rmsda = \sqrt{\frac{\sum_{k=1}^{n-1}(\psi_k^{i_1} - \psi_k^{i_2})^2 + (\phi_{k+1}^{i_1} - \phi_{k+1}^{i_2})^2}{2 * (n-1)}} \tag{3.3}$$

where n is a length of a fragment.

A neural network with an architecture of 10x10 neurons was used to generate the Kohonen feature map. Each neuron, which presented one local structural motif, was described with its position (x, y) in Kohonen map and 16-dimensional weight vector corresponding to ψ and ϕ angles, which can be visualized with $\psi - \phi$ plot (see Figure3.5). The obtained Kohonen feature map is shown in Figure 3.6.

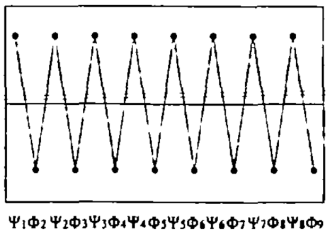

Figure 3.5: Example of $\psi - \phi$ plot of local structural motif [76]

Obtained structural motifs were compared with the ideal α-helix and β-strand motifs using *rmsda*. Ideal α-helix motif was presented using values $\phi = -57°$ and $\psi = -47°$, while ideal β-strand motif was presented using $\phi = -139°$ and $\psi = +135°$. The analysis showed the separation of motifs similar to α-helix from motifs similar to β-strand. Structure motifs similar to the ideal α-helix motif are located in the upper left corner, while motifs similar to β-strand are in the lower right corner.

A structural similarity measure was proposed for the comparison of two proteins in terms of constructed local structural motifs. Histogram can be calculated for each protein based on counting how often each neuron is active for that protein. The values were normalized in order to have a positive vector v with an absolute norm equal to one. For two proteins p^1 and p^2, similarity with range $[0, 1]$ was

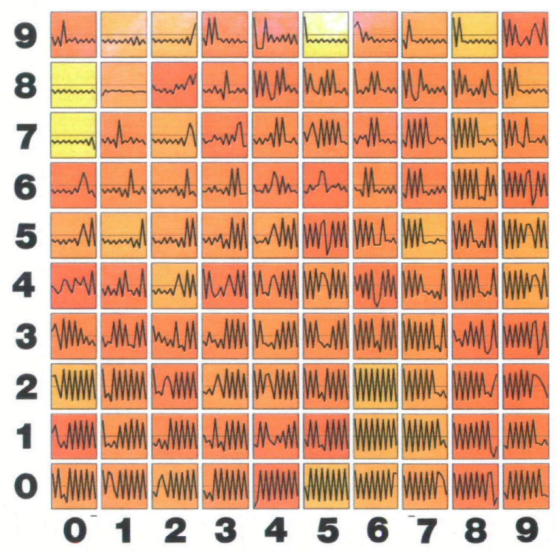

Figure 3.6: Kohonen feature map. Each cell presents one local structure motif shown by $\psi - \phi$ plot. The colours indicate the frequency of occurrence of an individual motif. Light yellow presents many instances and dark red a few instances. [76]

defined as $sim(p^1, p^2) = 1 - \frac{1}{2}\sum_{i=1}^{100} |v_i^{p^1} - v_i^{p^2}|$. Proteins with a similar proportion of the same local structural motifs have large similarity, while proteins with a different proportion of local structural motifs have a small similarity.

Structural Building Blocks

Fetrow and co-workers [38] developed SA Structural Building Blocks (SBBs) using autoassociative artificial neural network (autoANN) and K-means clustering. autoANN was used to reduce the dimensions of input data. Fragments of 7 amino acids from 116 proteins were used in the study. The fragments were described using geometry data: 15 $C\alpha$ distances between non-neighbouring amino acids, 5 virtual bond angles and 4 virtual dihedral angles of the $C\alpha$. Geometry data was encoded. Since $C\alpha$ distances had bimodal distributions, they were encoded with two attributes with range $[0, 1]$. The first encoded attribute was associated with the first mode of the distribution, and the second encoded attribute was associated with the second mode of the distribution. For each distance, the value of the encoded attribute associated with the mode to which distance belonged was set to a value proportional to the relationship between the distance and the range of the associated mode. If distance belonged to the first mode, the second encoded

attribute was set to 0, and if distance belonged to the second mode, the first encoded attribute was set to 1. Virtual bond angles were normalized to the range $[0, 1]$. Each virtual dihedral angle was presented with two attributes: one for the *sine* and one for the *cosine* of the angle. Attributes with *sine* and *cosine* of the angles were normalizes to the range $[0, 1]$.

Autoassociative artificial neural network is a machine learning algorithm that trains a network to reproduce values of input layer as values of output layer using a hidden layer with a smaller number of nodes then input layer [59]. Since the output layer activations depend on the hidden layer activations, the input data's important properties are encoded in the hidden layer activation. Authors developed the autoANN with 8 neurons in the hidden layer, by which the properties of the fragment described with 43 attributes were encoded with 8 attributes. Encoded values of fragments were clustered using K-means algorithm and Euclidean distance into 6 clusters. Each cluster presented one structural building block. Secondary structures were assigned to amino acids in proteins using DSSP. In the comparison of the SBBs and frequency of secondary structures at each amino acid position, it was observed that two SBBs corresponded to helix and sheet classes, two SBBs were associated with helix caps, and two SBBs were associated with strand caps.

I-sites

Bystroff and Baker [10] used a combination of sequence-based and structure-based clustering in order to obtain a structural alphabet. First, they used the algorithm K-means for sequence-based clustering. The most frequent structure in terms of backbone torsion angles in a cluster was defined as cluster representative, which the authors called structure *paradigm*. The sequence profile (log odds scoring matrix) was calculated for each cluster using its instances. Using the iterative procedure, cluster instances with a different structure from a *paradigm* were excluded, and the profile was updated. After updating a profile, a dataset was searched for new instances of a cluster. Using 471 protein sequences for training, fragments of length from 3 to 15 were extracted and used for clustering. In a sequence-based clustering authors used similarity measure D_{pq} calculated by the equation:

$$D_{pq} = \sum_{ij} \log[\frac{P_{ij}(p) + \alpha * F_i}{(1+\alpha) * F_i}] * \log[\frac{\sum_{k \in q} P_{ij}(k) + \alpha' * F_i}{(N_q + \alpha') * F_i}] \tag{3.4}$$

where P is weighted amino acid frequency profile for each position in the dataset; $P_{ij}(p)$ is frequency of amino acid i in position j of fragment p; N_q is the number of fragments in a cluster q; F_i is the frequency of amino acid i in the dataset; $\alpha = 0.5$ and $\alpha' = 1.5$ were determined empirically. As a structural similarity between fragments, the authors used a combination of RMS distance matrix error (dme) and the maximum deviation in backbone torsion angles (mda). For two fragments $f1$ and $f2$, dme was calculated by the equation:

$$dme = \sqrt{\frac{\sum_{i=1}^{L} \sum_{j=i-5}^{i+5} (\alpha_{i\to j}^{f1} - \alpha_{i\to j}^{f2})^2}{N}} \tag{3.5}$$

where L is the length of fragment $f1$, $\alpha_{i\to j}$ is the distance between $C\alpha$ atoms of amino acid at position i and amino acid at position j in a fragment; and mda was calculated by the equation:

$$mda(L) = \max_{i=1,L-1} (\Delta\phi_{i+1}, \Delta\psi_i) \tag{3.6}$$

The obtained number of clusters was 82 and the structural library was called I-sites. I-sites library was used for the prediction of the local protein structure based on sequence-sequence profile comparison. To calculate the prediction accuracy, the authors compared the torsion angles of 8 amino acid fragments with the true values. They considered that the prediction was correct if the mda value was less than 120° or if the $rmsd$ was less than 1.4Å. The accuracy of prediction on 55 test proteins was 50%.

Recurrent Oligomers

Micheletti and co-workers [66] used knowledge-based clustering on fragments of 75 proteins for their research about local motifs. Fragments of length l were described using C_α coordinates. Coordinate root mean square deviation ($cRMS$) was used as a distance measure between two superimposed fragments f_1 and f_2 with the Kabsch procedure. $cRMS$ was calculated by the equation:

$$cRMS = \sqrt{\frac{\sum_{i=1}^{l} |\overrightarrow{r}_k^{C_\alpha}(f_1) - \overrightarrow{r}_k^{C_\alpha}(f_2)|^2}{N}} \tag{3.7}$$

where $\overrightarrow{r}_k^{C_\alpha}(f_i)$ presents C_α coordinates of k-th amino acid in a fragment f_i.

Algorithm 2: Cluster procedure for oligomers

Data: F: set of fargments of length l; *crms_ cutoff*: distance cutoff for calculating proximity score of a fragment
Result: R: set of representative fragments ordered by proximity score
while *While F is not empty* **do**
  foreach $f \in F$ **do**
    calculate $f_{\text{proximityscore}}$ as the number of neighbours within a *crms_ cutoff* distance
  end
  Find a fragment $r \in F$ with the largest $f_{\text{proximityscore}}$
  Declare r as a representative fragment and add it to R
  Remove the representative fragment r and all it neighbours within a *crms_ cutoff* distance from F
end

The procedure used for clustering fragments of length l is described in Algorithm 2.

The authors used a histogram of distances between all fragments of length l to obtain information about clusters in a dataset. Using the distribution of pair-distances, they extracted information about the number of clusters and typical distance within clusters. If a pair-distance histogram has a form of a bell, no cluster exists in the dataset, while a histogram with clusters has two peaks: a smaller peak represents the average distance in a cluster, and a larger peak represents the average distance between two clusters. In the analysis, the authors used fragments of length between 3 and 10. For each length of fragments except for length 3, the histogram showed two peaks. With the increasing length of the fragments, the height of the first peak decreases, which led the authors to conclude that each fragment represents a separate cluster for very large l. By the analysis of histograms, they concluded that the intra-cluster distance is about 0.65Å for all l, so they use it as a *crms* cutoff in a clustering procedure.

Using clustering procedure for fragment lengths 4, 5, 6, and 7, authors extracted 28, 202, 932 and 2561 representative fragments (prototypes), respectively. Prototypes were called oligons. Using the obtained prototypes for protein reproduction, the authors showed that proteins could be represented with a few tens of fragment prototypes of length 5 or 6 with 1Å*crms* per amino acid.

For the analysis or correlation between amino acids and oligons, authors subdivided amino acids into four groups (Gly, Pro, hydrophobic and polar amino acids)

and mapped amino acids in fragments to the appropriate group. It was shown that a correlation between transformed fragments and oligons existed.

Simulated-annealing K-means

Kolodny and co-workers [58] constructed and analysed different size structural alphabets based on fixed-length fragments in 200 proteins in order to find structural alphabets that are accurate and with low complexity. Simulated-annealing K-means method was used for clustering on non-overlapping fragments. The length of fragments was between 4 and 7 amino acids. The size of detailed described libraries was from 4 to 250 prototypes. Fragments of length l were described using C_α coordinates and *crms* (equation 3.7) was used as a distance measure between two fragments. Fragment with the minimal sum of *crms* among fragments in a cluster was used as cluster centroid. Obtained SAs were compared by accuracy and complexity.

Accuracy was measured as the ability of SA prototypes for local-fit and global-fit approximations on test proteins that were not used in clustering. Local-fit was measured as the average *crms* of all fragments of test proteins and their best-fit prototype in SA. Global-fit was measured as average *crms* of the reconstructed 3D protein structure using prototypes of SA and protein native structure. Authors defined the complexity of a library as $s^{\frac{1}{(l-3)}}$ were s is a size of a SA, and l is the length of fragments. Complexity presents the average states per amino acid in a protein. The complexity of obtained SAs varies between 3.16 and 15 per amino acid.

The average local *crms* was below 0.85Å for all obtained SAs, and the average global *crms* was below 2.57Å . The best global *crms* was 0.76Å and was obtained for SA with fragments of length 5 and size of 225.

Based on the analysis, the authors stated the following conclusions:

- Among SAs of the same length, accuracy of local-fit approximation increase with the size of SA;
- Among SAs of the same complexity, accuracy of local-fit approximation decrease with the length of SA;
- Among SAs of the same complexity, accuracy of global-fit approximation increase with the length of SA;

- Longer fragments are more accurate since they include more correlation than shorter fragments;
- Extensive datasets are needed to make reliable SAs for the longer fragments.

3.3 Protein Blocks

One of the most used structural alphabets is Protein Blocks (PBs) [15, 36]. De Brevern and his team constructed PBs to approximate the structural patterns that exist in protein backbones and to predict local 3D structure of the backbone from the amino acid sequence in the terms of PBs. They used overlapping fragments of length 5 described with dihedral angles (ψ, ϕ). As a distance measure between fragments, *rmsda* (equation 3.3) was used. In the first study [15], a dataset consisted of 228 proteins was used for prototypes construction. The same procedure was applied on a dataset consisting of 400 proteins in the second study [36].

The clustering of fragments was based on the Kohonen map and had two phases. In the first phase of clustering, only dihedral angles were considered. In the second phase of clustering, transitions between prototypes (protein blocks) obtained in the first phase were considered. More precisely, the clustering procedure had the following steps:

1. Kohonen network was trained in C cycles (user-defined parameter) using fragments described with eight dihedral angles $\psi_{-2}, \phi_{-1}, \psi_{-1}, \phi, \psi, \phi_{+1}, \psi_{+1}, \phi_{+2}$ and *rmsda* as distance measure to obtain B (user-defined parameter) protein blocks.

2. Proteins in the training set were encoded using the obtained protein blocks from step 1. Transitions matrix between PBs was calculated using the frequencies of the pairs of consecutive PBs observed in encoded proteins. Training of the Kohonen network was continued by reading consecutive fragments in proteins. During the processing of a fragment i, its n (user-defined parameter) structurally closest protein blocks by *rmsda* were found. Then, among the n structurally close protein blocks, one with the highest transition frequency for the protein block assigned to a previously fragment in a protein (fragment *i-1*) was chosen. This step was repeated in C cycles.

3. Optimal number of prototypes was determined using structural similarity and transition similarity between pairs of PBs. Structural similarity between two PBs obtained in the previous step was calculated using *rmsda*, and transition similarity between two PBs was calculated using transition probabilities of processed PBs to other PBs. The similarity threshold was a user-defined parameter. In cycles, from pairs of similar PBs, one PB was deleted. At the end, the two similar PBs did not exist.

As a result of the described procedure, 16 prototypes (PBs) were selected. PBs are labeled with letters, from *a* to *p*. Each PB is defined by eight dihedral angles $\psi_{-2}, \phi_{-1}, \psi_{-1}, \phi, \psi, \phi_{+1}, \psi_{+1}, \phi_{+2}$. Associated dihedral angles to each PBs are shown in Table 3.2 and representation of each PB is shown in Figure 3.7.

PB	ψ_{-2}	ϕ_{-1}	ψ_{-1}	ϕ	ψ	ϕ_{+1}	ψ_{+1}	ϕ_{+2}
a	41.14	75.53	13.92	-99.80	131.88	-96.27	122.08	-99.68
b	108.24	-90.12	119.54	-92.21	-18.06	-128.93	147.04	-99.90
c	-11.61	-105.66	94.81	-106.09	133.56	-106.93	135.97	-100.63
d	141.98	-112.79	132.20	-114.79	140.11	-111.05	139.54	-103.16
e	133.25	-112.37	137.64	-108.13	133.00	-87.30	120.54	77.40
f	116.40	-105.53	129.32	-96.68	140.72	-74.19	-26.65	-94.51
g	0.40	-81.83	4.91	-100.59	85.50	-71.65	130.78	84.98
h	119.14	-102.58	130.83	-67.91	121.55	76.25	-2.95	-90.88
i	130.68	-56.92	119.26	77.85	10.42	-99.43	141.40	-98.01
j	114.32	-121.47	118.14	82.88	-150.05	-83.81	23.35	-85.82
k	117.16	-95.41	140.40	-59.35	-29.23	-72.39	-25.08	-76.16
l	139.20	-55.96	-32.70	-68.51	-26.09	-74.44	-22.60	-71.74
m	-39.62	-64.73	-39.52	-65.54	-38.88	-66.89	-37.76	-70.19
n	-35.34	-65.03	-38.12	-66.34	-29.51	-89.10	-2.91	77.90
o	-45.29	-67.44	-27.72	-87.27	5.13	77.49	30.71	-93.23
p	-27.09	-86.14	0.30	59.85	21.51	-96.30	132.67	-92.91

Table 3.2: Protein Blocks reference angles [15])

To calculate average *rmsd* in clusters, C_α coordinates for each prototype were constructed using associated dihedral angles. The most frequent PB *m* had the smallest *rmsda* (15°), while the PB *j* with the smallest frequency had the largest *rmsd* (0.83Å). The average *rmsda* in obtained clusters was 30.1°.

Based on the analysis of the relationship between PBs and 3-state secondary structures of their amino acids, the authors made a coarse classification of PBs.

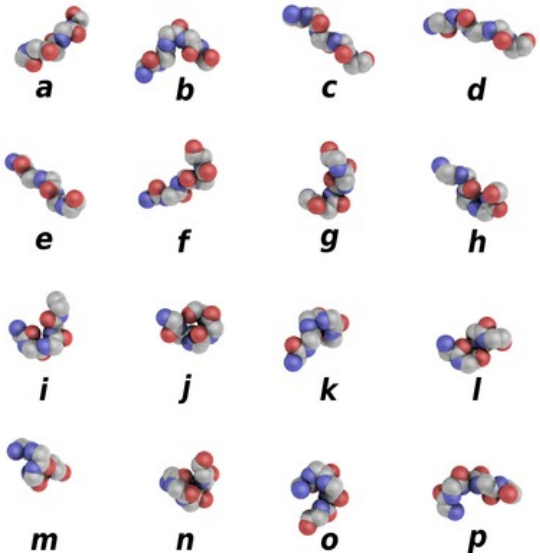

Figure 3.7: Representation of 16 Protein Blocks obtained with clustering procedure [27]. Carbon atoms are represented in grey, oxygen atoms in red and nitrogen atoms in purple.

The most frequent PBs m and d correspond to regular secondary structures α-helix and β-sheet, respectively. Less frequent PBs (from g to j) correspond to the coil, while the others correspond to caps of α-helix and β-sheet. The frequency, average *rmsd*, average *rmsda* and corresponding secondary structure for each PB obtained in [36] are shown in Table 3.3.

Protein with a known 3D structure can be translated to a sequence of PBs by assigning a PB to every five consecutive amino acids in a chain using *rmsda* criteria (see Figure 3.8 for the illustration of translation). To each fragmet of five consecutive amino acids, with centre amino acid at position i and dihedrals $(\psi_{i-2}, \phi_{i-1}, \psi_{i-1}, \phi_i, \psi_i, \phi_{i+1}, \psi_{i+1}, \phi_{i+2})$, a PB with the lowest *rmsda* is assigned. Besides the 16 PBs, the PB sequence of a protein can contain a letter Z at any position i which indicates that for a fragment of five amino acids with the centre amino acid at position i dihedral angles can not be calculated. Coding of protein chains to a sequence of PBs based on PDB data can be done using the Pbxplore tool [27].

Each amino acid in a protein is associated with five protein blocks (except the first two and the last two amino acids in a sequence). Approximate angles of an amino acid are calculated as the average value of the associated angles from the five protein blocks.

PB	frequency(%)	rmsd(Å)	rmsda (°)	secondary structure
a	3.89	0.46	45.2	N-cap β
b	4.41	0.47	42.5	N-cap β
c	8.12	0.51	38.4	N-cap β
d	18.85	0.41	29.7	β
e	2.45	0.71	40.9	C-cap β
f	6.68	0.40	37.5	C-cap β
g	1.15	0.60	50.6	mainly coil
h	2.40	0.46	47.0	mainly coil
i	1.86	0.41	43.4	mainly coil
j	0.83	0.83	49.0	mainly coil
k	5.45	0.3	35.9	N-cap α
l	5.46	0.53	32.5	N-cap α
m	30.22	0.31	15.0	α
n	1.99	0.31	26.8	C-cap α
o	2.77	0.48	38.3	C-cap α
p	3.47	0.47	43.8	C-cap α / N-cap β

Table 3.3: Description of Protein Blocks [36]. For each protein block, its frequency, average *rmsd* between its constructed C_α coordinates and cluster members in a training set, average *rmsda* between its dihedral angles and dihedral angles of cluster members in a training set and coarse classification based on corresponding secondary structures in which protein block most frequently occurs are presented.

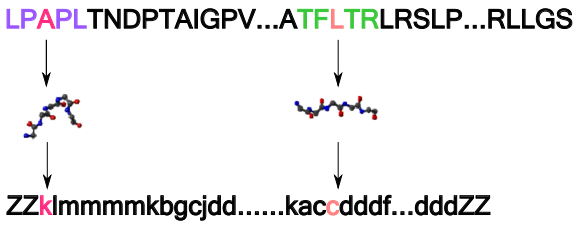

Figure 3.8: Illustration of translation of the amino acid sequence to PB sequence

Besides for the description of the 3D protein backbone, SA Protein Blocks is applied in other studies in bioinformatics, such as superimpose protein 3D structures [65, 22, 25], mine protein structures [65], define binding sites [17], and also to analyse local conformation of disorder proteins (or regions) [30].

Predictors for Protein Blocks

Several predictors of Protein Blocks from an amino acid sequence have been developed since 2000.

Bayesian Probabilistic Approach

De Brevern and his team developed the first PBs predictor as a part of the development process of SA Protein Blocks [15]. Their PBs predictor was based on the Bayesian probabilistic approach. Proteins in the dataset used for the construction of SA PBs were coded into PBs sequences. The occurrence matrix of amino acids for sequence window of length 15 (7 amino acids on the left and 7 amino acids on the right of amino acid to which PB is assigned) was calculated for each PB. Conditional probability $P(aa_i \text{ in } j/PB_y)$ was calculated for each of twenty amino acids (aa_i, $i \in [1, 20]$, PB y (y is from a to p) and position j in a sequence window ($j \in [1, 15]$).

In the PBs prediction process for a given amino acid sequence, for each sequence window X_s (for an amino acid in a position s in amino acid sequence, seqence window is $(aa_{s-7}, , ..., aa_{s-1}, aa_s, aa_{s+1}, ...aa_{s+7})$) probability of belonging central amino acid (aa_s) to a PB y was calculated using Bayes' theorem by the equation:

$$P(PB_y/X_s) = \frac{P(X_s/PB_y) * P(PB_y)}{P(X_s)} \tag{3.8}$$

$P(PB_y)$ is the probability of occurrence of PB y in a training dataset. $P(X_s)$ is the probability of occurrence of sequence X_s in a training dataset, calculated as a product of frequencies of its amino acids. $P(X_s/PB_y)$ is a conditional probability of occurrence of window sequence X_s in window sequences in which central amino acid belongs to PB_y. $P(X_s/PB_y)$ was calculated by the equation:

$$P(X_s/PB_y) = \prod_{j=-7}^{j=7} P(aa_j/PB_y) \tag{3.9}$$

where $P(aa_j/PB_y)$ is a probability of occurrence of amino acid of type aa at the position j in window sequences belonging to PB_y and can be calculated using occurrences matrices.

To determine the optimal PB for a sequence window X_s, the authors used the ratio R_y defined as

$$R_y = \frac{P(PB_y/X_s)}{P(PB_y)} = \frac{P(X_s/PB_y)}{P(X_s)} \tag{3.10}$$

For a window sequence X_s, a PB with the highest R_y was chosen as optimal. The accuracy (Q_{16}) of the PBs prediction on the test set was 34.4%.

To improve the prediction of PBs, the authors developed *sequence families*, which are based on the idea that one PB may be associated with different types of sequences (*1 Protein Block - n Sequences*). Using clustering procedure, a set of sequences corresponding to each protein block was further divided into f groups. For each sequence family l of a protein block y, occurrence matrix PB_y^l was calculated. Briefly, the clustering procedure had the following steps:

1. For each PB_y and its group of sequence fragments (l), occurrence matrix PB_y^l was initialized with the amino acid frequencies of PB_y. In order to obtain different matrices per group, random noise was added to each.

2. Each sequence fragment of PB_y was compared with each obtained occurrence matrix PB_y^l and score R_{y^l} was calculated. Sequence fragment was assigned to a sequence family with the max score. The occurrence matrix of the sequence family with the best score was updated, taking into account the processed fragment's amino acids.

Using sequence families, the Q_{16} of the prediction was 40.7%. Definition of sequence families was further improved in [36] by using the larger database (450 proteins) and Q_{16} as an additional criterion in the learning and selection of the optimal sequence families. The number of sequence families per PB was between 1 and 6. The Q_{16} of the prediction was improved to 48.7%. Calculated Q_{16} and Q_{14} for used Bayesian approaches in [15, 36] are shown in Table 3.4. Q_{14} is the accuracy of predictor without taking into account two most frequent Protein Blocks (PBs m and d).

Dual-layer model

Dong and his team designed a dual-layer model for PBs prediction [16]. PSSMs generated by PSI-BLAST for proteins in the dataset were used as input to a model. One instance in a dataset is PSSM information of a window sequence of length

Accuracy	Bayesian approach without SFs	Bayesian approach with SFs	Bayesian approach with improved SFs
Q_{16} (%)	34.4	40.7	48.7
Q_{14} (%)	-	36.5	37.4

Table 3.4: Accuracy of the PBs prediction using Bayes approaches [15, 36]

15. Each amino acid in a window sequence was described with 21 values (one per 20 amino acid types and one indicator of whether the amino acid is beyond the chain). The target PB of an instance corresponds to a central amino acid in a window. The first layer of a model is called the sequence-to-structure layer because it classifies the sequence information into local structures. The second layer of a network is called the structure-to-structure layer because it classifies the first layer's outputs into PBs. The first-layer classifier's output is a vector of length 16, and each value in a vector presents the likelihood that the central amino acid in a fragment belongs to a particular prototype. The output of the first-layer is an input to the second layer of a model.

Feed-forward back-propagation neural networks with a single hidden layer were used as classifiers in a model (one network per layer). The number of units in a hidden layer in a first-layer classifier was 100, and in a second-layer classifier was 80. Classifiers were trained using five-fold cross-validation, momentum term 0.9 and learning rate 0.005. The model was trained using overlapping fragments from 1400 chains. Obtained prediction accuracy for protein blocks was 58.5%.

LOCUSTRA

Zimmermann and his team [88] also developed a two-layer method for PBs prediction. Their model is called LOCUSTRA, and it is based on support vector machines. PSSMs generated by PSI-BLAST were used as input to the first layer. Like in a dual-layer model [16], each amino acid in a sequence window of length 15 was described with 21 values. A target PB of an instance corresponds to a central amino acid in a window. The first layer of a method is composed of pairwise-coupling classifiers. In pairwise coupling, the positive class contains fragments of one PB while the negative class contains fragments from one of the other PBs. The second layer is composed of one-per-class classifiers, which use the first layer's output as input. In a one-per-class approach, the positive class contains fragments of one PB, while the negative class contains fragments from all other PBs. Au-

thors used SVM algorithms implemented in LIBSVM software [11]. ν-SVM was used to train the first-layer classifiers, and C-SVM was used to train the second-layer classifiers. For the PBs prediction for a given window sequence as input, the following steps are applied:

1. PB with the highest number of votes from the classifiers of the second-layer is assigned.
2. If more than one PB have the highest number of votes and PB d is among them, then PB d is assigned.
3. If PB d is not among the PBs with the highest number of votes, PB m is assigned.

The dataset contained 1,112 protein chains for training and 222 chains for testing. Achieved prediction Q_{16} on a test set was 61%.

svmPRAT

Rangwala and his team [73] developed a svmPRAT toolkit for resolving general residue-wise classification or regression problems. Problems must be described in the form of feature matrices, i.e. amino acid at each position in a sequence must be described using a fixed-length vector. Also, the user can define the number of neighbours (which is used to define the length of a window sequence), which must be considered during the processing of an amino acid in a sequence. svmPRAT consists of two programs: svmPRAT-L for learning models and svmPRAT-P for prediction based on the learned model.

For the classification problem, svmPRAT-L learns one-versus-rest binary classification models based on support vector machines. svmPRAT includes several kernel functions. Obtained models from svmPRAT-L predict each class's likelihood, and a class with the highest likelihood is assigned to a given instance as output. The construction of dual-layer models is also provided in svmPRAT.

The toolkit was applied for the prediction of Protein Blocks on a dataset with 1600 proteins. PSSMs generated by PSI-BLAST and predicted secondary structures by YASSPP [54] were used as input. Models were trained using kernel functions *soe* (normalized second-order exponential kernel) and *rbf* (radial basis function) and different sequence window length. The best achieved Q_{16} was 68.9% for sequence window of length 19.

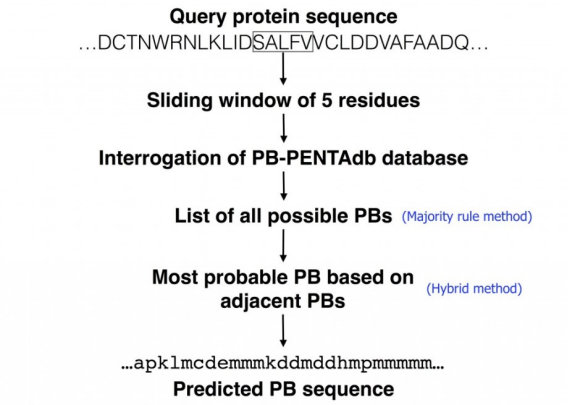

Figure 3.9: General scheme of a PBs prediction by PB-kPRED [2]

PB-kPRED

A fragment and knowledge-based approach, called PB-kPRED, was proposed by Vetrivel and his team [26]. The authors created a database PENTAdb, which contains pentapeptides extracted from 274,920 protein chains and their corresponding PBs. For PBs prediction, PB-kPRED searches PENTAdb. Prediction of PBs is based on querying the PENTAdb for each fragment of 5 amino acids in a given sequence.

Two methods were defined to chose the optimal PBs among the returned PBs for a query fragment: the majority rule method and the hybrid method. The majority rule method chooses the most frequently observed PBs. The hybrid method considers amino acids of three consecutive PBs of which the central block corresponds to a query fragment in order to take into account information about the local environment. Table with normalized frequency (odds) for consecutive three PBs were calculated. For each 5 consecutive amino acids in a query fragment of length 7 (which defines three consecutive PBs), all found PBs in the database are returned. The score is calculated for each central PB among returned PBs as $S_1 * S_2$. S_1 is the number of occurrences of central PB and S_2 is the sum of the odds calculated for each returned combination for a processed central PB. Central PB with the highest score is optimal. The general scheme of a PBs prediction by PB-kPRED is shown in Figure 3.9.

PBs prediction was performed on a query dataset of 15,544 proteins using pro-

[2]Source is `http://www.bo-protscience.fr/kpred`

teins with different sequence identity cut-offs from PENTAdb in order to evaluate the influence of homologues on the quality of the prediction (from 30% to full database). Achieved average accuracy was 66.31%.

4 New Protein Blocks predictors

An overview of published methods for the prediction of Protein Blocks from an amino acid sequence is given in chapter 3. Described PBs predictors can be divided into three groups according to the used approaches and sequence information used as inputs to the predictors:

- PBs predictors based on Bayesian approach [15, 36] which use amino acid sequence and defined sequence families.
- Predictor PBk-PRED [26] based on fragment and knowledge-based approach which searches for compatible sequences in a database of pentapeptides from protein structures. PBk-PRED can be applied only if compatible sequences can be found.
- PBs predictors based on different machine learning approaches [73, 88, 16], mainly Support Vector Machines, which use evolutionary information of a chain in a form of PSSM matrix obtained using PSI-BLAST program [34].

The best-reported Q_{16} of these PBs predictors is 68.9% for SVMprat tool [73].

As part of the thesis, several PBs prediction models were developed using different data mining approaches and machine learning algorithms. Different information about the amino acid sequence in comparison with the available PBs predictors was used. The results described in this chapter are submitted for review [64].

4.1 Material

Protein chains in used dataset

In November 2018, the PISCES[84, 85] webserver was used to obtain a non-redundant set of protein chains as the base for the research. PISCES is a public webserver for culling sets of protein sequences from the Protein Data Bank (PDB) [7] by user-specified cutoff for sequence identity and structural quality. To obtain enough proteins with good structure quality for the research, the specified percentage identity cutoff of chains was 25%, the resolution cutoff was 2.5 Å, and the R-factor cutoff was 0.25. The total number of chains in the used dataset was 11,159, and the total number of amino acids was 2,632,534. Using the Pbxplore tool [27] and PDB data for proteins obtained by PISCES, each chain was coded to a sequence of protein blocks.

Data preparation for PBs model building

Information of a chain that can be predicted or determined based on the amino acid sequence was used in addition to an amino acid sequence as input in the model-building phase in order to obtain more accurate PBs predictors. Method Spider3 was used for the prediction of structural properties of proteins; program RepeatPlus was used to determine the occurrence of certain types of repeats in amino acid sequences and several predictors were used to find potential protein intrinsically disordered regions. Figure 4.1 illustrates the process of PBs prediction models development. The amino acid sequences in FASTA format were presented as input to Spider3, protein intrinsically disordered regions predictors and RepeatPlus. Their results were combined with amino acid sequences and corresponding PBs sequences and used as input to classification algorithms.

Structure properties

Spider3 [44] is a method for the prediction of one-dimensional structure properties from an amino acid sequence using LSTM-bidirectional recurrent neural networks and evolutionary information of a chain. The result of a method contains predicted secondary structure described with 3 states (H for helix, E for sheet and C for

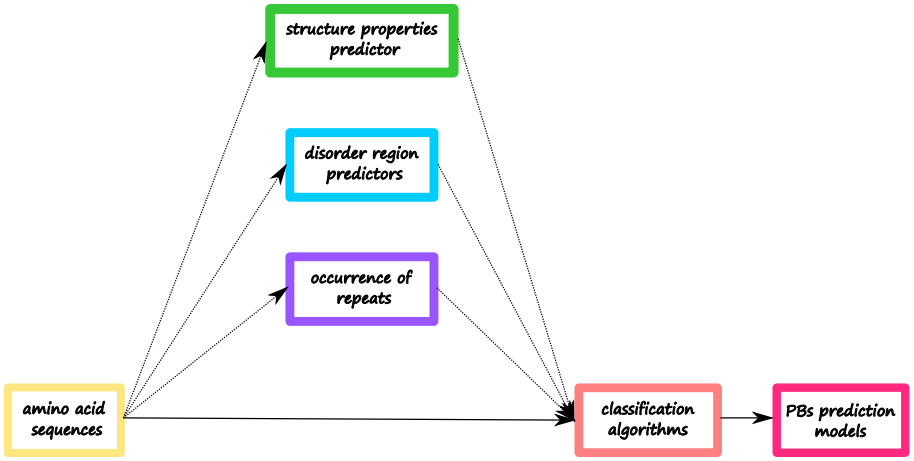

Figure 4.1: Illustration of the process of PBs prediction models development

coil), accessible surface area (ASA) and backbone ϕ and ψ angles for each amino acid in a sequence given as input. Spider3 was used as a web service.

Repeats in amino acid sequences

Program RepeatsPlus [50] based on a method defined by Jelovic et al. [51] was used for finding a statistically significant subset of a determined set of repeats in amino acid sequences. Direct repeats (DN) and inverse repeats (IN) with minimal length 2 in amino acid sequences were extracted. A pair of substrings x and y of a string z where x begins at position $start_x$ of z, and substring y begins at position $start_y$ of z, is

- *direct repeat* (DN) if and only if $x = y$ and $start_x < start_y$;
- *inverse repeat* (IN) if and only if $x = w$ and $start_x \leq start_y$ where w is inverse of y (for example, the inverse of string abc is cba).

Substring x is a *left part of a repeat* and substring y is a *right part of a repeat*. Repeats of both types in protein chains were extracted in the format: $chain\_id, start_x, end_x, start_y, end_y, x, y$

Disorder regions of protein

Whole proteins or parts of proteins that do not fold or lack fixed 3D structure are called disorder proteins and disorder regions [28]. Methods, such as X-ray crystallography and NMR spectroscopy [21], are used for experimental determination

of disorder regions of proteins. Several predictors have been developed to locate potentially disorder regions for a given amino acid sequence. Some of these predictors were used to mark potentially disorder regions in chains from the dataset: VL-XT, DisEMBL, Espritz, GlobPlot, IUPred2A, IsUnstruct and RONN.

PONDR (Predictor Of Natural Disordered Regions) VL-XT predictor [31] is based on neural networks. The output of the PONDR VL-XT is a prediction score between 0 and 1 for each amino acid in a given sequence. A score larger than 0.5 indicates the potential that amino acid belongs to a disorder region.

DisEMBL [32] uses three criteria for assigning order or disorder state to each amino acid in a sequence:

- *loops/coils* criterion relies on DSSP [53] tool for secondary structure assignment. The amino acids to which DSSP assignes loop state (T, S, B, I) are labeled with disorder state.

- *hot loops* criterion relies on DSSP tool for secondary structure prediction and amino acids mobility based on B factor. Disorder state is assigned to the amino acids from loops with a high degree of mobility.

- *missing coordinates* criterion relies on X-Ray structures in the PDB database. Amino acids with missing coordinates are labelled with disorder states.

An ensemble of artificial neural networks was built for each of the defined criteria for the disorder. DisEMBL with different criteria was applied on amino acid sequences in the used dataset. Result obtained using *loops/coils* criterion was labelled with DisEMBL_LC; the result obtained using *hot loops* criterion was labelled with DisEMBL_HL and the result obtained using missing coordinates was labelled with DisEMBL_R465.

Espritz [83] is an ensemble of predictors built using bidirectional recurrent neural networks. As input parameters for the predictors were used: five Atchley indices [5], which transform amino acid codes to five numerical values and summarize amino acid variability, *one-hot* encoding of 20 amino acids and multiple sequence alignment profile. Proteins from different datasets were used for training and testing the predictors: X-ray disorder, DisProt disorder, NMR mobility and other datasets.

Espritz predictor based on different datasets was applied on amino acid sequences in the used dataset for the research. Result obtained using the X-ray

disorder dataset was labelled with Espritz_XRAY; the result obtained using the DisProt disorder dataset was labelled with Espritz_DISPROT and the result obtained using the NMR mobility dataset was labelled with Espritz_NMR.

GlobPlot [60] calculates function for the propensity of amino acids to be in globular or non-globular states. It is based on propensity for a given amino acid to be in *random coil* and *regular secondary structure*.

IUPred2A [67, 20] combines two predictors - IUPred and ANCHOR2. IUPred predicts disorder or order state for each amino acid in the amino acid sequence while ANCHOR2 predicts disordered binding sites. For the prediction, IUPred and ANCHOR2 use an energy estimation based on the interaction between amino acids in the local environment. IUPred2A was used for the prediction of short stretches of disorder (result was labelled with IUPred2A_S) and long disordered regions (result was labelled with IUPred2A_L).

IsUnstruct [61, 62] predictor is based on the Ising model from statistical physics which was adapted to the disorder–order transition in a protein chain.

RONN (regional order neural network) [87] predictor is a modification of bio-basis function neural network method [80] for the detection of disordered regions.

All predictors except VL-XT were downloaded and used in the local environment as standalone versions. VL-XT was used as a web service. EpDis-MassPred system[49] was used for the generation of the output of all disorder predictors except VL-XT for protein chains in the dataset.

The result of VL-XT per chain is in format *position, amino_acid, prediction_score* and the result of EpDis-MassPred contains chain_id, dataset name, the start position of the region, the end position of the region, region state, predictor name. Region state can be D for disorder region or O for order region.

Secondary structures

Dictionary of Secondary Structure of Proteins (DSSP) program [53] was used to assign secondary structure states to amino acids in order to compare the true DSSP and predicted secondary structure from Spider3. For the comparison of the DSSP and predicted secondary structures from Spider3, eight DSSP states were transformed into three states (H, E and C), as it was done for training the Spider3 method:

- states G, H and I were transformed to H;

- states B and E were transformed to E;
- other states were transformed to C.

Data preparation

After processing and merging the outputs of Spider3, RepeatPlus, DSSP and disorder predictors, each amino acid in a sequence was described with:

- categorical values:
  - one-letter name;
  - true secondary structure from DSSP;
  - predicted secondary structure by Spider3;
  - disorder region flag per disorder predictor;
  - flag per left and right part of DN repeat;
  - flag per left and right part of IN repeat;
- numerical values:
  - predicted backbone angles by Spider3;
  - predicted accessible surface area by Spider3.

In the research, the dataset for building PBs prediction models was prepared in both data formats described in section 2.2: fixed-length sliding window format and sequence format. Sliding window of length five was used, i.e. one instance corresponded to a fragment of five consecutive amino acids in a sequence $(A_{x-2}, A_{x-1}, A_x, A_{x+1}, A_{x+2})$ (x is a position of amino acid in a sequence). The amino acid at each position in a fragment was described with a one-letter name and properties obtained with Spider3 (backbone angles, secondary structure and ASA). Repeat and disorder flags at the level of the amino acid were replaced with repeat and disorder flags at the fragment level. Table 4.1 contains a detailed description of repeat flags at the fragment level which were calculated for DN and IN repeats. Disorder flags at the level of a fragment were calculated by the rule: If any amino acid in a fragment belongs to a disordered region, the flag value is 1, otherwise 0. The target attribute contained PB of the fragment's central amino acid. Fixed-length sliding window version of dataset contained only fragments

whose central amino acid had a defined PB, i.e. fragments whose assigned protein block was not Z.

Part of the chain without missing amino acids in the PDB file was one instance in sequence format version of data. Each amino acid in an instance was described with its one-letter name, properties obtained with Spider3, flags of repeats at the amino acid level and IDR flags at the amino acid level. Target attribute contained the associated sequences of PBs. It must be noted that for the first two and last two amino acids in an input sequence, *Z* was assigned instead of PB since a PB cannot be assigned to them.

Dataset partition

For building PBs prediction models, two types of data partition were used:

- partition based on protein sequences. 11,159 sequences from PISCES were split into two groups called *subset1* and *subset2* for building PBs prediction models based on fixed-length sliding window format data and PBs prediction models based on sequence data. *Subset1* had 7,761 chains, and *subset2* had the remaining 3,398 chains. To test the impact of the size of the training dataset on the quality of the PBs prediction model, instances of chains in *subset1* were used initially as training dataset and instances of chains in *subset2* as test dataset. Afterwards, instances of chains in *subset2* were used as training dataset, and instances of chains in *subset1* as test dataset. Besides the partition based on *subset1* and *subset2*, partition on training and test parts with the ratio of 50:50 was used for building PBs prediction models based on sequence data.
- partition based on instances in fixed-length sliding window format data. Dataset was partitioned in training, validation and test parts using different ratios of instances. This partition type can be used only for building PBs prediction models based on fixed-length sliding window format data.

Exploration of dataset

In order to check the quality of the dataset and relationships between attributes, the following analyses of the dataset were performed:

- amino acid distribution in the dataset;

Disorder flag on the instance level	**Description**
PB out repeat	If no amino acid of an instance belongs to either the left or right part of any repeat, the flag value is 1, otherwise is 0.
repeat in PB	If amino acids of an instance contain the left or right part of any repeat (applicable only to repeats of length <= 5), the flag value is 1, otherwise is 0.
PB in repeat	If complete instance is in the left or the right part of any repeat, the flag value is 1, otherwise is 0.
PB center in repeat	If the central amino acid of an instance is in the left or right part of any repeat, the flag value is 1, otherwise is 0.
PB intersect left repeat	If complete instance is in the left part of any repeat, the flag value is 1, otherwise is 0.
PB intersect right repeat	If complete instance is in the right part of any repeat, the flag value is 1, otherwise is 0.
left edge of left repeat in PB	If amino acids of an instance are on the left edge of the left part of any repeat, the flag value is 1, otherwise is 0.
right edge of left repeat in PB	If amino acids of an instance are on the right edge of the left part of any repeat, the flag value is 1, otherwise is 0.
left edge of right repeat in PB	If amino acids of an instance are on the left edge of the right part of any repeat, the flag value is 1, otherwise is 0.
right edge of right repeat in PB	If amino acids of an instance are on the right edge of the right part of any repeat, the flag value is 1, otherwise is 0.
PB intersect homorepeat	If amino acids of an instance have an intersection with (either left or right edge of) repeat, and repeat is homorepeat than the flag value is 1, otherwise is 0.

Table 4.1: Repeat flags at the fragment level for DN and IN repeats

- secondary structure distribution in the dataset;
- Spider3 prediction quality;
- analysis of the occurrence of DN and IN repeats in amino acid sequences;
- analysis of predicted disorder regions.

Also, comparison of instances of chains in *subset1* and *subset2* was performed.

Amino acid distribution in the dataset

Amino acid frequencies in the dataset were compared to calculated amino acid frequencies in the UniProtKB/Swiss-Prot protein knowledgebase release 2021_01 (UniProtKB/Swiss-Protdb)[1]. For the comparison of *subset1* and *subset2*, amino acid frequencies were calculated in chains of each subset. The correlation of amino acid frequencies in the used dataset and UniProtKB/Swiss-Prot database is 0.99, as well as the correlation of amino acid frequencies in chains of *subset1* and *subset2*. Table 4.2 shows the calculated amino acid frequencies. Comparing amino acid frequencies in the used dataset and UniProtKB/Swiss-Prot database, it can be noticed that amino acids Methionine (M) and Serine (S) were slightly more represented in UniProtKB/Swiss-Prot db, while amino acids Aspartic Acid (D) and Tyrosine (Y) were slightly more represented in the dataset. Comparing amino acid frequencies in the chains of *subset1* and *subset2*, it can be noticed that the largest differences correspond to Methionine (M) and Glutamic Acid (E). The differences of frequencies in *subset1* and *subset2* for Methionine and Glutamic Acid are 1.56% and -0.5%, respectively.

In the analysis of frequencies of amino acid fragments of lenght 5 ($A_1A_2A_3$ A_4A_5) in whole dataset per protein block and position of amino acid in a fragment, it was noticed that some amino acids are well or poor represented at some positions of certain PBs:

- Glycine (G) is well represented at position 2 for PB *a* (60.2%), at position 3 for PBs *i* (58.8%) and for PB *j* (82.4%), at position 4 for PBs *p* (41.3%), *h* (58.1%) and *o* (61.0%), and at position 5 for PBs *g* (47.4%), *e* (58.2%) and *n* (64.3%).
- Proline (P) is well represented at position 2 (12.0%), and position 5 (11.8%) in PB *b*, at positions 4 (13.6%) and 3 (10.3%) in PB *f*; at position 4 (12.2%)

Amino acid	**dataset (%)**	***subset1*(%)**	***subset2*(%)**	**UniProtKB Swiss-Prot db(%)**
A	8.15	8.04	8.41	8.25
C	1.23	1.29	1.08	1.38
D	5.96	5.9	6.11	5.46
E	6.77	6.63	7.13	6.72
F	4.2	4.19	4.22	3.86
G	6.96	7	6.85	7.07
H	2.36	2.38	2.29	2.27
I	5.84	5.75	6.08	5.91
K	5.75	5.72	5.83	5.8
L	9.61	9.52	9.84	9.65
M	1.69	2.13	0.57	2.41
N	4.36	4.37	4.32	4.06
P	4.53	4.56	4.46	4.73
Q	3.83	3.78	3.95	3.93
R	5.17	5.17	5.18	5.53
S	6.1	6.12	6.06	6.63
T	5.44	5.41	5.5	5.35
V	6.94	6.91	7.01	6.86
W	1.44	1.47	1.38	1.1
Y	3.66	3.64	3.71	2.92

Table 4.2: Amino acid frequencies in whole dataset, part *subset1*, part *subset2*, and in UniProtKB/Swiss-Prot database

in PB *g*; at position 2 (12.9%) in PB *i*; at position 3 (17.7%) in PB *k*; at position 2 (17.2%) in PB *l*. Proline is poorly represented in PB *n* at all positions (frequency is from 0% to 1.9% per position), and its frequency is 0% in PB *b* at position 5.

- Serine (S) is well represented in PB *k* at position 2 (15.2%), PB *l* at position 1 (14.4%) and PB *f* at position 3 (13.3%).

In fragments of PB *m*, each amino acid has a similar frequency at all positions in a fragment. For example, Leucine (L) has frequencies in the range from 10.7% to 11.2%.

Secondary structure distribution in the dataset

Analysis of the secondary structure distribution in fragments of length 5 per PB was performed. Transformed 3-state DSSP secondary structures of amino acids were used in the analysis. The results are in accordance with the results of analysis of PBs and secondary structures published in [15, 36]. Conclusions per PB are:

- PBs *a, b* and *c* are associated with β-sheet N-caps since
  - percentages for β-sheet in fragments of PB *a* from position 1 to 5 are: 8.3%, 4.3%, 21.6%, 73.7%, and 48.1%;
  - percentages for β-sheet in fragments of PB *b* from position 1 to 5 are: 33.5%, 28.5%, 14.6%, 35.3%, and 38.8%;
  - percentages for β-sheet in fragments of PB *c* from position 1 to 5 are: 7.4%, 25.4%, 45.4%, 53%, and 51.5%.
- PBs *e* and *f* are associated with β-sheet C-caps since
  - percentages for β-sheet in fragments of PB *e* from position 1 to 5 are: 66.6%, 66.7%, 56%, 33.6%, and 8.2%;
  - percentages for β-sheet in fragments of PB *f* from position 1 to 5 are: 41.4%, 40.3%, 29.2%, 8.5%, and 13.6%;
- PBs from *g* to *j* are mainly associated with coils.
- PBs *k* and *l* are associated with α-helix N-caps since
  - percentages for α-helix in fragments of PB *k* from position 1 to 5 are: 3.8%, 0.2%, 50.0%, 55.0,%and 57.8%, while for coil are: 67.5%, 83.6%, 49.4%, 44.4%, and 37.1%;
  - percentages for α-helix in fragments of PB *l* from position 1 to 5 are: 1.6%, 58.3%, 64.7%, 71.3%, and 64.4%, while for coil are: 84%, 41.2%, 34.6%, 27.2%, and 30.1%.
- PBs *n*, *o* and *p* are associated with α-helix C-caps since
  - percentages for PB *n* from position 1 to 5 are: 83.3%, 82.8%, 75.9%, 38.1%, and 2.3%, while for coil are: 13.4%, 16.8%, 23.7%, 61.5%, and 96.7%;

  - percentages for α-helix in fragments of PB o from position 1 to 5 are: 65.7%, 59.2%, 28.9%, 1.2%, 6.2%, while for coil are: 31.3%, 40.5%, 70.5%, 97.4%, 82.9%;
  - percentages for α-helix in fragments of PB p from position 1 to 5 are: 66.1%, 44.1%, 21.1%, 0.8%, 7.5%, while for coil are: 33.3%, 55.4%, 77.8%, 85.5%, and 63.2%.

- The two most frequent PBs (m and d) are associated with regular secondary structures:
  - PB m is associated with the central α-helix. Percentages for β-sheet in fragments of PB d from position 1 to 5 are: 91.9%, 92.5%, 91.5%, 89.2% and 82.0%.
  - PB d is associated with β-sheet. Percentages for β-sheet in fragments of PB d from position 1 to 5 are: 64.2%, 70.6%, 74.7%, 68.9% and 56.2%.

There is no significant difference in the secondary structure distribution in chains in *subset1* and *subset2* since the correlation of secondary structure frequencies is 0.99.

Spider3 prediction quality

Analysis of the quality of the Spider3 predictor on the dataset for secondary structures and dihedral angles was performed. As a performance measure for secondary structure, accuracy was used. The 3-state secondary structures assigned by DSSP to amino acids in the dataset presented the true secondary structures. Measured accuracy on the dataset is 85.43% , which is in accordance with the reported accuracy of 84% for Spider3 secondary structure prediction in [44]. As expected, H state is best predicted by Spider3 with a recall of 0.9, E state is worst predicted with a recall of 0.79. The recall of secondary structure per state varies among PBs:

- the recall of state H is between 0.77 and 0.95 for PBs *d, f, k, l, m* and *n*, while for other PBs is up to 0.51, and for PBs i is only 0.03;
- the recall of state E is between 0.71 and 0.87 only for PBs *b, c, d* and *e*, while recall is up to 0.62 for other PBs, and for PB o is only 0.04;

- the recall for state C is between 0.82 and 0.96 for PBs from a to l and PBs o and p, while for PBs m and n is 0.66 and 0.71, respectively.

Mean Absolute Error (mae) was used for the comparison of true angles and predicted angles by Spider3. mae was calculated by the equation:

$$mae = \frac{\sum_{i=1}^{n} |true\_angle_i - predicted\_angle_i|}{n} \tag{4.1}$$

where n is the number of amino acids.

The calculated mae on the used dataset are 17.4° for ϕ and 25.0° for ψ, which is better then reported mae for angles by Heffernan et al. in [44] where mae is 18.3° for ϕ and 27.0° for ψ. Calculated mae for angles of a central amino acid in fragments per PB are in the range from 8.4° to 59.6°. PBs associated with α-helix have the smallest mae: 8.4° for ϕ and 12.3° for ψ for PB m, 14.8° for ϕ and 21.1° for ψ for PB n. PBs associated with coils have largest mae: 55.8° for ϕ and 59.6° for ψ for PB j and 55.5° for ϕ and 37.7° for ψ for PB i.

Figure 4.2 shows calculated mae for the central amino acid in fragments of length 5 of each PB. Values are in degrees.

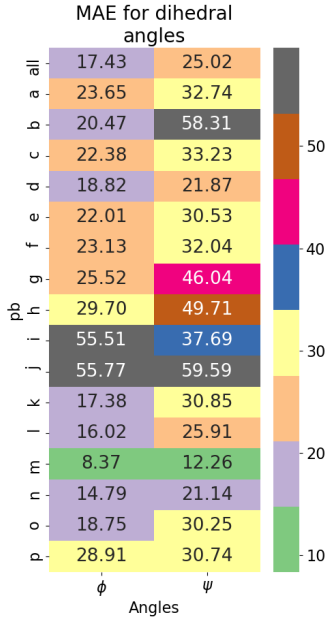

Figure 4.2: Mean absolute error between true dihedral angles in proteins and predicted angles by Spider3 per PB

Analysis of occurrence of DN and IN repeats in amino acid sequences

DN repeats are slightly more present in amino acid sequences in the dataset: on average, 33 amino acids per 100 amino acids in the sequence belong to DN repeats, while 35 amino acids per 100 amino acids in the sequence belong to IN repeats. In the analysis of the occurrence of repeats per position of amino acids in fragments of length 5 for each PB, it was observed that:

- the central amino acid in PB j has the highest frequencies of DN and IN repeats. 44.9% of PB j fragments have a central amino acid belonging to DN repeats, and 48.7% to IN repeats.
- PBs m, n and o have the highest frequencies of DN and IN repeats at each position. Repeat frequency per amino acid position in these PBs is between 39.2% and 42% for DN repeats and between 41.1% and 44.8% for IN repeats.
- PBs b and d have lower repeat frequencies at each position; frequencies of DN repeats per amino acid position are between 35.8% to 36.5% and between 38.7% and 39.8% for IN repeats.

Analysis of predicted disorder regions

The occurrence of disorder fragments was calculated for the whole dataset and fragments per PB. Disorder fragments are fragments of length 5 with at least one amino acid in a disorder region. The occurrence of disorder fragments was calculated for each applied predictor and it is from 5.35% to 91.76% for the dataset (see Table 4.3). For easier comparison of disorder predictors per PBs, the difference between the maximal disorder fragment frequency in PBs and the minimal disorder fragment frequency in PBs (disorderdiff) was calculated by the equation:

$$disorder_{diff} = max_{x \in PBs} f_x - min_{x \in PBs} f_x \tag{4.2}$$

where f_x is a frequency of disorder fragments for a PB x.

Table 4.3 shows the calculated percentage of predicted disorder fragments and the difference between the max and min frequency of disorder fragments among PBs calculated as $disorder_{diff} * 100$ for each disorder predictor. It can be noticed that:

- protein blocks are similar according to predictors VL-XT and IUPred2A_L. Percentage of disorder fragments is 26.50% for VL-XT and $disorder_{diff}$ is 0.029, while percentage of disorder fragments is 91.76% for IUPred2A_L and $disorder_{diff}$ is 0.039;

- PB m has the smallest disorder fragment frequency or is among PBs with the smaller disorder fragment frequency for each disorder predictor.

- predictor GlobPlot has high $disorder_{diff}$ because disorder fragment frequency of PB m is 8.1%, while disorder fragment frequencies of PBs from g to j and PB e are between 29.3% and 34.8%.

- the highest calculated value for $disorder_{diff}$ is for disorder predictors DisEMBL_HL and DisEMBL_LC due to the lower disorder fragment frequency of PB m than the disorder fragment frequency of other PBs. For disorder predictor DisEMBL_HL, disorder fragment occurrence for PB m is 25.6%, while for other PBs is between 36.6% and 45.5%. For predictor DisEMBL_LC, disorder fragment occurrence for PB m is 43.5%, while for other PBs is between 74.2% and 90.3%.

Disorder predictor	**Occurrence of fragments is disorder region (%) [16]**	$disorder_{diff} * 100$
Espritz_DISPROT	5.35	2.57
Espritz_XRAY	5.37	2.14
DisEMBL_R465	7	2.45
IUPred2A_S	7.25	4.26
Espritz_NMR	14.04	16.28
IsUnstruct	15.07	4.75
GlobPlot	18.4	26.8
RONN	19.17	6.01
VL-XT	26.5	2.94
DisEMBL_HL	37.02	19.94
DisEMBL_LC	70.02	46.83
IUPred2A_L	91.76	3.85

Table 4.3: Percentage of predicted disorder fragments and calculated $disorder_{diff} * 100$ for each disorder predictor

4.2 PBs prediction models

Program Spider3 predicts protein structure properties, among which are dihedral angles. Predicted dihedral angles can be used stand-alone for the prediction of Protein Blocks for a given amino acid sequence. Q_{16} for Protein Blocks predicted for amino acid sequences using only the predicted dihedral angles by Spider3 is 72%. The aim of the research was to use data mining and predicted disordered regions and determined DN and IN repeats, besides predicted protein structure properties by Spider3, in order to obtain a PBs prediction model with higher Q_{16}. With this choice of input data in a prediction model, where predicted dihedral angles are used as part of the input data for the prediction of prototypes of the structural alphabet defined by angles, two questions can arise:

a) If the dihedral angles determined by predicted prototypes of the structural alphabet by the new model are less precise than the predicted dihedral angles used as part of the model input, why this approach should be used?

b) If the dihedral angles determined by predicted prototypes of the structural alphabet are better than the predicted dihedral angles used as part of the new model input, whether a recursive approach should be used? Should dihedral angles determined by predicted prototypes in iteration *i-1* (i = 1,2, ... n) be used in iteration i as part of the input to the model and how many iterations should be used?

Even if predictors of structural alphabet prototypes do not outperform the existing dihedral angle predictors, as is the case with Protein Blocks, the availability of accurate structural alphabet prototype predictors is important since the structural alphabets have been confirmed as very useful in various studies, such as protein structure analysis and comparison of proteins using structural alphabet prototypes [65, 69].

Applied classification algorithms

Based on the format of the dataset, different classification algorithms can be used for building the PBs classification model. As described, the dataset was prepared in two formats which are usually used in the development of predictors for protein structure properties.

For data in fixed-length sliding window format, pattern classification algorithms can be applied. Since an instance of a dataset in fixed-length sliding window format describes the part of a sequence (usually a small part of the whole sequence), only the relations between amino acids in a fragment can be learned by a model. The relationship of amino acids that are in the neighbourhood according to their positions in the sequence is called a short-range relationship. Bidirectional recurrent neural networks (BRNN) can be applied to sequence format data. The advantage of BRNN is the ability to learn short-range, as well as long-range relationships between amino acids in a sequence [42, 43]. BRNN have been successfully applied for the prediction of protein structural properties [44, 29]. Datasets used for the development of existing PBs predictors based on machine learning algorithms [73, 88] were in fixed-length sliding window format. The list of applied classification algorithms in the research and used data mining packages is shown in Table 4.4.

Data format	**Package**	**Algorithm**
Fixed-length sliding window	IBM Intelligent miner	SPRINT
	SPSS Modeler	C5.0, CART, CHAID, QUEST, Random Trees, Neural networks, XGBoostTree
	Python library scikit-learn	Multilayer Perceptron (MLP), Random Forest (RF)
Sequence	Python library Keras	LSTM-BRNN

Table 4.4: List of used data mining packages and classification algorithms

Classification based on fixed-length sliding window data

In order to develop an accurate PBs prediction model, different models were built using data in fixed-length sliding window format and several classification algorithms from data mining packages SPSS Modeler [48], IBM Intelligent miner [46], and Python library scikit-learn [24, 23]. Obtained PBs prediction models were built using different decision tree algorithms and neural networks in SPSS Modeler, as well as Decision Forest and Multilayer Perceptron implemented in the

Python library scikit-learn (see Table 4.4). Also, the algorithm SPRINT implemented in IBM Intelligent miner was used for building the PBs prediction model.

Additional preprocessing for the dataset in fixed-length sliding window data format wasn't necessary before the application of classification algorithms implemented in SPSS Modeler and IBM Intelligent miner, since they use preprocessing procedure to adapt data in the required format as part of the model-building phase. Since the algorithms in the Python library scikit-learn require the numeric input data, the categorical attributes (amino acid name and secondary structure) were transformed using *0 to n-1* coding. For building models based on neural networks implemented in Python packages, values of dihedral angles were transformed to range $[0, 1]$ by formula $\frac{angle+180}{360}$. Values of ASA attributes of amino acids were normalized to range $[0, 1]$ by dividing the ASA value with the maximal value of the corresponding amino acid. Repeat flags had 1/0 values. Disorder flags had D/O values or 1/0 when the numerical inputs were required.

PBs prediction models obtained using SPSS Modeler and Intelligent Miner

PBs prediction models were developed with SPSS Modeler using (a) data partition based on instances and ratio of 50:30:20 for training, test and validation parts and (b) data partition based on protein sequences (*subset1/subset2* partition). In order to find optimal parameter values of algorithms (such as boosting, bagging, cost matrix for Decision trees), different values have been tested.

Table 4.5 shows the calculated Q_{16} for the best PBs prediction models obtained using data partition based on fragments. The best-obtained model by Q_{16} was developed using C5.0 in SPSS Modeler with boosting option with value 10 and defined cost matrix with weight 10 for misclassification of PB *d*. The best model was called *PBC5.0d*. The Q_{16} of *PBC5.0d* for training and test parts is 80.8%, and for validation part is 80.7%.

Besides listed algorithms, the SVM algorithm was also applied, but SPSS Modeler failed to generate a model after a few weeks of execution.

The analysis showed that ASA attributes do not contribute to increasing the accuracy of models, so ASA attributes were not used in building the final PBs prediction models with SPSS Modeler and Intelligent Miner.

For building PBs prediction models based on sequence data partition (*subset1/subset2* partition), only C5.0 algorithm was used since it had the best Q_{16}

Algorithm	Q_{16} **for training part (%)**	Q_{16} **for test part (%)**	Q_{16} **for validation part (%)**
C5.0 with weight 10 for misclassification of PB d in cost matrix	80.83	80.78	80.72
Random Trees without cost matrix	67.36	67.32	67.3
CART without cost matrix	69.48	69.49	69.44
Exhaustive CHAID without cost matrix	63.29	63.15	63.25
QUEST without cost matrix	64.48	64.38	64.43
MLP	72.55	72.47	72.5
Bayes Net	70.01	69.94	69.89
XGBoost Tree	78.16	78.1	78.04

Table 4.5: Q_{16} of SPSS models obtained using SPSS Modeler on data with fixed-length sliding window format and dataset partition based on fragments

among models based on fragment partition.

First, the *subset1* was used for building PBs prediction models and *subset2* for the final test. Then, the opposite, *subset2* was used for building PBs prediction models and *subset1* for the final test. Models were built using a ratio of 50:50 for training and test parts in both cases. Best-obtained models were built using algorithm options: boosting 4 and cross-validation 10. PBs prediction models were built without and with a defined cost matrix with weight 10 for misclassification set in all cells referring to preferable PB. Table 4.6 shows the results of PBs prediction models built on *subset1*, while Table 4.7 shows the results of PBs prediction models built on *subset2*. Tables 4.6 and 4.7 show Q_{16} for training and test parts obtained by partitioning the subset used for building the model, and Q_{16} for final tests. For models built on *subset1*, Q_{16} of training and test parts is between 76.59% and 81.25%, but Q_{16} on the final test is worse, it is between 72.11% and 75.50%. For models built on *subset2*, Q_{16} for training and test parts is between 77.13% and 81.57%, and Q_{16} on final test is between 70.83% and 75.25%. Same as for models built on *subset1* , models built on *subset2* perform a bit worse on the final test. It can be noticed that models built with the same parameter values but on different subsets behave similarly according to Q_{16} for the training and test parts and on the final evaluation. The highest difference for Q_{16} is 3.19% (on final tests for a

pair of models built with a defined cost matrix for PB *n*). Q_{16} differences for the rest pairs of models and data subsets are less than 1.51%.

PB for which was increased a weight for misclassification in cost matrix	Q_{16} **for training partition (%)**	Q_{16} **for test partition (%)**	Q_{16} **for *subset2* (%)**
a	78.64	78.62	73.14
b	79.62	79.64	73.14
c	80.74	80.59	72.43
d	81.17	81.25	72.11
e	78.09	78.16	72.55
f	80.03	80.02	72.68
g	77.81	77.87	73.44
h	78.09	78.19	73.22
i	77.73	77.77	73.47
j	77.42	77.5	73.57
k	79.31	79.31	72.88
l	79.36	79.43	72.92
m	80.47	80.45	72.41
n	77.6	77.73	75.5
o	77.98	78.06	73.32
p	78.67	78.63	73.1
without cost matrix	76.59	78.55	73.84

Table 4.6: Q_{16} for PBs prediction models built on *subset1* using algorithm C5.0 with options boost 4 and cross-validation 10 and without or with a cost matrix for preferable PB

PBs prediction model based on SPRINT algorithm was built using IBM Intelligent miner and ration of 50:50 for training and test parts. The obtained Q_{16} for training data is 71.83%, and 71.77% for test data.

PBs prediction models obtained using library scikit-learn

As listed in Table 4.4, algorithms Multilayer Perceptron (MLP) and Random Forest (RF) from library scikit-learn were used for building the PBs prediction models. For finding the optimal values of algorithm parameters, 5-fold cross-validation was applied on a sample of 3,720 sequences with a resolution up to 1.7 Å. The ratio of 70:30 for the training-test partition was used. Optimal architecture for Multilayer Perceptron (MLP) was: 3 hidden layers with 100 neurons per layer, logistic sig-

PB for which was increased a weight for misclassification in cost matrix	Q_{16} **for training partition (%)**	Q_{16} **for test partition (%)**	Q_{16} **for *subset1* (%)**
a	79.00	79.04	71.91
b	79.96	79.92	71.63
c	81.25	81.02	71.15
d	81.56	81.57	70.83
e	78.67	78.66	71.96
f	80.42	80.54	71.34
g	78.44	78.53	72.23
h	78.56	78.64	72.01
i	78.16	78.26	72.24
j	78.24	78.24	72.28
k	79.87	79.81	71.54
l	79.69	79.76	71.6
m	80.93	80.94	71.17
n	78.2	78.28	72.31
o	78.48	78.56	72.22
p	79.17	79.21	71.94
without cost matrix	77.13	77.22	75.25

Table 4.7: Q_{16} for PBs prediction models built on *subset2* using algorithm C5.0 with options boost 4 and cross-validation 10 and without or with a cost matrix for preferable PB

moid function as activation function, Adam solver for weight optimization, initial learning rate 0.0001 and the maximum number of iterations 500. The optimal parameter values for Random Forest (RF) classifier were: the number of trees was 100, the impurity measure was Gini function, the maximum depth of the tree was 50, the minimum number of samples required to split an internal node was 50.

Based on optimal parameters, the final models were built using fragment-based training-test partition with a ratio of 50:50. Table 4.8 shows the Q_{16} for obtained models.

Besides the dataset with fragments of length 5, the longer fragments (with lengths from 7 to 13 amino acids) were also used for building the PBs prediction models with algorithms from the scikit-learn library. PBs prediction models based on fragments longer than 5 did not provide better Q_{16}, which leads to the conclusion that the properties of 5 amino acids that determine one protein block are sufficient for the prediction of protein blocks.

Algorithm	Q_{16} **(%) for training part**	Q_{16}**(%) for test part**
RF	77.44	74.37
MLP	74.30	74.19

Table 4.8: Q_{16} for PBs prediction models built using algorithms Random Forest and Multilayer Perceptron

Algorithms SVM and KNN from library scikit-learn were also applied using 5-fold cross-validation and dataset for parameters determination. The obtained Q_{16} model for SVM was less accurate (Q_{16} on a test part was 73.6%) and the model building lasted several days. Hence, the SVM was not used to build a model using the larger data. For all tested values for parameters of the KNN algorithm, only overfitted models were obtained since the Q_{16} for the training set was always 100%, and up to 65% for the test set. Thus, neither the KNN was used to build the model using the whole data.

Classification based on sequence data

PBs prediction models based on data in sequence format were built using Long Short Term Memory Bidirectional Recurrent Neural Networks (LSTM-BRNN) and python libraries Keras [37] and TensorFlow [2]. For finding the optimal architecture of the neural network, 3,720 sequences with resolution up to 1.7 Å were used. The chosen architecture used: LSMT layer with 64 nodes per direction, Adam optimization algorithm, concatenation as merge mode, tanh as activation function, accuracy as evaluating metric and categorical cross-entropy as a loss function.

For building the final model, *subset1/subset2* partition was used. *subset1* was used as training part and *subset2* as part for testing. 20% of instances in the training partition was used as validation data. The model was built using 40 epochs. Q_{16} for the obtained LSTM-BRNN model is 75.0% on the training partition and 75.3% on the test partition.

Networks with other activation functions and merge mode did not increase Q_{16}, nor networks with more complex architecture.

Besides *subset1/subset2* partition, PBs prediction models were built using partition based on sequences with a ratio of 50:50 for training-test parts. The obtained

LSTM-BRNN model is similar to the model built using *subset1/subset2* partition, since the Q_{16} for the training part was 75.4% and 74.7% for the test part.

Since Spidar3 also uses sequence format of data, this approach provides learning of the long-range relationship between amino acids.

4.3 Analysis of PBs prediction models

The obtained PBs prediction models were analyzed by:

- comparing the obtained PBs prediction models from different groups;
- the influence of disorder flags and repeat flags on quality of PBs prediction models;
- detailed overview of the best PBs prediction model;
- comparison with other works.

Comparisons of the obtained PBs prediction models

The best developed PBs prediction models from each group of models were used in comparison:

- PBs prediction model *PBC5.0d* developed using SPSS Modeler, as the best-developed model;
- MLP and RF models obtained using scikit-learn library;
- BRNN-LSTM model obtained using library Keras and data in sequence format.

Performance of the selected PBs prediction models was measured using Q_{14} (see Table 4.9), precision and recall of PBs calculated on test dataset part (see Figure 4.3a and 4.3b). Q_{14} was used to measure the PBs prediction models performance for PBs other than the two most represented PBs (PB *m* and PB *d*). Since the PBs *m* and *d* are two PBs that all models predict well, Q_{14} is useful for comparing the models based on their performance for less frequent and less predictable PBs.

PBs prediction model	Q_{14}(%)
PBC5.0d	68.7
MLP (scikit-learn)	58.7
RF(scikit-learn)	58.6
BRNN-LSTM	58.5

Table 4.9: Q_{14} of PBs prediction models

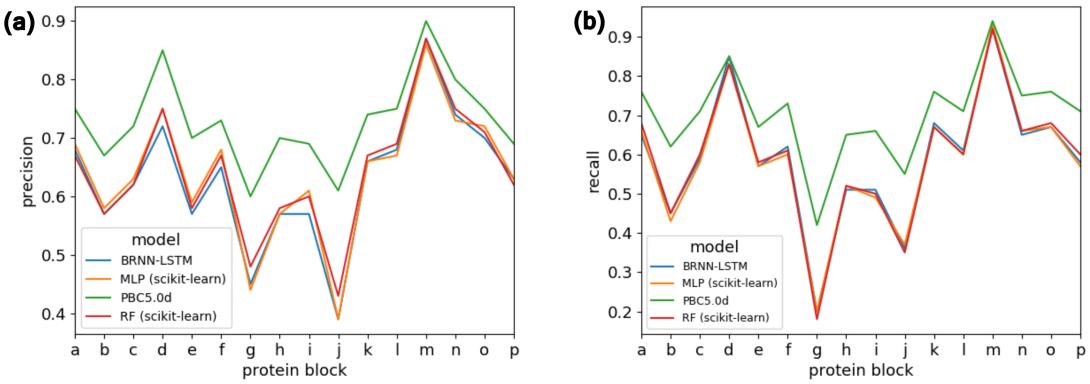

Figure 4.3: Precision (a) and recall (b) calculated on the test dataset for PBs prediction models

Model *PBC5.0d* has the best performance by Q_{14} (68.7%) and the highest precision and recall for all PBs. All PBs models have high and similar precision and recall for PB *m*; precision for PB *m* is between 0.87 and 0.9, and the recall is between 0.92 and 0.94. Also, all models have a similar recall for PB *d*. For other PBs, the model *PBC5.0d* performs better than other PBs models; especially for PBs associated with the secondary structure coil.

Besides the PBs prediction models from different model groups, models based on *subset1/subset2* partition and built using algorithm C5.0 and defined cost matrix for particular PB were compared. *subset1* was futher partitioned using a ratio of 50:50 on the training and test part. Calculated precisions on the test part for obtained models are shown in Figure 4.4a, while the calculated recalls are shown in Figure 4.4b. On the x-axis is PB for which the weight of misclassification in the cost matrix was set to 10, and on the y-axis is PB for which the performance measure (precision and recall) is calculated.

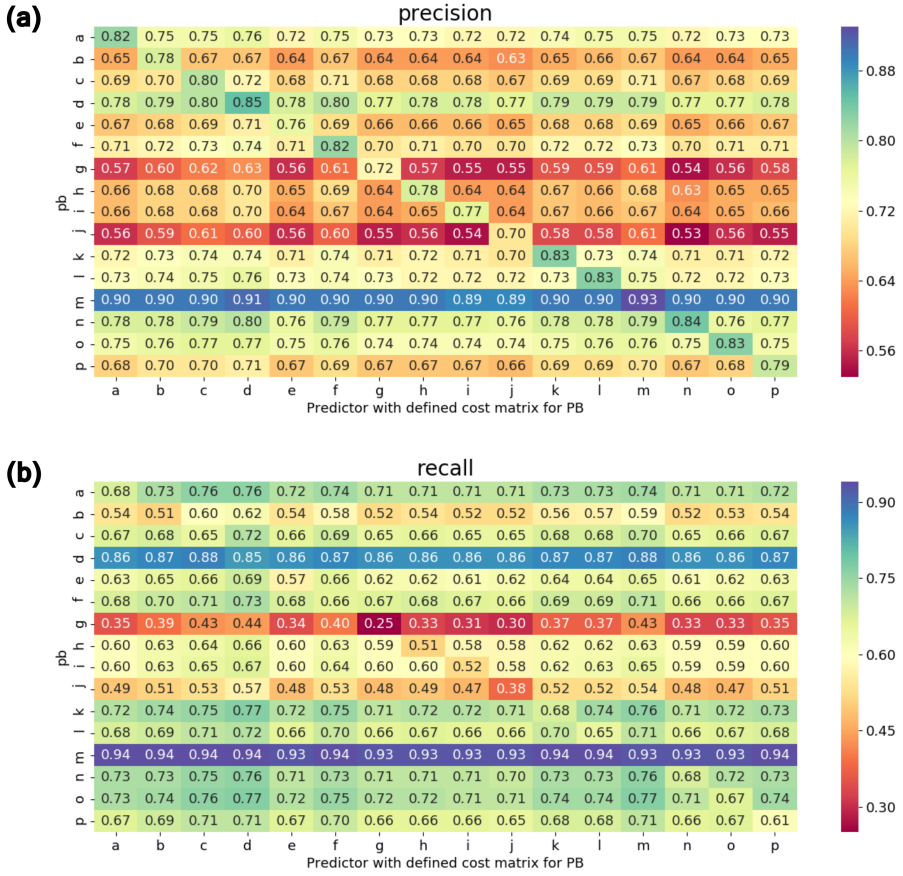

Figure 4.4: Precision (a) and recall (b) of C5.0 models with a defined cost matrix for a particular PB on test part

It can be noticed that the highest precision for each PB has a PBs model obtained using a cost matrix with increased cost for that PB, while it is the opposite for a recall. The positive side of a model built using an increased cost for one PB is that it makes fewer mistakes for instances to which it assigns that PB, and the negative side is that the model will decrease the number of truly predicted instances of that PB. The model with the increased cost for PB *d* is the best among the compared models since it has one of the highest values for recall and precision for most of PBs.

Analysis of the importance of disorder flags and repeat flags on PBs prediction

In order to analyse the importance of disorder flags and repeat flags on PBs prediction, different algorithms were used for building pairs of PBs prediction models using the same parameter values but different data. One PBs model of a pair was built using flags of disordered regions and repeats, and the other without flags of disordered regions and repeats. The difference of Q_{16} for models in a pair is less than 0.01 for all applied algorithms, so the flags of disordered regions and repeats didn't improve the overall accuracy. On the other side, some pairs of models have different precision for some PBs. For easier comparison, the difference of PB precision for a pair of models was calculated by the formula

$$\begin{aligned} precision_{diff} = (&\text{precision of model with flags} - \\ &\text{precision of model without flags}) * 100 \end{aligned} \tag{4.3}$$

Table 4.10 shows the comparison of pairs of models obtained using SPSS Modeler and algorithms: (a) C5.0 with the defined matrix cost for PB *d*, (b) CART and (c) XGBoost-Tree. For each pair of models is shown: precisions calculated using test part and a model built on the dataset with flags, precisions calculated using test part and model built on the dataset without flags, and the difference of precisions calculated as $precision_{diff}$.

According to $precision_{diff}$ calculated for pairs of models built using C5.0, the flags of disordered regions and repeats increase the most the precision for PBs associated with coils (PBs *j*, *i*, and *h*). The usage of flags in model based on CART increases the precision for PBs *n, p, b, a, d* and *j*, while not using the flags increases precision for PBs *f, h* and *l*. Disorder and repeat flags are not very important for models built using XGBoost-Tree algorithm since the absolute value of $precision_{diff}$ for all PBs is less than 1.2 except for PB *i*.

It can be concluded that more accurate predictions for some individual PB can be obtained using models that do not have the highest Q_{16}. For example, a model obtained using XGBoost-Tree has the best precision for PBs *g, i* and *o*, while a model obtained using C5.0 has the best precision for some of other PBs (for example PBs *h* and *j*).

Protein Block	C5.0 with the increased cost for PB d			CART			XGBoost-Tree		
	precision of model with flags	precision of model without flags	$precision_{diff}$	precision of model with flags	precision of model without flags	$precision_{diff}$	precision of model with flags	precision of model without flags	$precision_{diff}$
a	0.75	0.74	0.77	0.63	0.60	2.99	0.75	0.74	0.48
b	0.67	0.65	1.51	0.42	0.39	3.53	0.65	0.64	0.39
c	0.72	0.71	0.78	0.54	0.54	0	0.66	0.66	0.42
d	0.85	0.84	1.26	0.73	0.71	2.5	0.78	0.77	0.21
e	0.70	0.70	0.93	0.62	0.60	1.39	0.70	0.69	1.14
f	0.73	0.72	0.60	0.50	0.56	-6.15	0.71	0.71	0.52
g	0.60	0.58	2.86	0.40	0.40	0	0.65	0.65	0.23
h	0.70	0.68	2.18	0.52	0.55	-3.05	0.68	0.68	-0.11
i	0.69	0.67	1.46	0.46	0.44	1.42	0.72	0.70	1.96
j	0.61	0.57	3.90	0.46	0.44	2.08	0.60	0.60	-0.30
k	0.74	0.73	1.65	0.60	0.61	-1.00	0.72	0.72	-0.09
l	0.75	0.74	1.16	0.59	0.61	-2.97	0.74	0.74	0.50
m	0.90	0.90	0.06	0.86	0.87	-0.58	0.89	0.89	0.16
n	0.80	0.78	1.91	0.78	0.71	7.52	0.80	0.81	-0.46
o	0.75	0.75	0.12	0.64	0.66	-1.43	0.78	0.77	1.19
p	0.69	0.68	0.80	0.57	0.50	6.40	0.69	0.68	0.89

Table 4.10: Comparison of precision per PB for pairs of PBs prediction models built using same classification algorithm and data with and without disorder and repeat flags

Analysis of the best PBs prediction model

As the best obtained PBs prediction model, the performance of *PBC5.0d* was analysed using precision, recall, F_1 and F_2 per PB calculated on test part (see Figure 4.5a.) Also, the normalized confusion matrix of the test part for *PBC5.0d* was calculated (see Figure 4.5b).

Model performs well for the most represented PBs which are associated with regular secondary structures. F_1 for PB m is 0.92 and for PB d is 0.85. However, it has poorer performance for less represented classes that are associated with the coil. F_1 for PB g is 0.50 and 0.58 for PB j.

In the case of false classification, the instances are assigned to PBs m and d for the most PBs. It can be noticed, that the percentage of false assigned instances of PBs i and j to PB p is the highest among false assigned for those two classes.

Comparison with other PBs predictors

The best-obtained model in the research, model *PBC5.0d*, was compared with previously published methods for the prediction of PBs. PBs predictors were compared by the recall reported in the published paper [26] and the recall of PBC5.0d calculated on the test part of the dataset (see Table 4.11). PBC5.0d performance is the best for all PBs except for PB g. Method PB-kPRED using a hybrid method and homologous at 100% for the prediction of a query test has a better recall for PB g.

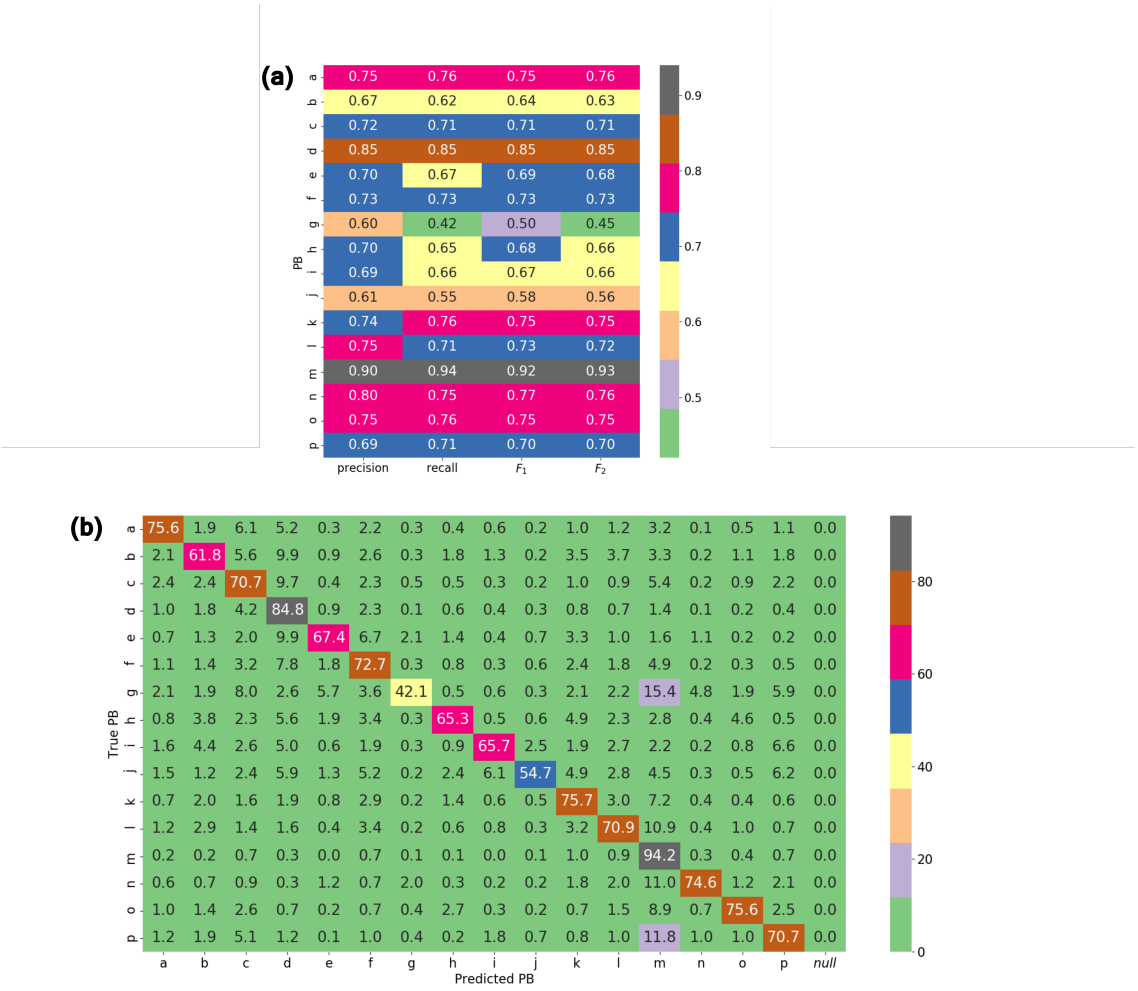

Figure 4.5: Performance measures (a) and confusion matrix (b) calculated on test dataset for *PBC5.0d* model

PB	**Bayes method**	**LOCUSTRA**	**PB-kPRED using a hybrid method and homologous at 30%**	**PB-kPRED using a hybrid method and homologous at 100%**	**PBC5.0d**
a	0.57	0.58	0.46	0.67	0.76
b	0.21	0.26	0.19	0.52	0.62
c	0.33	0.45	0.29	0.59	0.71
d	0.54	0.72	0.40	0.67	0.85
e	0.39	0.45	0.29	0.57	0.67
f	0.31	0.41	0.29	0.60	0.73
g	0.30	0.27	0.15	0.43	0.42
h	0.41	0.38	0.33	0.61	0.65
i	0.38	0.37	0.30	0.59	0.66
j	0.50	0.48	0.21	0.50	0.55
k	0.33	0.46	0.36	0.64	0.76
l	0.36	0.43	0.32	0.60	0.71
m	0.71	0.84	0.56	0.76	0.94
n	0.50	0.52	0.35	0.62	0.75
o	0.48	0.55	0.38	0.63	0.76
p	0.29	0.41	0.32	0.59	0.71

Table 4.11: Comparison of the performance by the recall of PBs predictors of other authors[26] and the best-obtained model *PBC5.0d* in the research

5 Development of new structural alphabets

Existing structural alphabets described in chapter 3 have been developed using the datasets with up to a few hundreds of proteins. Up to now, the largest dataset used in structural alphabet research had <1500 proteins [81]. As Kolodny stated in [58] in 2002, it is to be expected that a more precise structural alphabet can be obtained using a larger database with proteins whose structure has been experimentally determined. This chapter analyzes the capability of developing a new structural alphabet using a clustering algorithm TwoStep, in order to create a base for the development of new predictors of local protein structures.

5.1 Process of the development of structural alphabets

For the analysis, different sets of prototypes (structural alphabets (SAs)) were constructed using clustering on groups of fixed-length fragments with a length of L amino acids ($L = 4, 5, ..., 10$) and a different number of prototypes. Dihedral angles (ϕ and ψ) were used for the description of fragments. A prediction model using a classification algorithm was developed for each SA. SAs were analysed and compared based on their ability to approximate the structure of protein and the accuracy of the corresponding predictor. The aim was to find the optimal number of prototypes in SA and optimal fragment length for approximation of the protein structure and prediction of prototypes for a given amino acid sequence. General steps of the development for one SA are illustrated in Figure 5.1.

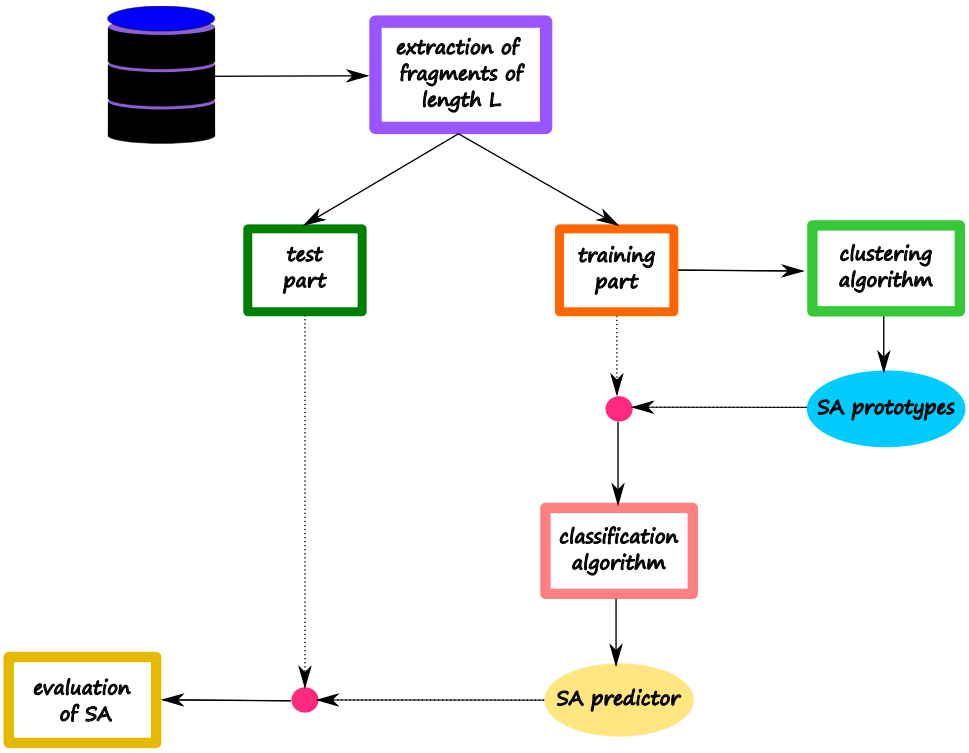

Figure 5.1: Steps in the research about structural alphabets

5.2 Material

The data preparation phase for the research was similar to the data preparation phase for building PBs predictors. The PISCES [84, 85] webserver was used to obtain a non-redundant set of protein chains as the base for the research in January 2020.

The specified percentage identity cutoff of chains was 25%, the resolution cutoff was 2 Å, and the R-factor cutoff was 0.25. The specified cutoffs except resolution cutoff were the same as for proteins used for the development of PBs prediction models. The specified protein resolution cutoff for the development of new structural alphabets was 2 Å instead of 2.5 Å in order to use only proteins with more precisely experimentally determined structures. The total number of chains in the used dataset was 9,287. Coordinates of protein chain atoms were obtained from the PDB database [7]. Dihedral angles (ϕ and ψ) were calculated for each amino acid in a chain based on PDB data. DSSP program [53] was used to assign 8-state secondary structures to amino acids in used chains. Spider3 [44] was used for the prediction of structure properties (backbone angles, 3-state secondary structure

and accessible surface area (ASA)) for amino acid sequences of chains in a dataset.

After merging the results obtained by calculating dihedral angles, DSSP program and Spider3, the following properties for each amino acid in protein chains were extracted:

- one-letter amino acid name;
- true ϕ and ψ angles;
- secondary structure assigned by DSSP (true secondary structure);
- predicted ϕ and ψ angles, secondary structure and ASA.

The chains in the dataset were partitioned in training and test parts with a ratio of 50:50. The fragments of chains in the training part were used for clustering, i.e. development of SAs, and for building the corresponding prediction models of obtained SAs. Fragments of chains in the test part were used for the evaluation of obtained SAs prediction models.

For the development of SAs, fixed-length fragments of length from 4 to 10 amino acids were extracted from protein chains in the dataset and grouped by the length. Amino acids of a fragment were described with dihedral angles. $2 * (L - 1)$ dihedral angles were extracted $(\psi_i, \phi_{i+1}, \psi_{i+1}, ..., \phi_{i+L-2}, \psi_{i+L-2}, \phi_{i+L-1})$ for a fragment of L amino acids with the first amino acid at position i in a sequence.

Each fragment in each group, used for the construction of the SAs, had the corresponding fragment in a dataset used for building the prediction models of SAs. Fragments used for building prediction models were described with one-letter amino acids and predicted values by Spider3 (backbone angles, secondary structures and ASAs).

5.3 Development of structural alphabets

For extraction of prototypes from the local structure of proteins, the TwoStep clustering algorithm in SPSS Modeler was applied on each L fixed-length fragment group ($L = 4, 5, ..., 10$). Best results were obtained using log-likelihood as a distance measure in the search of dense regions in a dataset and Bayesian Information Criterion (BIC) for the determination of the best number of clusters. In order to find the optimal number of prototypes for each of the fixed-length

fragment groups, a clustering algorithm was applied repeatedly with different restrictions for the minimal and maximal number of clusters. The difference between the used limits for the minimal and the maximal number of clusters was 10. As a minimal number of clusters, the values from 10 to 100 with a step of 10 were used. A difference of 10 between a minimal and a maximal number of clusters was used to compare how the structural alphabet's ability to approximate protein structure and the accuracy of prediction models of extracted prototypes change with the increasing number of prototypes. The results should point to the range in which the optimal number of prototypes is. For each used restriction (minimal cluster number, maximal cluster number), the minimal cluster number was determined as the best number of clusters. Also, a clustering algorithm was applied with a restriction of 16 clusters on each of the fixed-length fragment groups. Based on the results of each clustering on fragment groups, one SA was defined. Thus, 11 SAs for each used fragment length were obtained. One prototype of a SA was defined based on members of one cluster. Each dihedral angle $\hat{\alpha}$ ($\hat{\alpha} \in (\psi_1, \phi_2, \psi_2, ..., \phi_{L-1}, \psi_{L-2}, \phi_L)$) of a prototype was calculated as mean of angles of cluster members with which the prototype is associated. The mean angle was calculated using

$$\hat{\alpha} = atan2(\frac{1}{M}\sum_{i=1}^{M}\sin\alpha_i, \frac{1}{M}\sum_{i=1}^{M}\cos\alpha_i) \tag{5.1}$$

where M is the number of fragments (members) in a cluster and α_i is the corresponding angle of a member i in a cluster. Each SA is labelled with a name L_n where L is a length of fragment and n is the size of SA, i.e. the number of prototypes. Each prototype of a SA is labelled with a number. Labels in one SA are from 1 to n. An amino acid sequence can be translated to labels of prototypes of a SA by processing overlapping fragments of length L from the N-terminus to the C-terminus and assigning to it the most similar prototype. The prototype label corresponds to the first amino acid in the fragment. The last $L-1$ amino acids in a sequence do not have the assigned prototype. A special label can be assigned to them so that the prototype sequence would be the same length as the amino acid sequence, as Z is assigned to the first two and last two amino acids in a sequence when the sequence is coded with PBs. The angles of the prototypes assigned to the fragments are used to determine the angles of amino acids in a sequence. The position of the amino acid in a fragment also corresponds to the position of the angles in the prototypes used to determine the angles of the amino

acid. To determine the angle of one amino acid in a sequence, the mean value of the corresponding angles of all prototypes to which it belongs is used.

As expected, in the analysis of the graphical presentation of ϕ and ψ angles for each position of amino acid in prototypes per SA using Ramachandran plot, it can be observed that prototypes cover most the regions of $\alpha - helix$ and $\beta - sheet$ in 3-state secondary structure (see Figure 3.4). The sparse regions are beginning to be covered when the size of SA is larger (usually from 50). Graphical presentation of ϕ and ψ angles for each position of amino acid in prototypes per SA is shown in Appendix in Figures from 7.9 to 7.21.

For each SA, *rmsda* between its prototypes and corresponding cluster members were calculated, as well as the size of clusters as a percentage of fragments in them. The minimal, maximal and average values of obtained *rmsda* and the minimal and maximal size of clusters for each constructed SA are shown in Table 5.1. In the analysis of SAs per group, it was observed that all SAs groups have in common:

- each SA in a group has at least one big cluster;
- the biggest decrease in the average *rmsda* is when the size of SA is increased from 10 16 or 16 to 20;
- as the size of SA increases, the average *rmsda* generally decreases, but each SAs group has a threshold in size from which the increase in the size of SA does not significantly affect the decrease in the mean *rmsda* (for example, for a group of SAs with length 5, that threshold is 30, since the mean *rmsda* for SA 5_30 is 34.36° and for SA 5_40 is 34.32°).

Each obtained SA has one big cluster, which is labelled with 1. The percentages of fragments covered by prototypes in SAs are shown in Figures from 7.1 to 7.7 in Appendix. In the analysis of the occurrence of DSSP states in obtained prototypes, it was observed that state H has the largest frequency of occurrence at each amino acid position in prototype 1. All ψ angles in prototypes with label 1 are around $-40°(+/-5°)$ and all ϕ angles are around $-65°(+/-5°)$. Frequencies of DSSP states per amino acid position in fragments cover by prototypes in SAs with 20 prototypes are shown in Figures from 7.22 to 7.35 in Appendix.

Parts of amino acid sequences without missing coordinates in the PDB file for at least 10 amino acids were coded in sequences of prototypes using all obtained SAs. For each amino acid sequence, *rmsda* was calculated between true dihedral

SA	min rmsda	max rmsda	avg rmsda	min cluster size (%)	max cluster size (%)
4_10	11.61	60.15	37.89	2.08	33.65
4_16	7.5	62.21	32.51	1.59	27.98
4_20	7.36	69.89	33.58	0.55	27.9
4_30	7.36	68	31.37	0.54	27.9
4_40	6.43	59.81	29.41	0.28	26.8
4_50	6.29	59.71	27.68	0.28	26.68
4_60	6.27	59.71	26.21	0.28	26.66
4_70	6.18	59.01	25	0.17	26.58
4_80	6.18	67.05	24.62	0.13	26.58
4_90	6.1	66.94	24.35	0.05	26.33
4_100	6.09	66.9	24.28	0.05	26.32
5_10	10.57	65.77	46.51	4.67	29.76
5_16	9.56	66.79	39.18	1.9	28.83
5_20	9.47	66.75	36.29	1.86	28.76
5_30	7.59	64.44	34.36	0.5	25.43
5_40	7.45	58.42	34.32	0.48	25.34
5_50	7.33	61.5	34.46	0.24	25.19
5_60	7.24	63.38	32.93	0.22	25.09
5_70	7.24	63.36	31.7	0.12	25.09
5_80	6.21	66.18	30.74	0.11	23.23
5_90	5.83	66.18	29.77	0.11	23.03
5_100	5.09	66.18	28.95	0.09	18.94
6_10	9.56	66.19	51.5	5.06	26.11
6_16	9.36	62.99	42.5	1.71	25.91
6_20	9.18	63.65	41.36	1.71	25.74
6_30	8.15	68.87	38.98	1.05	24.46
6_40	8.15	66.72	38.97	0.32	24.46
6_50	6.74	66.65	37.06	0.31	20.46
6_60	6.51	69.31	36.45	0.29	20.34
6_70	6.51	65.82	36.48	0.21	20.34
6_80	6.46	67.66	35.5	0.21	20.29
6_90	6.46	67.36	35.09	0.2	20.29
6_100	6.46	65.23	34.64	0.16	20.29
7_10	19.54	65.85	54.74	3.89	30.93
7_16	9.8	65.72	46.37	1.85	23.54
7_20	9.74	62.92	44.16	1.84	23.51
7_30	9.49	78.43	43.26	0.88	23.32
7_40	7.08	78.44	39.93	0.88	19.18
7_50	7.01	78.24	39.83	0.47	19.14
7_60	6.95	78.04	39.2	0.23	19.08
7_70	6.88	75.23	38.74	0.22	19.02
7_80	6.83	74.78	38.53	0.22	18.49
7_90	6.83	71.59	38.02	0.2	18.49
7_100	6.66	71.39	36.89	0.2	18.26
8_10	9.95	65.07	53.21	4.6	21.1
8_16	9.74	64.75	48.19	1.92	20.93
8_20	9.48	63.27	45.36	1.57	20.72
8_30	9.37	67.49	45.11	1.35	20.66
8_40	9.29	68.87	43.36	0.94	20.56
8_50	9.27	74.06	43.95	0.47	20.55
8_60	7.76	73.88	42.42	0.41	18.1
8_70	7.59	74.43	41.37	0.4	17.98
8_80	7.53	74.31	41	0.24	17.93
8_90	7.47	73.88	41.27	0.21	17.88
8_100	7.46	71.25	40.78	0.21	17.86
9_10	10.31	69.09	53.9	4.75	18.82
9_16	9.98	64.7	52.25	3.1	18.61
9_20	9.48	64.59	47.65	1.61	18.29
9_30	9.21	66.33	45.34	1.59	18.08
9_40	9.04	68.32	45.16	1.08	17.94
9_50	9.03	72.88	44.73	0.48	17.94
9_60	7.25	74.94	44.05	0.41	14.73
9_70	7	74.86	43.3	0.33	14.51
9_80	7	73.58	43.69	0.25	14.51
9_90	7	73.35	43.66	0.23	14.51
9_100	6.9	73.58	42.64	0.22	14.4
10_10	18.93	68.87	56.95	3.87	21.75
10_16	10.91	64.8	53.34	2.53	16.77
10_20	10.91	64.8	49.06	1.61	16.77
10_30	10.08	66.72	46.38	1.5	16.29
10_40	9.55	73.85	47.22	0.95	15.99
10_50	9.21	73.94	45.63	0.87	15.79
10_60	9.16	73.88	45.58	0.42	15.75
10_70	7.67	73.8	44.51	0.34	12.94
10_80	7.64	73.62	44.95	0.24	12.93
10_90	7.61	73.68	44.31	0.24	12.92
10_100	7.61	73.64	44.29	0.24	12.92

Table 5.1: Minimal, maximal and average value of *rmsda* (in °) calculated between the cluster prototypes and corresponding cluster members and the minimal and maximal size of clusters (in percent) for each constructed SA

angles and angles assigned based on SA prototypes. Also, *mae* for ϕ and ψ angles were calculated based on true and assigned angles. Minimal, maximal and average value of calculated *rmsda* and *mae* for ϕ and ψ angles are shown in Table 5.2. By

increasing the size of SAs from 10 to 100, the average value of *rmsda* decreases from 10° to 14° in groups of SAs with length up to 7, while for groups of SAs with length from 8 to 10, *rmsda* decreases by up to 5°. A significant reduction in *rmsda* in SAs with longer lengths requires a larger number of prototypes than 100. It is noticeable that the increase of SA size in each group of SA with the same length significantly affects the decrease of *mae* for ψ angle, but it does not affect the *mae* of ϕ angle.

Based on the comparison of calculated *mae* of angles for constructed SAs and *mae* reported for Spider3 predictor in [44] (18.3° for ϕ and 27.0° for ψ), it can be concluded that obtained SAs with *mae* less than 27.0° could be useful for better prediction of protein structure.

5.4 Development of predictors for structural alphabets

Besides the ability of good approximation of protein structure, the usability of a structural alphabet also depends on the availability of predictors of its prototypes for a given amino acid sequence. For comparison SAs based on the accuracy of prototype predictor, a classification model was developed for each SA using a similar approach based on a fixed-length fragment format database which was used for building PBs prediction models. The length of the sliding window was the same as the length of the prototypes, i.e. no information on amino acids in the neighbourhood outside of a fragment was used. The target attribute contained the assigned prototype of a fragment. Models for SAs prediction were developed using neural networks implementation in library Keras and fixed-length fragments. Prediction models were constructed using two hidden layers. Optimal values of algorithm parameters were determined for each SA. Obtained optimal activation function and nodes per hidden layer are shown in Table 5.3 for each SA, as well as calculated accuracy (i.e. Q_k were k is the number of prototypes) on training and test part of the dataset. It could be noticed that the length of prototypes doesn't have a big influence on the accuracy of obtained SAs prediction models. The increase of the number of prototypes in a SA decreases the accuracy of a prediction model of SA prototypes.

In the comparison of prediction models of the SAs with the same size S ($S = 10, 20, ...100$) based on recall of prototypes, it can be observed that prediction

SA	min rmsda	max rmsda	avg rmsda	mae for ψ	mae for ϕ
4_10	7.81	92.39	53.04	32.46	37.04
4_16	6.92	84.35	47.36	21.93	36.83
4_20	6.91	85.22	47.04	21.64	36.73
4_30	6.91	86.02	45.58	19.96	36.51
4_40	6.91	81.99	45.55	18.53	37.32
4_50	6.91	80.14	44.84	17.51	37.37
4_60	6.91	78.8	44.69	17.32	37.31
4_70	6.91	79.2	44.1	15.77	37.41
4_80	6.91	79.9	43.94	15.53	37.31
4_90	6.92	79.89	43.59	15.1	37.32
4_100	6.92	79.53	43.38	14.78	37.31
5_10	9	99.69	58.68	41.79	37.98
5_16	8.91	97.36	54.78	35.57	37.67
5_20	8.9	97	52.25	30.23	37.81
5_30	8.4	94.09	50.32	26.24	37.93
5_40	8.4	95.84	48.99	24.32	37.9
5_50	8.4	94.02	48.13	22.76	38.21
5_60	8.4	94.09	47.6	21.68	38.21
5_70	8.4	94.62	47.43	21.5	38.13
5_80	8.28	93.05	46.72	20.1	38.12
5_90	8.36	94.69	46.39	19.6	38.15
5_100	8.21	94.79	46.29	19.35	38.27
6_10	9.77	97.56	61.83	47.86	37.88
6_16	9.76	98.59	57.38	39.89	37.86
6_20	9.75	98.39	56.75	37.81	37.81
6_30	9.51	96.74	53.86	31.23	39.06
6_40	9.51	92.15	52.9	29.81	38.66
6_50	9.28	91.63	51.66	27.97	38.7
6_60	9.34	95.6	50.78	26.1	39.1
6_70	9.34	94.84	50.67	25.81	39.11
6_80	9.34	90.83	49.43	23.57	39.1
6_90	9.34	90.8	49.02	23.36	39.11
6_100	9.34	90.21	48.43	22.3	39.11
7_10	10.25	93.35	65.52	55.4	37.12
7_16	10.52	93.02	60.74	46.21	38.58
7_20	10.52	95.12	58.82	41.74	38.45
7_30	10.51	97.29	55.67	35.79	38.44
7_40	9.91	90.03	53.5	31.51	38.56
7_50	9.91	90.87	53.76	31.7	39.14

SA	min rmsda	max rmsda	avg rmsda	mae for ψ	mae for ϕ
7_60	9.91	89.72	52.91	30.29	39.14
7_70	9.91	96.03	52.44	29.44	39.1
7_80	9.93	95.55	52.24	29.11	38.89
7_90	9.93	94.9	52.32	29.03	39.2
7_100	9.92	94.7	51.79	27.87	39.2
8_10	11.32	89.9	58.51	48.74	33.42
8_16	11.28	89.97	59.75	44.02	38.46
8_20	11.28	91.22	56.5	37.6	38.37
8_30	11.28	94.29	55.81	36.52	38.21
8_40	11.3	95.8	54.73	34.3	38.22
8_50	11.3	94.53	54.5	33.15	38.69
8_60	10.77	94.54	54.54	33.15	38.97
8_70	10.77	93.35	54.14	31.68	39.39
8_80	10.77	92.72	53.53	30.96	39.37
8_90	10.77	91.75	53.56	30.44	39.79
8_100	10.77	95.23	53.56	30.45	39.76
9_10	12.12	87.9	58.19	47.35	33.85
9_16	12.1	92.59	60.49	47.53	37.42
9_20	12.11	93.03	61.26	46.52	39.22
9_30	12.11	95.97	58.98	41.14	39.07
9_40	12.12	92.5	57.48	39.15	39.13
9_50	12.12	96.44	56.66	37.21	39.15
9_60	11.52	91.39	55.32	34.07	39.37
9_70	11.52	92.15	55.34	34.12	39.34
9_80	11.52	96.07	55.07	33.21	39.4
9_90	11.52	96.55	55.12	33.14	39.66
9_100	11.52	96.13	54.85	32.7	39.67
10_10	12.97	84.24	62.85	54.65	34.24
10_16	12.89	87.87	61.25	50.81	36.77
10_20	12.89	88.41	62.17	49.72	38.59
10_30	12.88	91.18	59.97	44.41	38.43
10_40	12.91	90.76	59.19	43.19	38.26
10_50	12.91	90.27	58.83	42.05	38.34
10_60	12.9	90.43	58.41	40.84	38.29
10_70	12.27	90.1	57.58	39.39	38.43
10_80	12.27	88.95	57.8	39.1	38.69
10_90	12.27	93	57.32	37.99	39.02
10_100	12.27	93.06	57.39	37.59	39.33

Table 5.2: *rmsda* and *mae* between true angles and angles calculated based on SAs prototypes for amino acid sequences in a training part of the dataset. Values are in °.

SA	number of nodes per layer	activation function	accuracy on training part	accuracy on test part
4_10	256	relu	77.26	77.21
4_16	128	relu	69.21	69.08
4_20	256	relu	68.54	68.45
4_30	512	relu	63.4	63.22
4_40	512	relu	58.68	58.58
4_50	512	relu	56.93	56.79
4_60	256	relu	54.72	54.6
4_70	512	relu	52.99	52.9
4_80	512	relu	52.27	52.26
4_90	512	relu	51.18	51.15
4_100	128	tanh	50.24	50.27
5_10	512	relu	75.51	75.44
5_16	256	relu	72.83	72.54
5_20	128	relu	69.16	69.35
5_30	256	relu	65.77	65.67
5_40	256	relu	62.39	62.33
5_50	128	tanh	61.08	60.97
5_60	64	tanh	58.41	58.27
5_70	512	relu	56.81	56.57
5_80	256	relu	54.44	54.3
5_90	512	relu	53.03	53.02
5_100	512	relu	48.91	48.86
6_10	64	tanh	75.82	75.75
6_16	512	relu	71.98	71.71
6_20	256	relu	70.68	70.58
6_30	512	relu	65.75	65.7
6_40	256	tanh	64.46	64.37
6_50	512	tanh	60.56	60.7
6_60	256	relu	59.95	60.03
6_70	512	relu	58.23	58.19
6_80	512	relu	57.43	57.38
6_90	512	relu	56.83	56.79
6_100	256	relu	55.55	55.42
7_10	512	relu	76.83	76.71
7_16	128	tanh	71.04	70.96
7_20	256	tanh	70.16	70.08
7_30	256	relu	67.31	67.42
7_40	128	relu	63.32	63.35
7_50	256	relu	61.1	60.95
7_60	256	relu	59.9	59.94
7_70	256	relu	59.23	59.28
7_80	512	relu	57.95	57.98
7_90	512	relu	56.28	56.1
7_100	256	relu	55.13	55.11
8_10	64	tanh	76.33	76.29
8_16	512	relu	70.32	70.03
8_20	256	tanh	69.33	69.23
8_30	512	relu	66.26	66.05
8_40	512	relu	64.4	64.3
8_50	128	relu	63.13	63.22
8_60	256	relu	59.86	59.75
8_70	512	relu	58.59	58.49
8_80	512	relu	57.91	57.75
8_90	512	relu	57.42	57.26
8_100	512	relu	56.18	56.03
9_10	256	relu	75.5	75.46
9_16	256	relu	70.64	70.16
9_20	512	relu	70.11	70.04
9_30	512	relu	65.74	65.73
9_40	256	relu	62.97	62.94
9_50	512	relu	61.72	61.5
9_60	512	relu	58.6	58.55
9_70	512	relu	57.68	57.64
9_80	512	relu	57.11	56.99
9_90	512	relu	56.43	56.21
9_100	128	tanh	55.05	54.95
10_10	256	sigmoid	76.51	76.49
10_16	512	sigmoid	70.92	70.8
10_20	512	sigmoid	69.41	69.28
10_30	512	sigmoid	65.3	65.48
10_40	512	sigmoid	63.11	63.11
10_50	256	sigmoid	61.43	61.28
10_60	512	sigmoid	60.62	60.5
10_70	512	sigmoid	58.02	58.13
10_80	512	sigmoid	57.23	57.27
10_90	512	sigmoid	56.39	56.22
10_100	512	sigmoid	55.24	55.09

Table 5.3: Description of the best-obtained prediction model of prototypes per SA. For each SA prediction model is presented architecture (activation function and the number of nodes per hidden layer), accuracy on the training part and accuracy on the test part.

models for longer SAs better predict less represented prototypes, since the minimal and average recall for SAs of the same size S increase with the increase of the SA length. The minimal, maximal and average values for precision and recall of prototypes are shown in Tables 5.4 and 5.5.

As shown in Table 5.2, SAs of length 4 with size from 16 to 100 have the smallest *mea* for ψ and smaller average *rmsda* comparing to other SAs. For the analysis of predicted angles of amino acids in chains using these SAs, SA prototypes were predicted for parts of amino acid sequences in the test part of the dataset without missing coordinates for at least 10 amino acids. *mae* of ϕ and ψ angles were calculated for true angles and angles assigned based on predicted prototypes. Results are shown in Table 5.6. *mae* of ϕ is similar for all SAs that are compared and is almost the same as *mae* calculated for ϕ angles assigned to amino acids based on the closest prototype of a SA (true prototype). *mae* of ψ decreases with increasing the SA size.

Developed prediction models for the SAs constructed with the TwoStep algorithm during the research (with the 100 as maximal size) didn't achieve *mae* less than 27° for predicted ψ angles. Based on the results, *mae* for predicted ψ angles decreases when increasing the SA size, so the goal could be achieved by applying the described clustering procedure for a larger number of prototypes or developing the more accurate SAs prediction models.

5.5 Comparison of structural alphabet 5_16 and Protein Blocks

Prototypes of SA 5_16 were compared with prototypes of Protein Blocks since both structural alphabets are defined over fragments of length 5 and have the same number of prototypes.

For each prototype of SA 5_16, the following values are extracted:

- average *rmsda* between prototype's dihedral angles and dihedral angles of its cluster members;
- frequency, i.e. the number of fragments in the training part of the dataset to which the prototype was assigned;

SA	min precision on training part	max precision on training part	avg precision on training part	min recall on training part	max recall on training part	avg recall on training part	min precision on test part	max precision on test part	avg precision on test part	min recall on test part	max recall on test part	avg recall on test part
4_10	0.59	0.89	0.71	0.49	0.93	0.68	0.59	0.89	0.71	0.5	0.93	0.68
4_16	0.52	0.87	0.63	0.38	0.93	0.59	0.52	0.87	0.63	0.38	0.93	0.6
4_20	0.25	0.87	0.59	0.03	0.93	0.54	0.24	0.87	0.59	0.03	0.93	0.54
4_30	0.24	0.83	0.52	0.06	0.95	0.47	0.22	0.83	0.52	0.06	0.95	0.47
4_40	0.27	0.86	0.48	0.04	0.94	0.43	0.26	0.86	0.48	0.04	0.93	0.43
4_50	0.13	0.85	0.44	0.01	0.94	0.39	0.16	0.85	0.44	0.01	0.94	0.39
4_60	0.15	0.81	0.42	0.01	0.96	0.35	0.15	0.81	0.42	0.01	0.96	0.35
4_70	0.13	0.83	0.4	0.01	0.95	0.34	0.13	0.83	0.41	0.01	0.95	0.34
4_80	0.14	0.82	0.38	0.01	0.95	0.32	0.13	0.82	0.38	0	0.95	0.32
4_90	0.12	0.83	0.37	0.01	0.95	0.31	0.09	0.83	0.37	0.01	0.94	0.31
4_100	0	0.88	0.35	0	0.9	0.28	0	0.87	0.34	0	0.9	0.28
5_10	0.53	0.9	0.71	0.55	0.93	0.7	0.53	0.9	0.71	0.55	0.93	0.7
5_16	0.53	0.89	0.66	0.44	0.94	0.62	0.53	0.89	0.66	0.44	0.94	0.62
5_20	0.4	0.89	0.62	0.29	0.94	0.59	0.4	0.89	0.63	0.29	0.94	0.59
5_30	0.3	0.83	0.58	0.04	0.95	0.52	0.27	0.83	0.58	0.03	0.94	0.52
5_40	0.26	0.84	0.53	0.02	0.95	0.48	0.24	0.84	0.52	0.01	0.94	0.48
5_50	0.17	0.86	0.5	0.02	0.93	0.44	0.16	0.86	0.5	0.02	0.92	0.45
5_60	0.16	0.84	0.48	0	0.94	0.42	0.14	0.84	0.48	0	0.93	0.42
5_70	0	0.86	0.44	0	0.92	0.41	0	0.86	0.44	0	0.92	0.41
5_80	0	0.84	0.43	0	0.92	0.39	0	0.83	0.43	0	0.92	0.39
5_90	0.18	0.82	0.42	0	0.93	0.37	0.14	0.82	0.42	0	0.93	0.37
5_100	0	0.76	0.4	0	0.95	0.35	0	0.76	0.4	0	0.95	0.35
6_10	0.57	0.91	0.73	0.54	0.93	0.71	0.57	0.91	0.73	0.54	0.93	0.71
6_16	0.5	0.9	0.68	0.41	0.93	0.65	0.51	0.9	0.68	0.41	0.93	0.65
6_20	0.5	0.89	0.65	0.39	0.94	0.62	0.5	0.89	0.65	0.39	0.94	0.62
6_30	0.44	0.87	0.6	0.36	0.94	0.56	0.45	0.87	0.6	0.35	0.94	0.57
6_40	0.35	0.89	0.55	0.2	0.94	0.51	0.34	0.89	0.56	0.2	0.93	0.52
6_50	0.14	0.85	0.54	0.05	0.92	0.47	0.15	0.85	0.54	0.05	0.92	0.48
6_60	0.22	0.86	0.51	0.01	0.91	0.45	0.21	0.86	0.51	0.01	0.91	0.45
6_70	0.2	0.84	0.49	0.03	0.93	0.44	0.18	0.84	0.48	0.02	0.93	0.44
6_80	0.15	0.85	0.48	0.01	0.92	0.42	0.17	0.85	0.48	0.01	0.92	0.42
6_90	0.1	0.86	0.46	0	0.91	0.41	0.12	0.86	0.46	0.01	0.91	0.41
6_100	0.1	0.85	0.44	0	0.92	0.39	0	0.85	0.44	0	0.92	0.39
7_10	0.55	0.89	0.73	0.57	0.93	0.71	0.55	0.89	0.73	0.57	0.93	0.71
7_16	0.54	0.89	0.68	0.4	0.93	0.66	0.54	0.9	0.68	0.4	0.93	0.66
7_20	0.5	0.9	0.66	0.39	0.93	0.64	0.5	0.9	0.66	0.4	0.93	0.64
7_30	0.45	0.88	0.61	0.32	0.94	0.58	0.45	0.88	0.61	0.32	0.94	0.58
7_40	0.45	0.85	0.58	0.27	0.91	0.54	0.45	0.85	0.58	0.28	0.91	0.55
7_50	0.4	0.83	0.56	0.33	0.92	0.52	0.4	0.84	0.56	0.34	0.91	0.53

Table 5.4: Classification report of best-obtained prediction models for SAs with length 4, 5, 6 and 7 with up to 50 prototypes

SA	min precision on training part	max precision on training part	avg precision on training part	min recall on training part	max recall on training part	avg recall on training part	min precision on test part	max precision on test part	avg precision on test part	min recall on test part	max recall on test part	avg recall on test part
7_60	0.24	0.84	0.53	0.07	0.92	0.49	0.23	0.84	0.54	0.07	0.91	0.49
7_70	0.21	0.86	0.51	0	0.88	0.46	0.19	0.86	0.51	0	0.88	0.47
7_80	0.15	0.8	0.5	0.04	0.93	0.46	0.14	0.8	0.5	0.04	0.93	0.46
7_90	0.17	0.82	0.48	0.01	0.91	0.44	0.18	0.82	0.48	0.01	0.9	0.45
7_100	0.17	0.86	0.47	0.02	0.88	0.43	0.14	0.86	0.47	0.02	0.88	0.44
8_10	0.63	0.91	0.75	0.67	0.91	0.73	0.63	0.91	0.75	0.68	0.91	0.74
8_16	0.47	0.91	0.69	0.45	0.91	0.67	0.47	0.91	0.69	0.45	0.91	0.68
8_20	0.52	0.91	0.67	0.36	0.92	0.66	0.51	0.91	0.67	0.35	0.92	0.66
8_30	0.45	0.9	0.62	0.31	0.93	0.6	0.45	0.9	0.63	0.31	0.93	0.6
8_40	0.36	0.89	0.59	0.32	0.93	0.56	0.36	0.89	0.59	0.32	0.93	0.56
8_50	0.36	0.88	0.56	0.24	0.94	0.52	0.37	0.88	0.57	0.24	0.94	0.53
8_60	0.32	0.86	0.54	0.03	0.91	0.5	0.28	0.86	0.54	0.02	0.91	0.51
8_70	0.18	0.83	0.53	0.12	0.94	0.49	0.18	0.83	0.53	0.11	0.93	0.49
8_80	0.19	0.84	0.52	0.08	0.92	0.48	0.18	0.84	0.52	0.08	0.92	0.48
8_90	0.24	0.83	0.51	0.04	0.93	0.46	0.21	0.83	0.51	0.04	0.93	0.47
8_100	0.21	0.83	0.49	0.02	0.92	0.45	0.2	0.84	0.49	0.02	0.92	0.45
9_10	0.66	0.9	0.74	0.62	0.9	0.73	0.66	0.9	0.74	0.63	0.9	0.74
9_16	0.54	0.9	0.69	0.5	0.91	0.68	0.55	0.9	0.69	0.49	0.91	0.68
9_20	0.5	0.9	0.69	0.5	0.91	0.66	0.5	0.9	0.69	0.49	0.91	0.66
9_30	0.4	0.89	0.63	0.33	0.93	0.61	0.4	0.89	0.64	0.33	0.92	0.61
9_40	0.4	0.87	0.59	0.29	0.94	0.56	0.41	0.88	0.59	0.3	0.93	0.57
9_50	0.35	0.89	0.57	0.26	0.92	0.54	0.35	0.89	0.58	0.27	0.92	0.54
9_60	0.33	0.83	0.55	0.24	0.9	0.52	0.33	0.83	0.56	0.24	0.9	0.52
9_70	0.3	0.84	0.54	0.23	0.86	0.5	0.29	0.85	0.54	0.22	0.86	0.51
9_80	0.27	0.84	0.52	0.21	0.88	0.48	0.27	0.84	0.52	0.2	0.88	0.49
9_90	0.2	0.82	0.51	0	0.89	0.47	0.18	0.82	0.51	0	0.89	0.48
9_100	0.22	0.85	0.5	0.02	0.92	0.46	0.19	0.85	0.5	0.02	0.92	0.46
10_10	0.7	0.91	0.75	0.61	0.89	0.73	0.7	0.91	0.75	0.62	0.89	0.73
10_16	0.55	0.88	0.71	0.42	0.89	0.68	0.55	0.89	0.71	0.42	0.89	0.68
10_20	0.47	0.89	0.68	0.37	0.89	0.67	0.48	0.89	0.69	0.37	0.89	0.67
10_30	0.37	0.88	0.65	0.39	0.91	0.62	0.38	0.88	0.65	0.39	0.91	0.63
10_40	0.37	0.89	0.6	0.33	0.91	0.58	0.37	0.89	0.6	0.33	0.91	0.58
10_50	0.3	0.88	0.57	0.3	0.92	0.55	0.31	0.88	0.58	0.3	0.92	0.55
10_60	0.32	0.87	0.56	0.29	0.92	0.54	0.33	0.88	0.57	0.3	0.92	0.54
10_70	0.32	0.83	0.55	0.24	0.89	0.52	0.33	0.83	0.55	0.25	0.88	0.53
10_80	0.31	0.81	0.53	0.25	0.91	0.5	0.31	0.81	0.53	0.26	0.91	0.51
10_90	0.26	0.85	0.52	0.13	0.93	0.48	0.27	0.85	0.52	0.13	0.92	0.49
10_100	0.29	0.84	0.5	0.09	0.87	0.48	0.28	0.84	0.5	0.08	0.86	0.48

Table 5.5: Classification report of best-obtained prediction models for SAs with length and 7 with more than 50 prototypes and SAs with length 8, 9 and 10

SA	**mae for ψ**	**mae for ϕ**
4_16	31.32	36.78
4_20	30.92	36.84
4_30	30.35	36.95
4_40	29.33	37.47
4_50	28.84	37.12
4_60	29.26	37.22
4_70	28.34	36.78
4_80	28.02	37.01
4_90	27.92	37.53

Table 5.6: mae for ψ and ϕ angles calculated using the predicted prototypes assigned by developed SA prediction models for SAs of length 4 and sizes from 20 to 90

- coarse classification based on corresponding secondary structures in which prototype most frequently occurs. Secondary structure classification was done based on the frequencies of DSSP states per amino acid position in cluster members of its associated cluster. Figure 5.2 and 5.3 show frequencies of DSSP states per amino acid position in members of a cluster associated with each prototype of SA 5_16.
- the most similar PB based on *rmsda* calculated between SA 5_16 prototypes and PBs.

The obtained values are shown in Table 5.7.

Comparing the prototypes of SA 5_16 (see Table 5.7) and PBs (see Table 3.3), it can be noticed that six prototypes of SA 5_16 have very similar prototypes in Protein Blocks by *rmsda* measure, occurence frequency and coarse secondary structure classification:

- prototype 1 and PB *m*: *rmsda* between them is 0.86°, they are associated with α-helix and occurences of 1 and *m* are 28.83% and 30.22%, respecitvely.

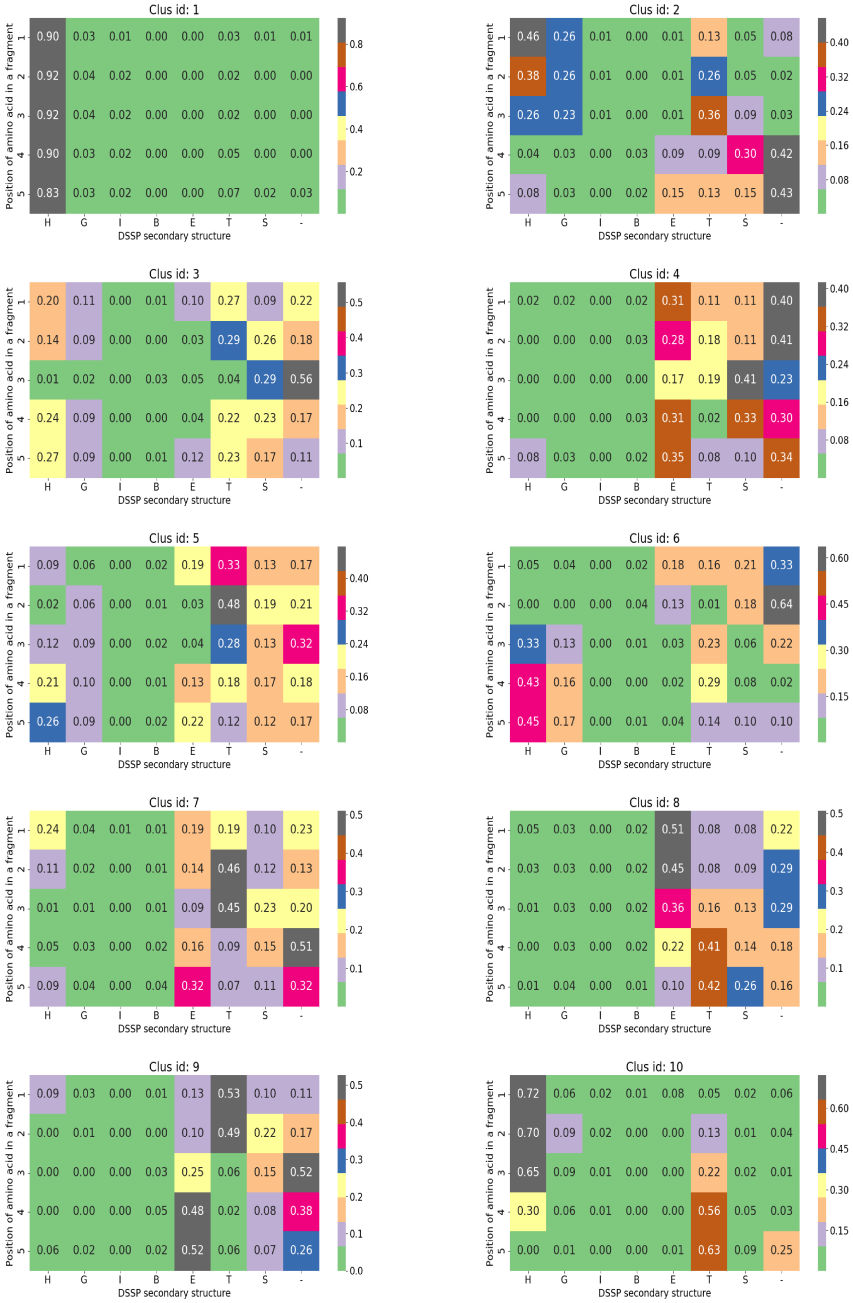

Figure 5.2: Frequencies of DSSP states per amino acid position in cluster members for prototypes of SA **5_16** with id from 1 to 10

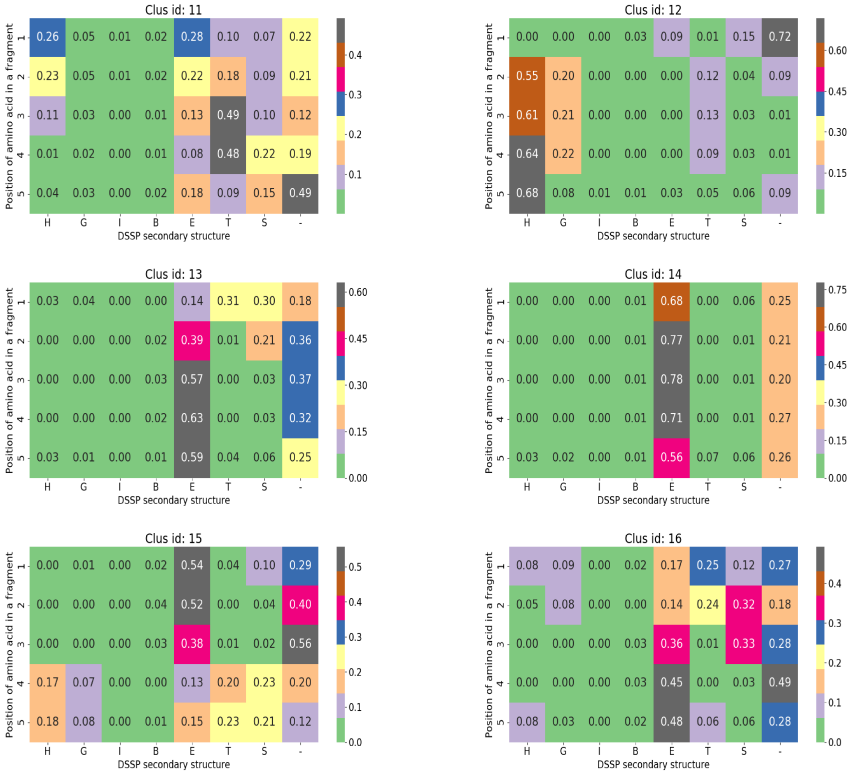

Figure 5.3: Frequencies of DSSP states per amino acid position in cluster members for prototypes of SA **5_16** with id from 11 to 16

Prototype	***rmsda*(°)**	**Frequency(%)**	**Secondary structure**	**Most similar PB (rmsda(°))**
1	9.56	28.83	α	m(**0.86**)
2	30.21	3.4	C-cap α	p(55.91)
3	40.43	2.75	mainly coil	m(62.56)
4	44.66	4.29	mainly coil	b(7.23)
5	66.79	2.31	mainly coil	l(58.47)
6	40.77	5.94	N-cap α	k(11.51)
7	58.41	4.63	mainly coil	i(49.3)
8	55.79	3.85	C-cap β	e(10.21)
9	46.92	3.62	N-cap β	a(**5.77**)
10	24.71	1.9	C-cap α	n(**2.34**)
11	62.67	5.09	C-cap α/β	h(46.11)
12	20.24	3.7	N-cap α	l(**4.92**)
13	30.58	5.59	N-cap β	c(16.74)
14	22.72	14.81	β	d(**3.37**)
15	32.17	4.6	C-cap β	f(**6.79**)
16	40.22	4.7	coil / N-cap β	c(46.57)

Table 5.7: Description of SA 5_16 prototypes

- prototype 10 and PB n: *rmsda* between them is 2.34°, they are associated with α-helix C-caps and of 10 and n are 1.9% and 1.99%, respectively.
- prototype 14 and PB d: *rmsda* between them is 3.37°, they are associated with β-sheet and occurences of 14 and d are 14.81% and 18.85%, respectively.
- prototype 12 and PB l: *rmsda* between them is 4.92°, they are associated with α-helix N-caps and of 12 and l are 3.7% and 5.46%, respectively.
- prototype 9 and PB a: *rmsda* between them is 5.77°, they are associated with β-sheet N-caps and of 9 and a are 3.62% and 3.89%, respecitvely.
- prototype 15 and PB f: *rmsda* between them is 6.79°, they are associated with β-sheet C-caps and of 15 and f are 4.6% and 6.68%, respecitvely.

For all listed prototype pairs except pair 9-a apply that SA 5_16 prototype has smaller average *rmsda* between its dihedral angles and dihedral angles of members of its associated cluster (for example, average *rmsda* of prototype 1 is 9.56° and average *rmsda* of PB m is 15°).

Prototype	**precision**	**recall**	F_1
1	0.89	0.94	0.91
2	0.58	0.44	0.50
3	0.56	0.46	0.50
4	0.53	0.45	0.48
5	0.59	0.51	0.55
6	0.60	0.67	0.63
7	0.72	0.72	0.72
8	0.66	0.67	0.66
9	0.71	0.71	0.71
10	0.79	0.68	0.73
11	0.75	0.73	0.74
12	0.70	0.66	0.68
13	0.64	0.58	0.61
14	0.70	0.84	0.76
15	0.58	0.48	0.52
16	0.63	0.45	0.53

Table 5.8: Classification report of aplied prediction model for SA 5_16 prototypes on test part

PBs associated mainly with coil have better average *rmsda* than SA 5_16 prototypes associated mainly with coil; average *rmsda* for PBs associated with the coil is in the range from 43.40° to 50.60°, while for SA 5_16 prototypes average *rmsda* is in the range from 40.43° to 66.79°.

The quality of the performance of the prediction model for SA 5_16 on the test part of the used dataset is shown in Table 5.8. Calculated precision, recall and F_1 are shown for each prototype.

Performance of PBs prediction model *PBC5.0d* (see Figure 4.5) was compared with the performance of prototype prediction model for SA 5_16, which was developed in the research. Compared prototype prediction models behave similarly for the most frequent and similar prototypes of these two structural alphabets (pairs 1-*m* and 14-*d*): recall and precision for prototype 1 are 0.94 and 0.89, respectively, while recall and precision for PB *m* are 0.94 and 0.90, and recall and precision for prototype 14 are 0.84 and 0.70, respectively, while recall and precision for PB *d* are 0.85.

Prototype prediction models have similar performance for prototypes associated mainly with coils. Precision for SA 5_16 prototypes 3, 4, 5 and 7 is in the

range from 0.53 to 0.72, and recall is in the range from 0.45 to 0.72. Precision for PBs from *g* to *j* is in the range from 0.6 to 0.7, and recall is in the range from 0.42 to 0.66.

It can be concluded that the applied procedure for the construction of new structural alphabets can be used as the base for the construction of new structural alphabets since its application on fragments of length 5 for 16 clusters managed to extract several prototypes which have very close prototypes by *rmsda* and coarse secondary structure classification in Protein Blocks, which is a structural alphabet that has been confirmed as very useful in various studies. Also, the distribution of prototypes of SA 5_16 and Protein Blocks by the coarse secondary structure classification is similar.

6 Conclusion

This dissertation presents one possible way of improving the problem of predicting structural alphabet (the set of prototypes of the local structure of proteins) for an amino acid sequence using data mining methods. New models for SA Protein Blocks prediction, developed using different data mining approaches and several classification algorithms are described. Developed models for Protein Blocks prediction, compared with the previously reported PBs predictors, use different information about the amino acid sequence as input. In conjunction with amino acid sequences, results of available protein structure predictors for backbone angles, secondary structure, accessible surface area and potential protein intrinsically disordered regions, as well as the locations of certain types of repeats in amino acid sequences were used for building new models for Protein Blocks prediction. It was shown that usage of information of a protein chain that can be predicted or determined based on the amino acid sequence can contribute to the development of more accurate PBs prediction models. The best-obtained models have accuracy from 74% to 80%, while the best-published accuracy of previously developed PBs predictors is 69%. Also, the best obtained Protein Blocks prediction model, built using the C5.0 algorithm, has the best recall for each prototype except for one, in comparison with the reported recall per prototype for previously developed PBs predictors. Analysis of the performance of the obtained PBs prediction models per Protein Block (prototype) shown that more accurate predictions for individual Protein Block can be obtained using models that are not globally the best by accuracy. Improved PBs prediction models developed in the thesis could be useful for bioinformatics studies based on Protein Blocks. For the development of a new structural alphabet in order to create a base for the development of new predictors of local protein structures, the clustering algorithm TwoStep was applied on fragments with the length from 4 to 10. For each group of fragments with the same length, sets of new structural alphabets with the number of prototypes in

the range from 10 to 100 were developed. The analysis of the obtained structural alphabets shown that the accurate prediction models of structural alphabets with prototypes of length 4 and size of 16 or larger could be useful for the prediction of backbone angle ψ. Developed prediction models for the constructed structural alphabets haven't achieved the accuracy which would outperform the existing predictors of protein structure properties. Based on the results, *mae* for predicted ψ angles decreases when increasing the SA size, so the goal could be achieved by applying the described clustering procedure for a larger number of prototypes or using a different approach for the development of SAs prediction models. Obtained structural alphabet with 16 prototypes of length 5 (SA 5_16) was compared with Protein Blocks. Several prototypes of SA 5_16 are very similar to PBs by *rmsda* (*rmsda* is less than 6.8° for 6 prototype pairs from SA 5_16 and Protein Blocks, and *rmsda* is less than 16.8° for 10 prototype pairs). The distribution of prototypes of SA 5_16 and Protein Blocks by the coarse secondary structure classification is very similar. Based on the similarity of SA 5_16 and PB, it can be concluded that the applied procedure for the construction of structural alphabets can be used as the base for the development of useful new structural alphabets.

Plans for future work include:

- development of ensemble of the obtained PBs prediction models with the highest precision for a particular Protein Block to increase the overall accuracy of Protein Blocks prediction;
- development of larger size structural alphabets based on TwoStep clustering algorithm and prediction models for their prototypes using different data mining approaches and classification algorithms.

7 Appendix

7.1 Size of clusters which corresponds to prototypes of structural alphabets

For each group of SAs with the same length of fragments, the sizes of clusters associated with their prototypes are shown.

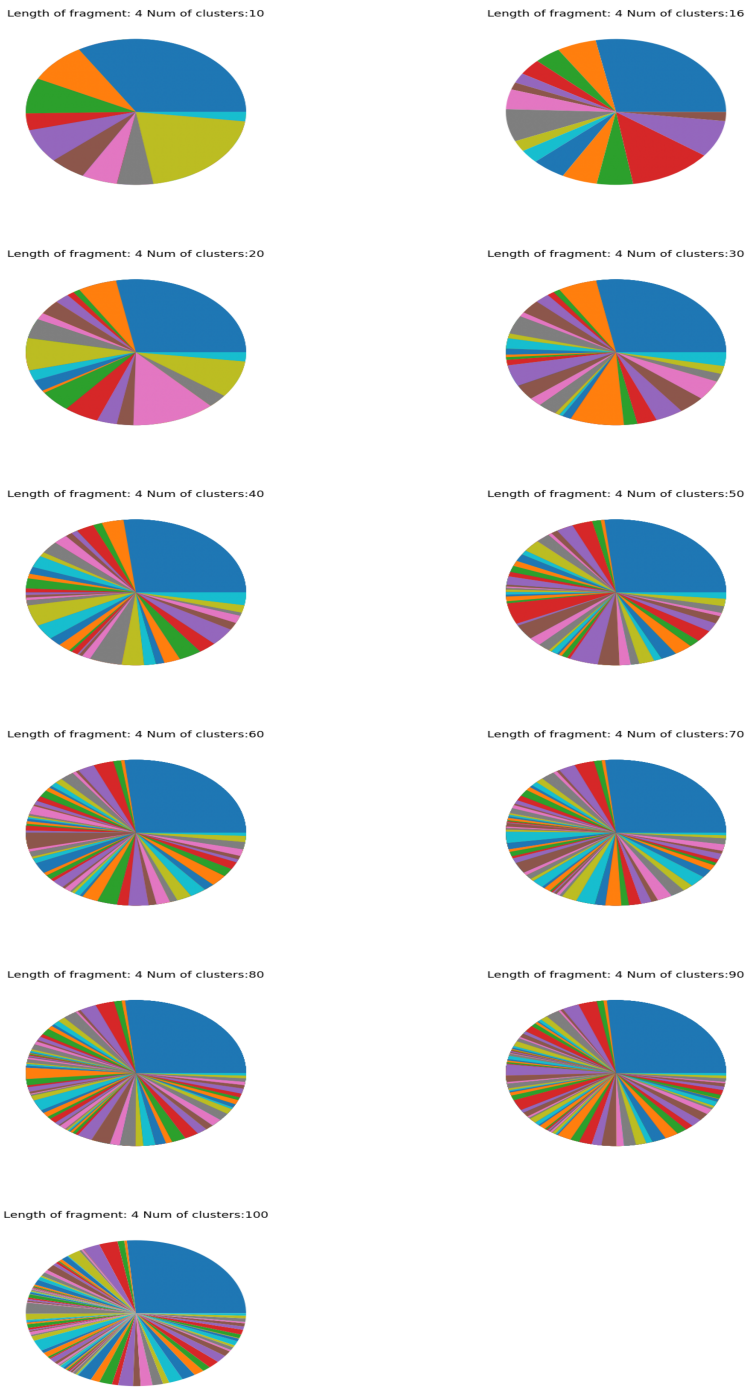

Figure 7.1: Size of clusters in structural alphabets constructed using fragments of length 4

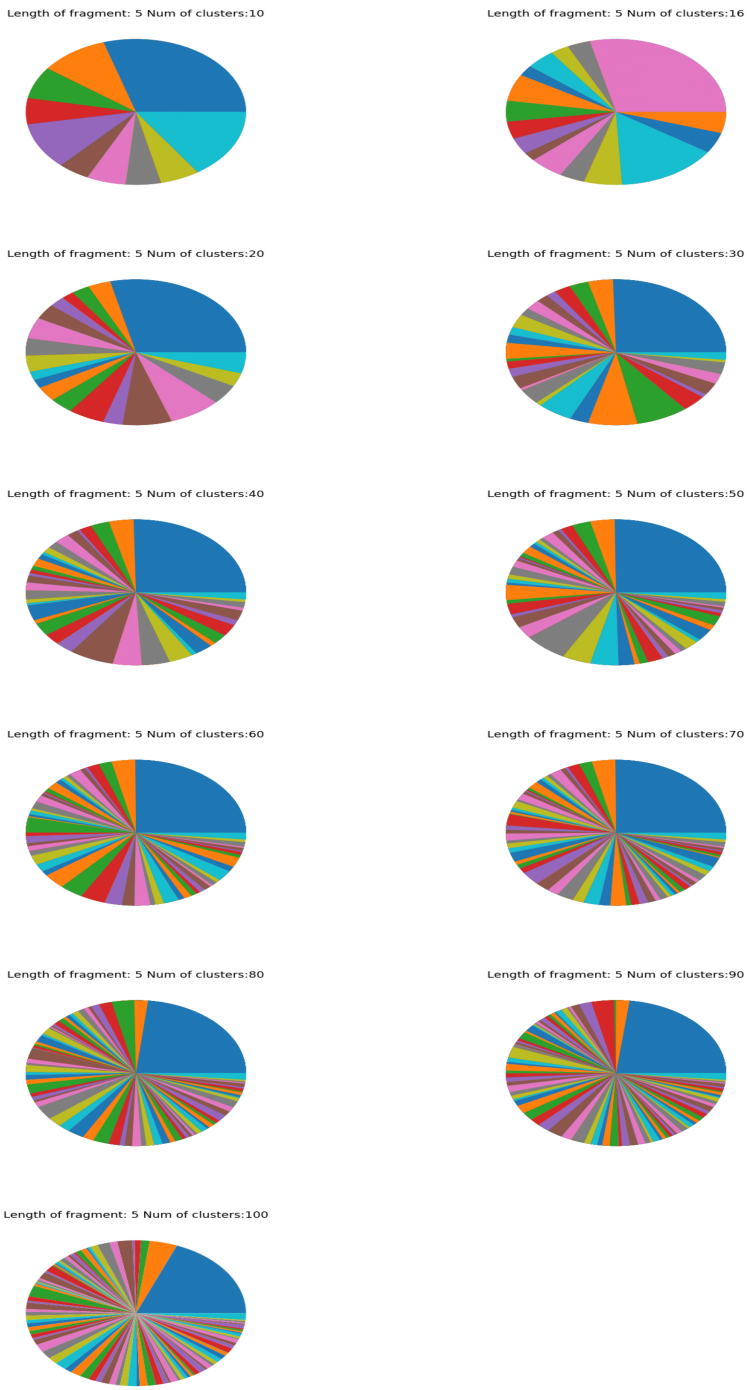

Figure 7.2: Size of clusters in structural alphabets constructed using fragments of length 5

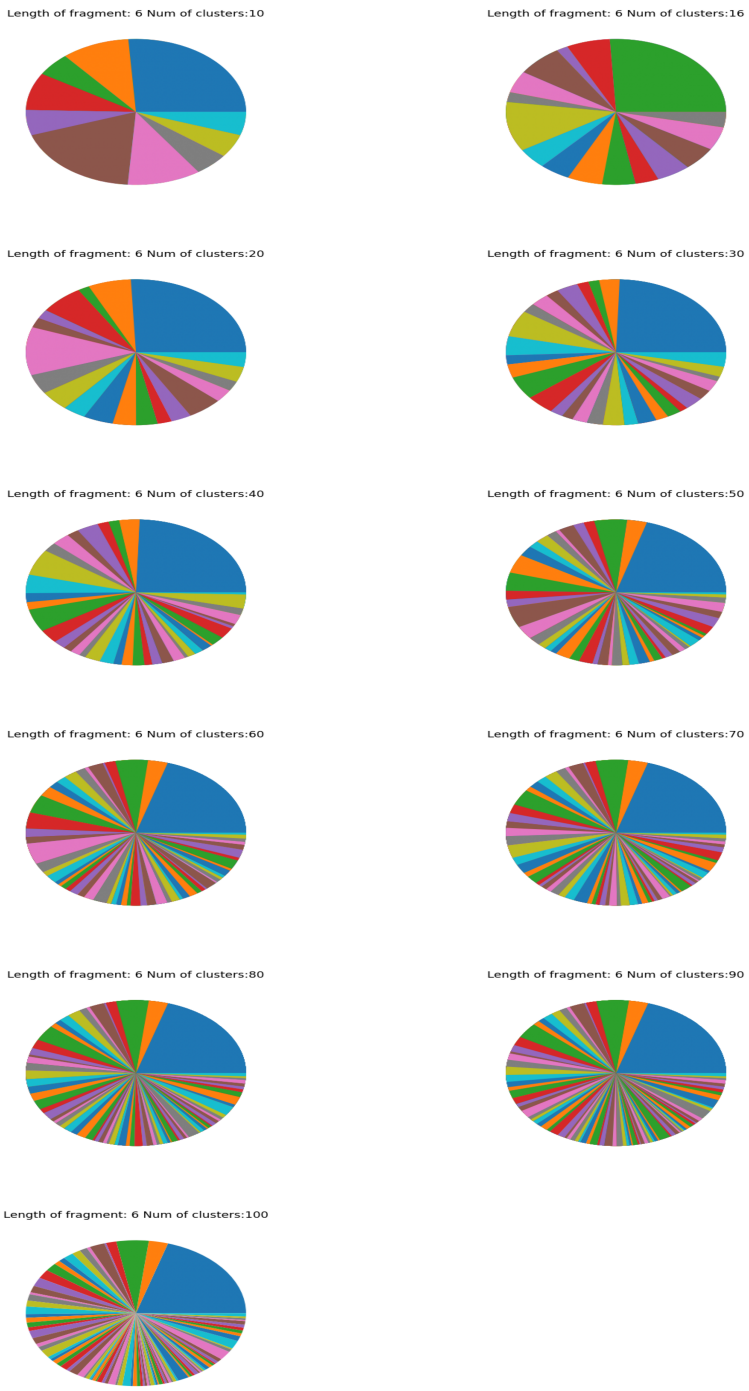

Figure 7.3: Size of clusters in structural alphabets constructed using fragments of length 6

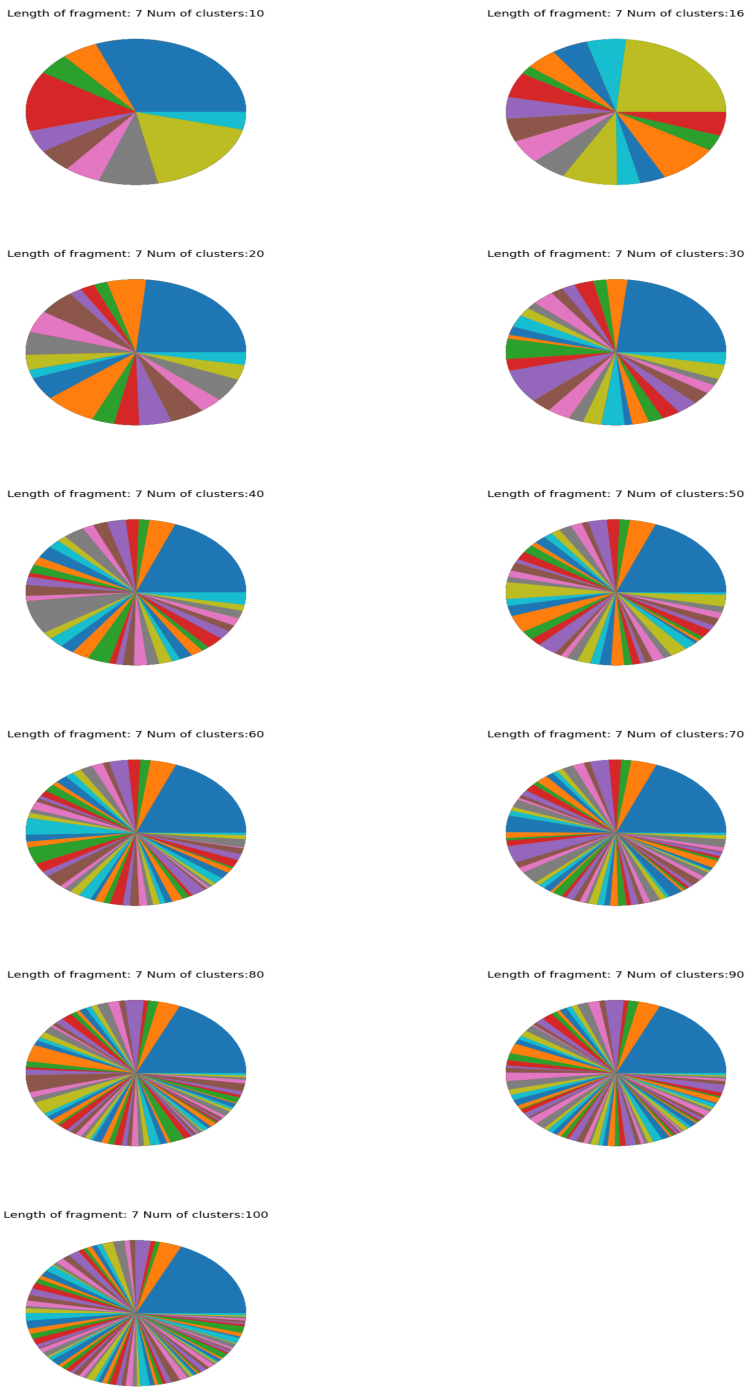

Figure 7.4: Size of clusters in structural alphabets constructed using fragments of length 7

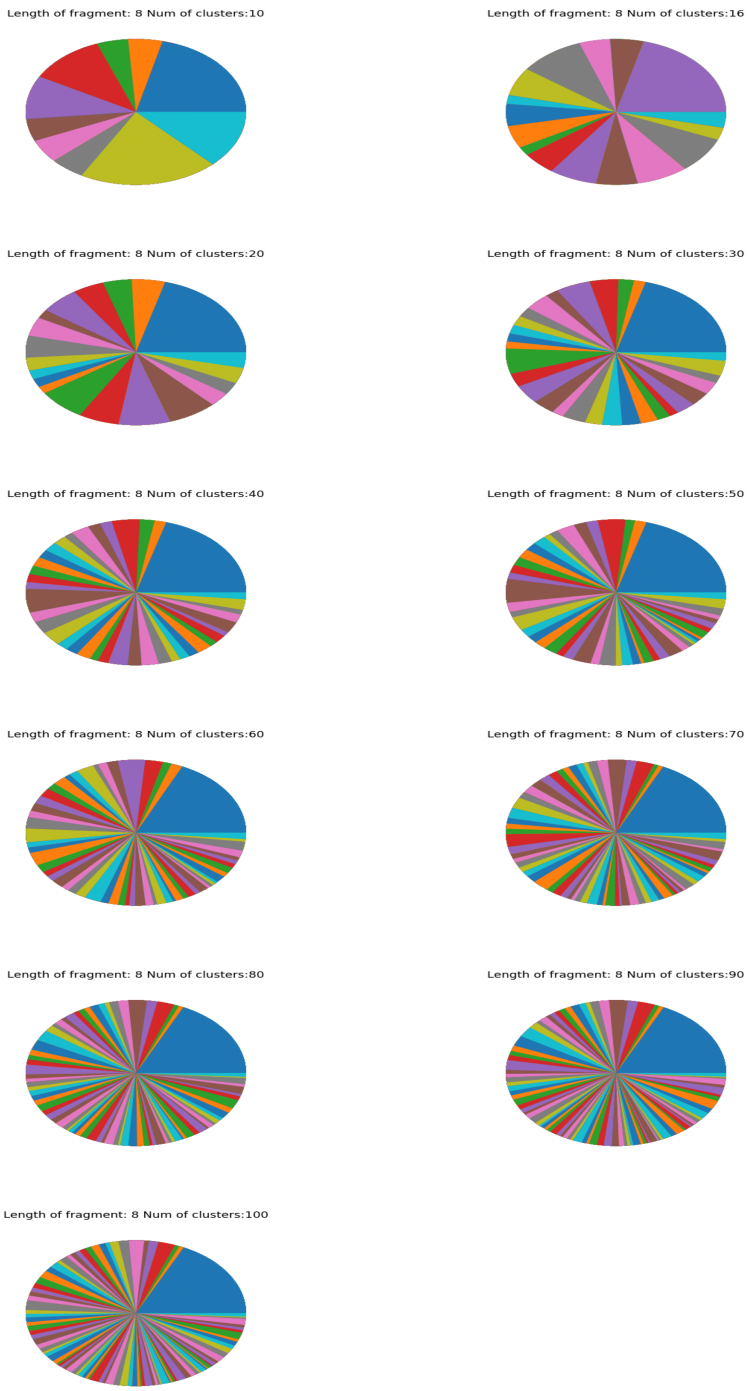

Figure 7.5: Size of clusters in structural alphabets constructed using fragments of length 8

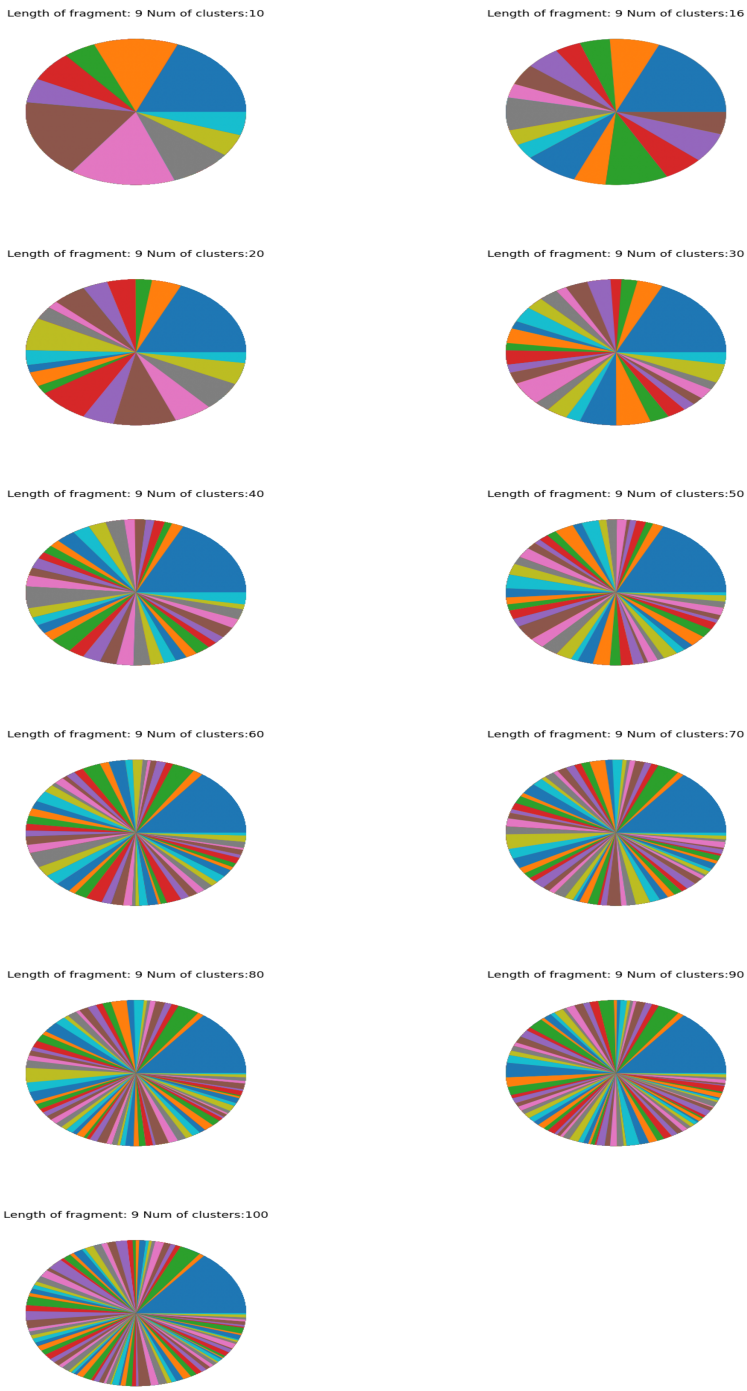

Figure 7.6: Size of clusters in structural alphabets constructed using fragments of length 9

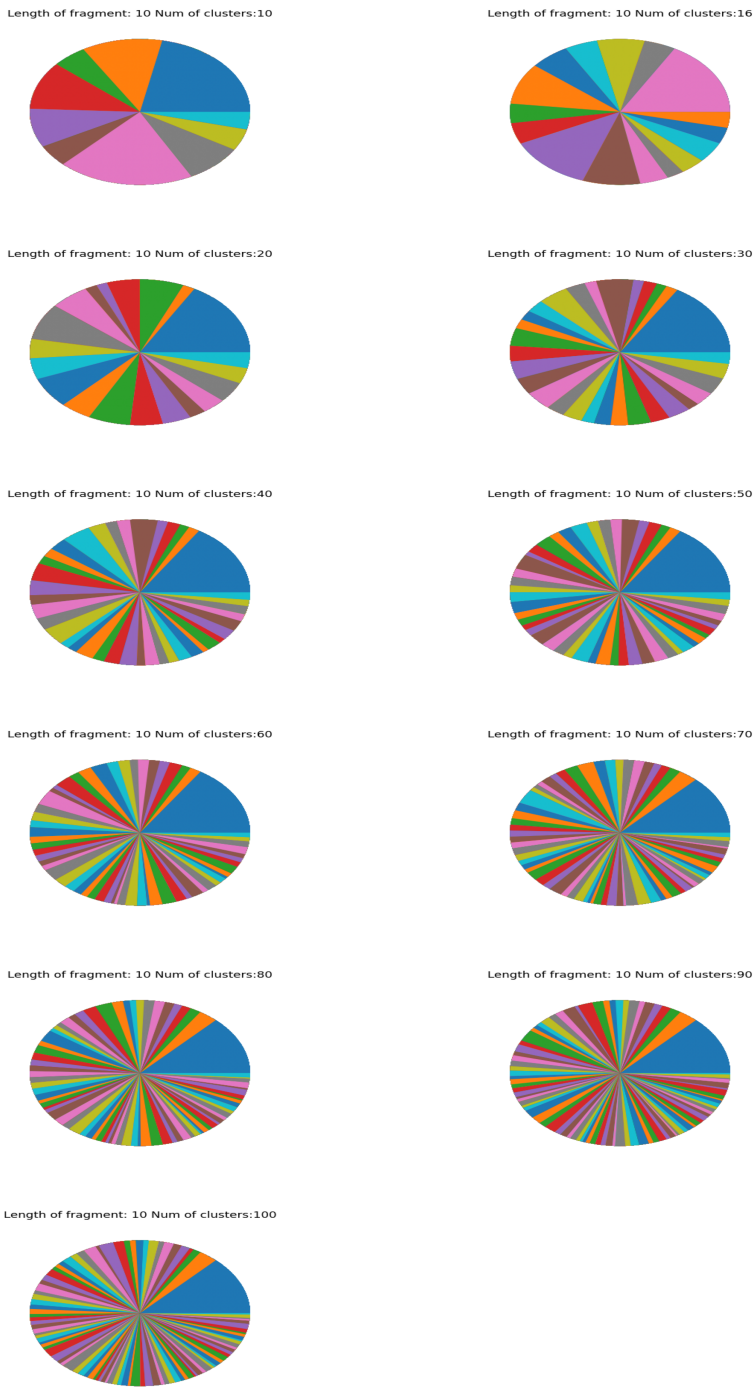

Figure 7.7: Size of clusters in structural alphabets constructed using fragments of length 10

7.2 Graphical presentations of SA prototypes

Graphical presentations of SA prototypes for each constructed SA using Ramachandran plot (R-plot) are shown in Figures from 7.9 to 7.21. ψ angle of the first amino acid and ϕ angle of the last amino acid in a prototype are shown in the first R-plot for each SA. Other R-plots of SA show ψ and ϕ angles of one amino acid in a prototype.

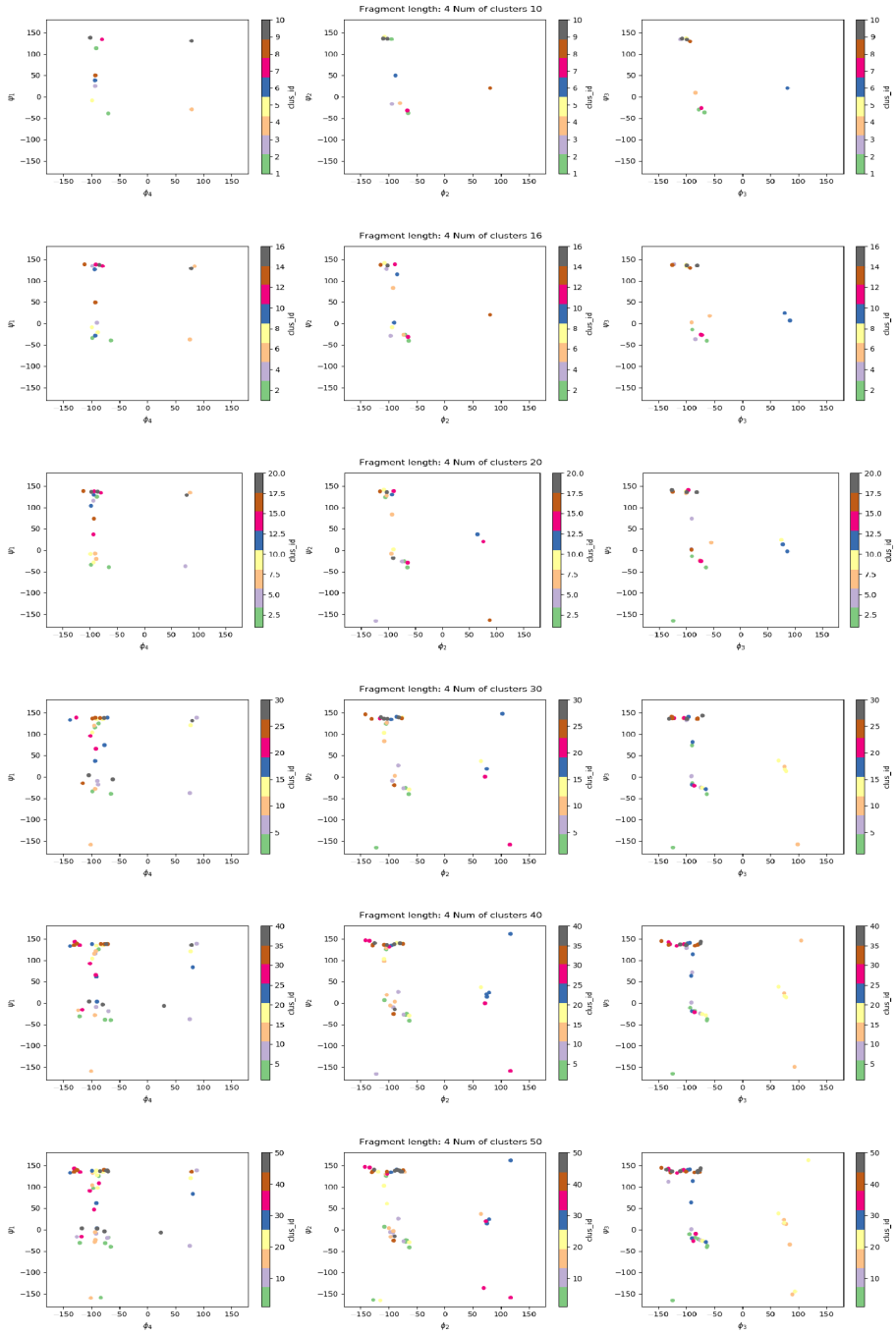

Figure 7.8: Dihedral angles of prototypes in structural alphabets of length 4 with number of prototypes from 10 to 50

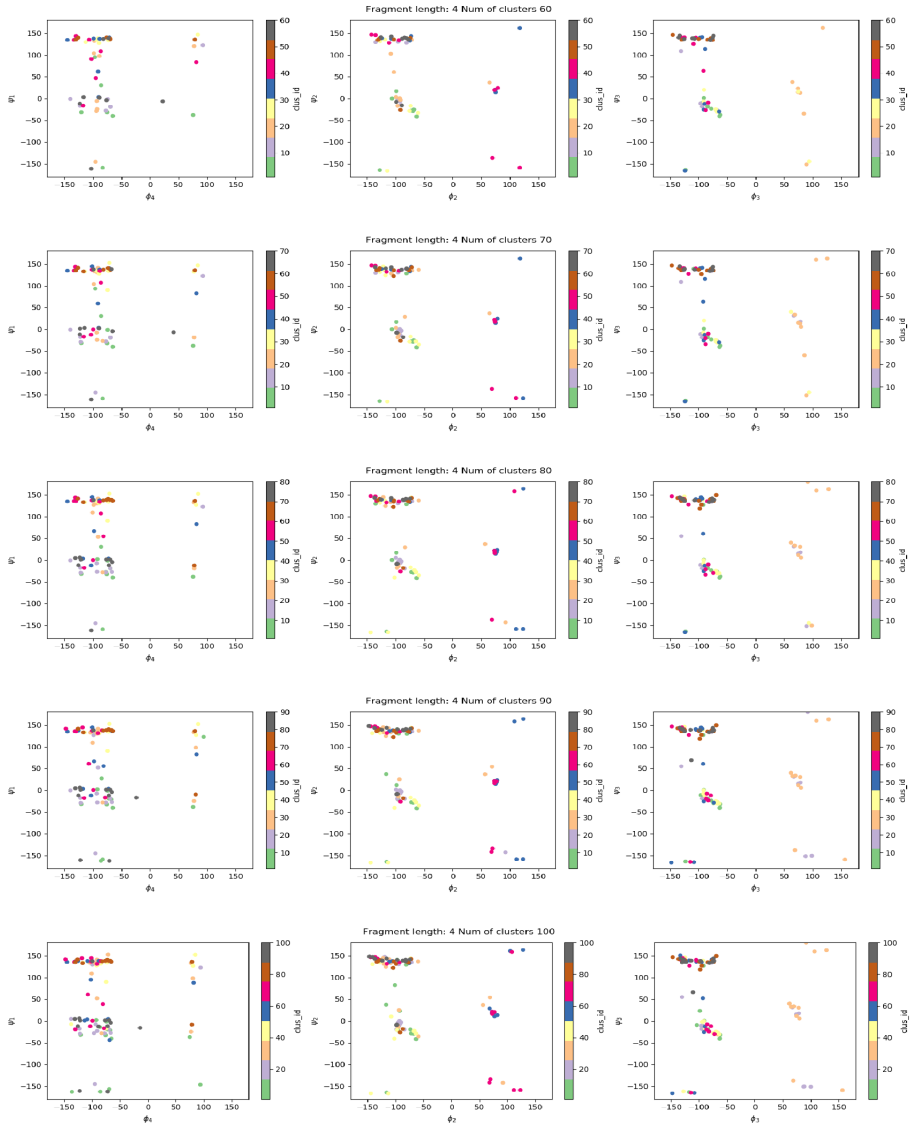

Figure 7.9: Dihedral angles of prototypes in structural alphabets of length 4 with number of prototypes from 60 to 100

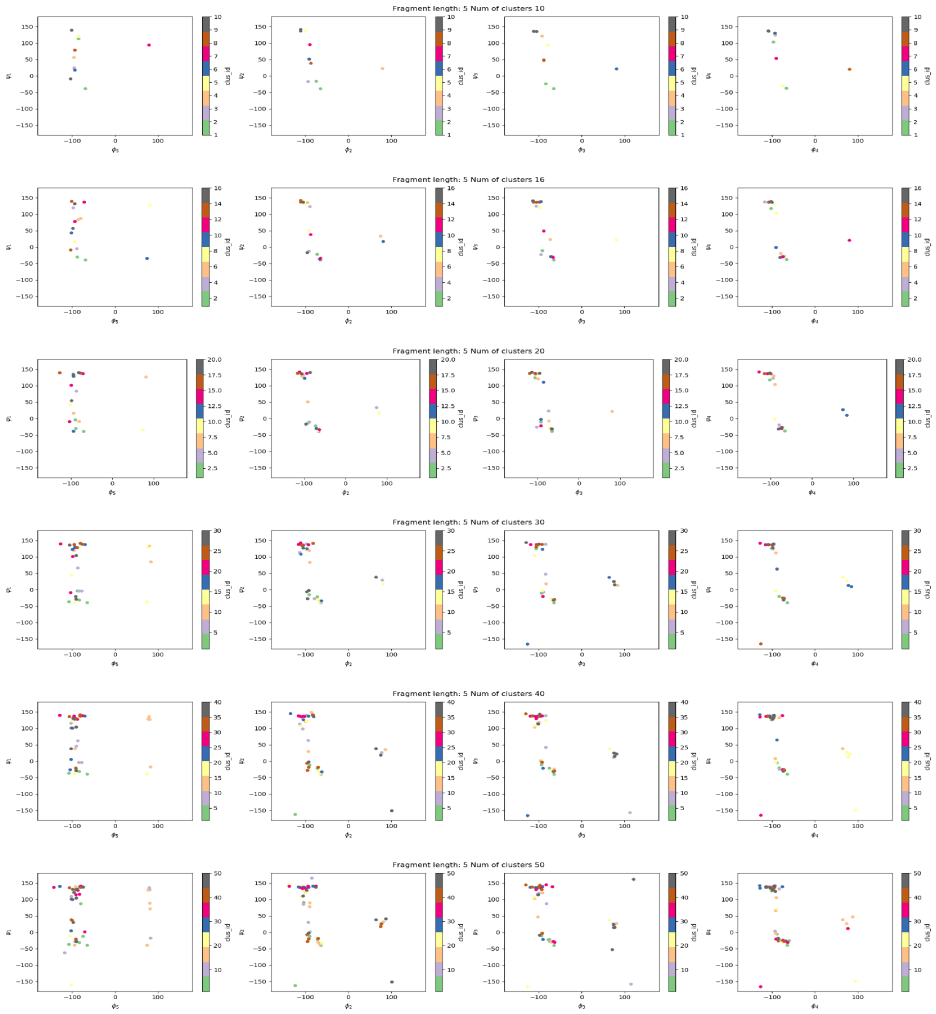

Figure 7.10: Dihedral angles of prototypes in structural alphabets of length 5 with number of prototypes from 10 to 50

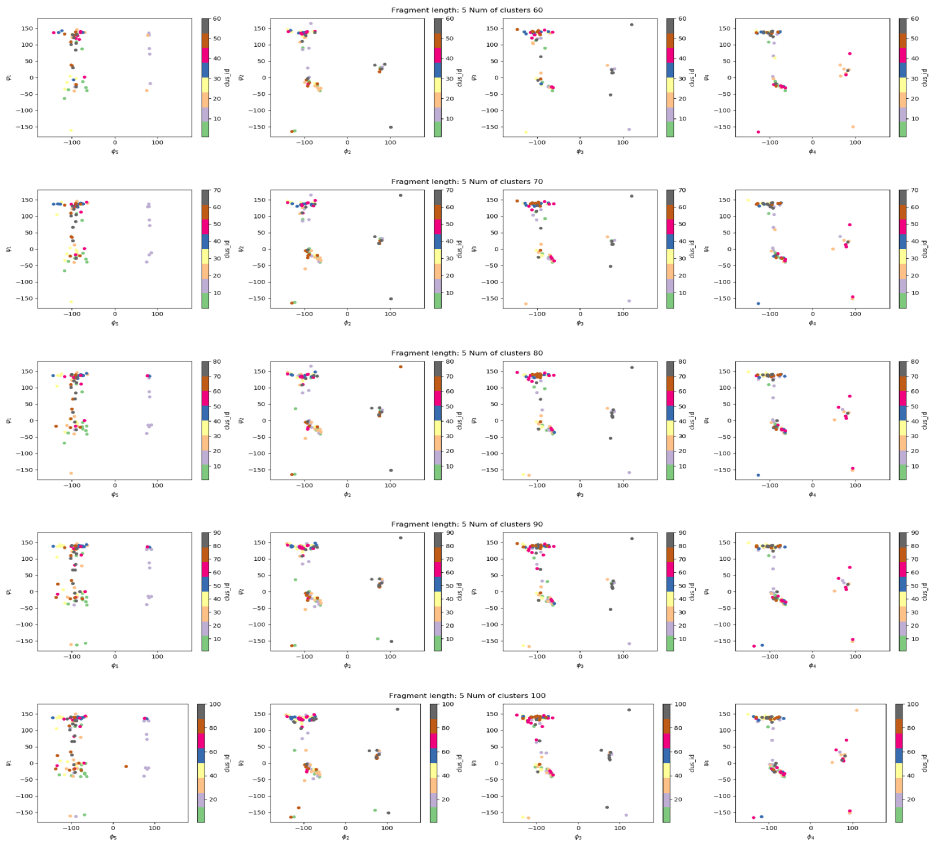

Figure 7.11: Dihedral angles of prototypes in structural alphabets of length 5 with number of prototypes from 60 to 100

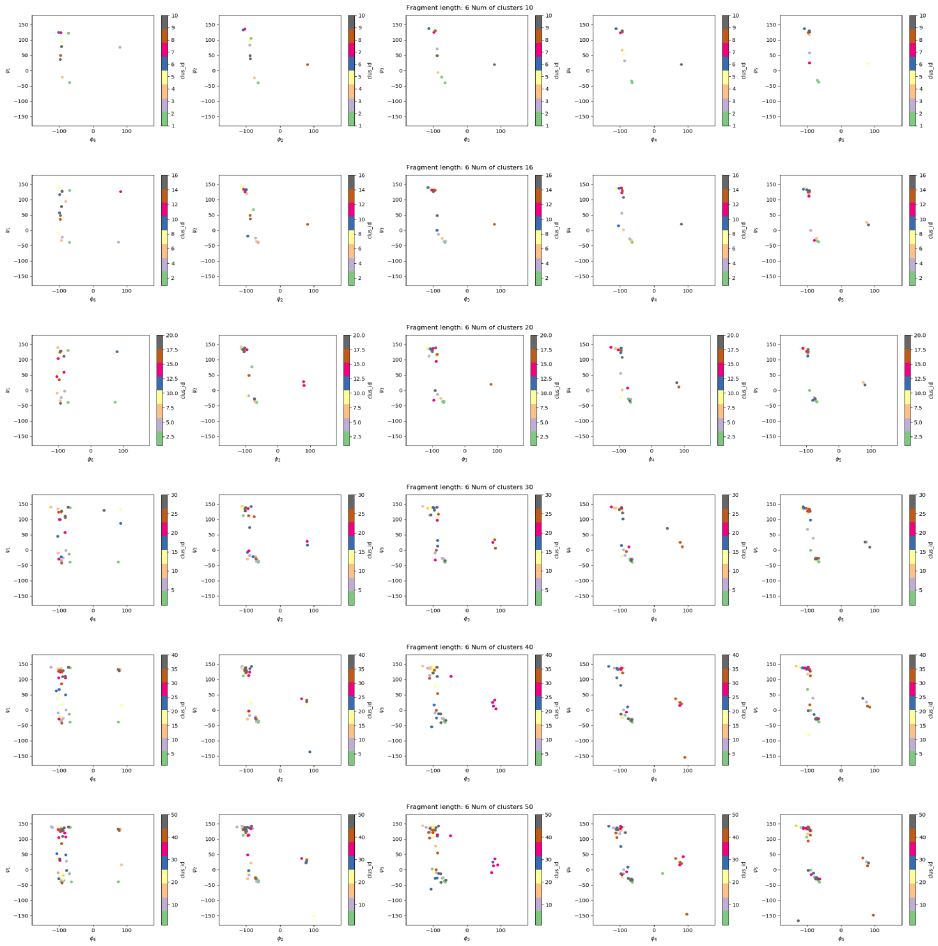

Figure 7.12: Dihedral angles of prototypes in structural alphabets of length 6 with number of prototypes from 10 to 50

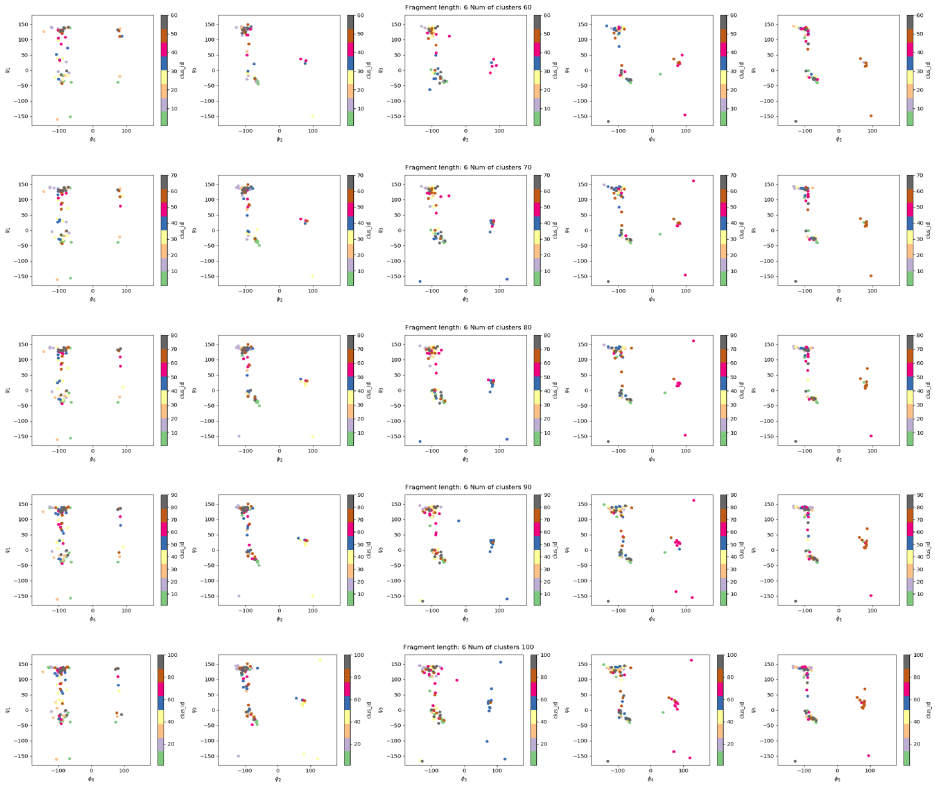

Figure 7.13: Dihedral angles of prototypes in structural alphabets of length 6 with number of prototypes from 60 to 100

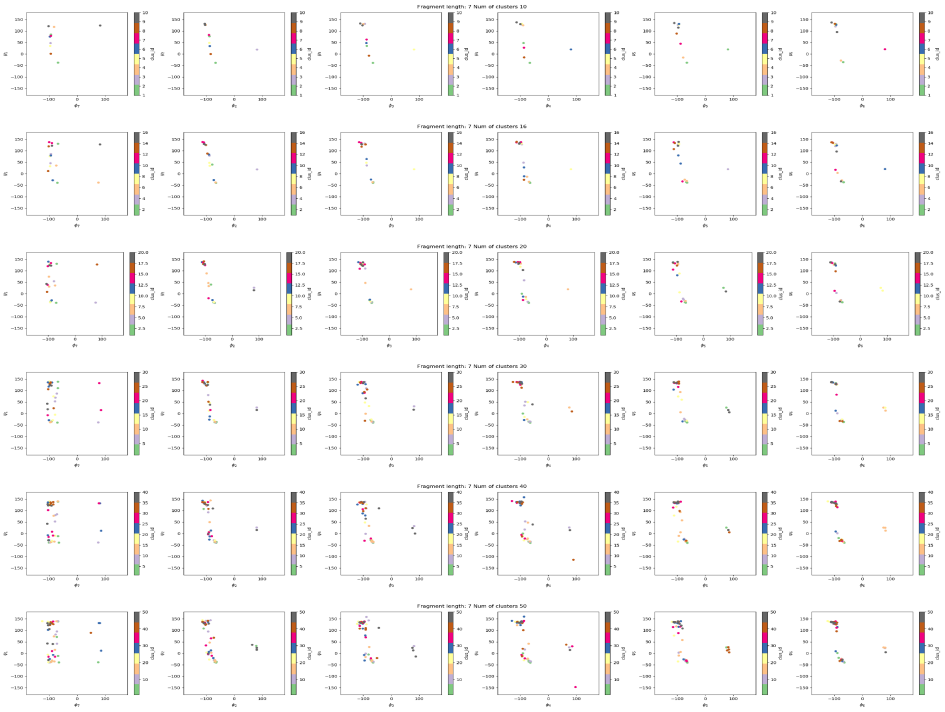

Figure 7.14: Dihedral angles of prototypes in structural alphabets of length 7 with number of prototypes from 10 to 50

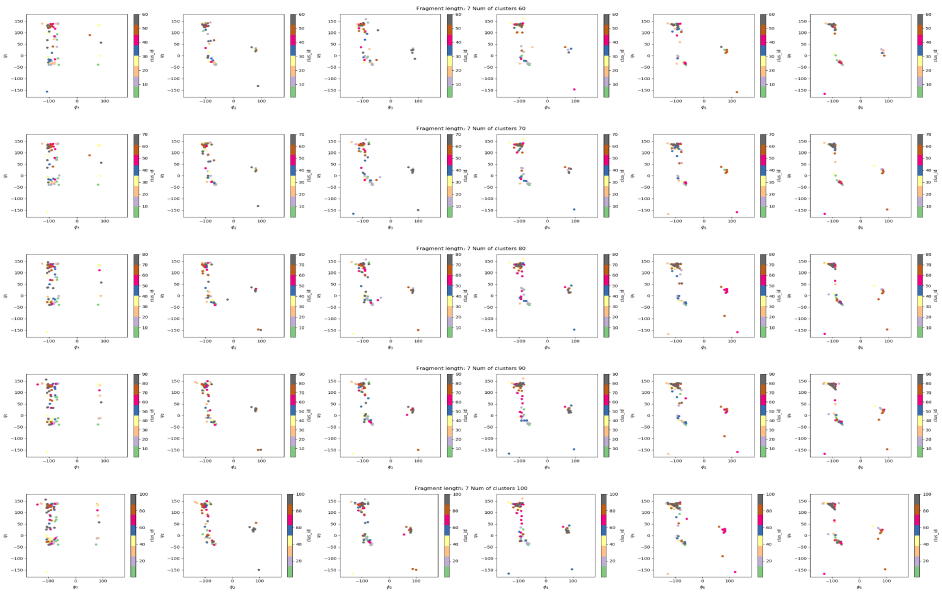

Figure 7.15: Dihedral angles of prototypes in structural alphabets of length 7 with number of prototypes from 60 to 100

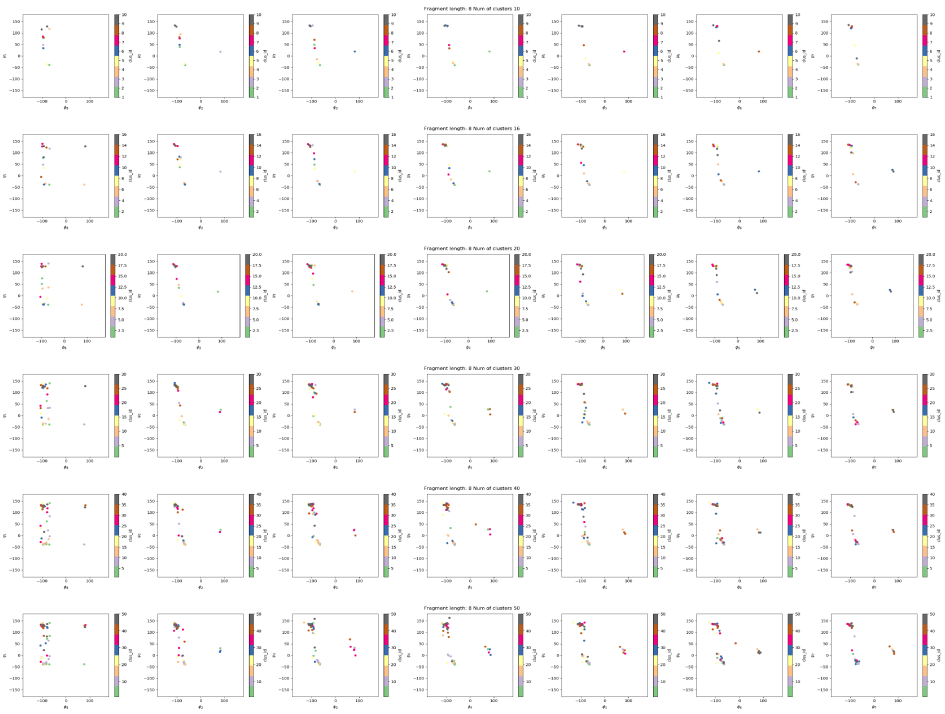

Figure 7.16: Dihedral angles of prototypes in structural alphabets of length 8 with number of prototypes from 10 to 50

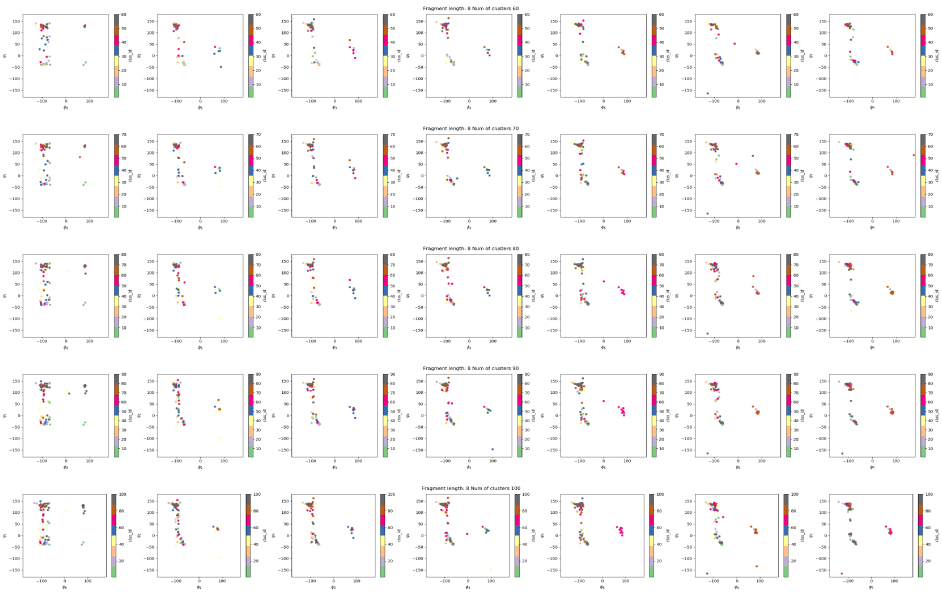

Figure 7.17: Dihedral angles of prototypes in structural alphabets of length 8 with number of prototypes from 60 to 100

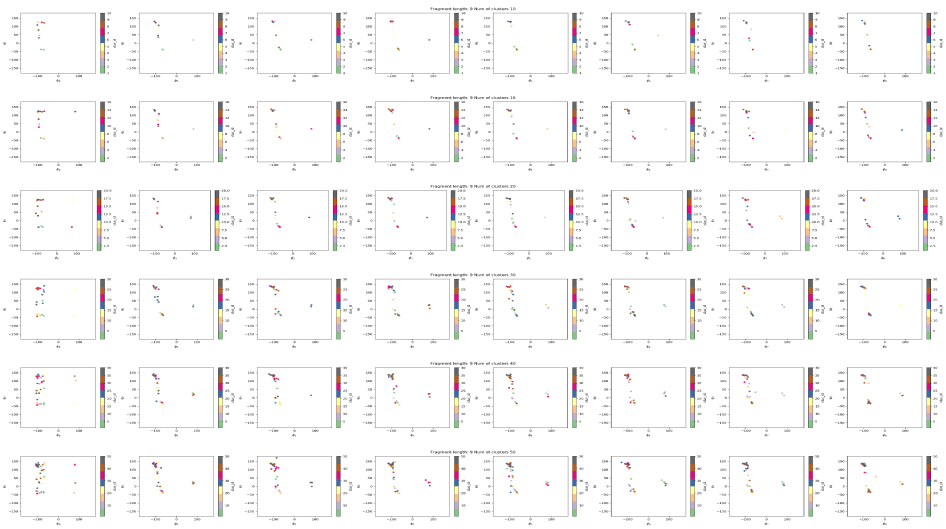

Figure 7.18: Dihedral angles of prototypes in structural alphabets of length 9 with number of prototypes from 10 to 50

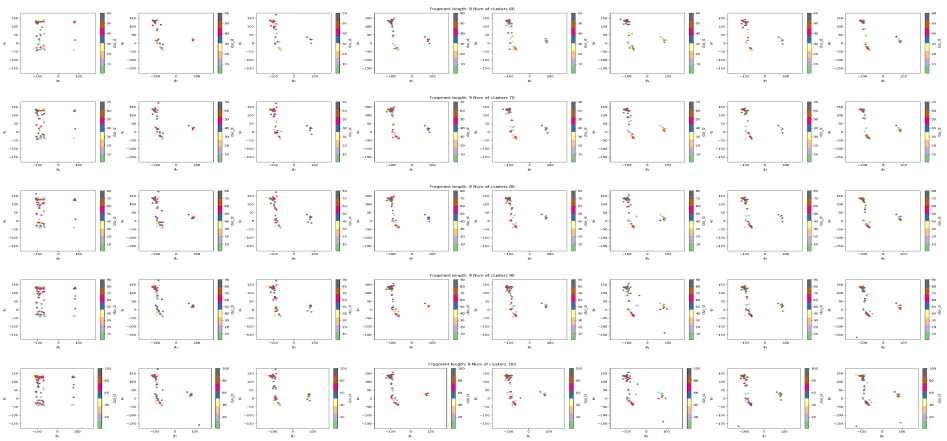

Figure 7.19: Dihedral angles of prototypes in structural alphabets of length 9 with number of prototypes from 60 to 100

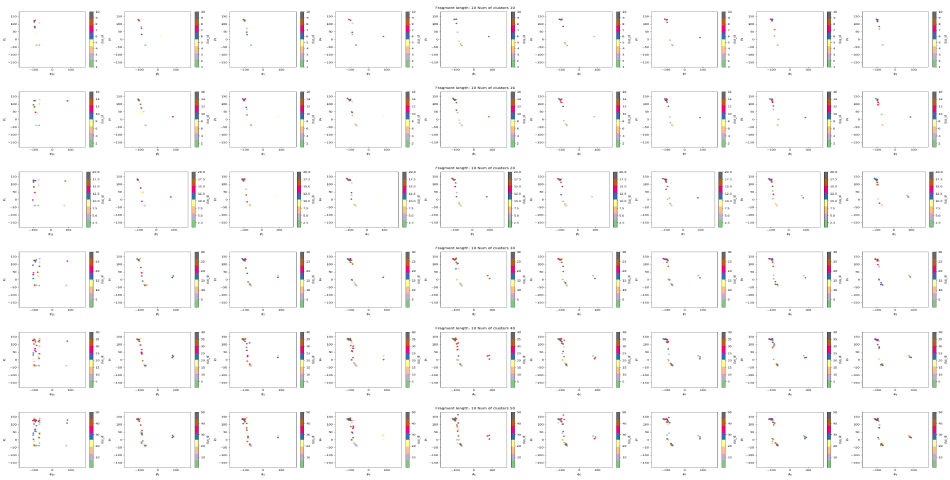

Figure 7.20: Dihedral angles of prototypes in structural alphabets of length 10 with number of prototypes from 10 to 50

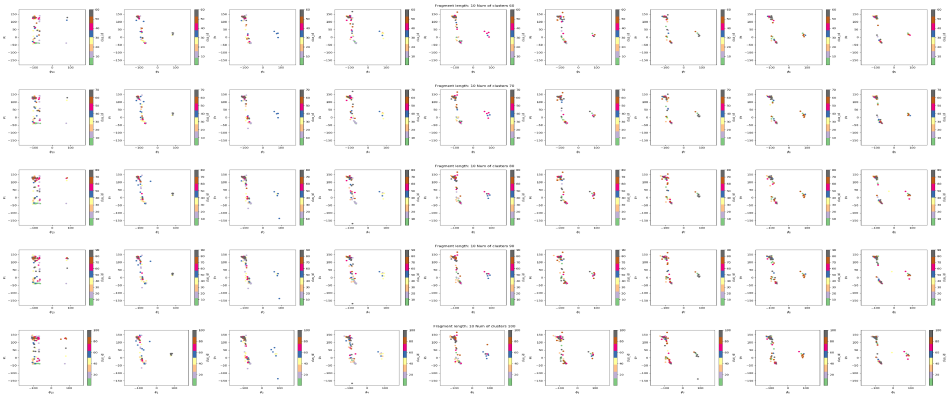

Figure 7.21: Dihedral angles of prototypes in structural alphabets of length 10 with number of prototypes from 60 to 100

7.3 Frequency of DSSP states in SAs with 20 prototypes

For each prototype of SAs with 20 prototypes, the frequencies of DSSP states per amino acid position in fragments covered by the prototype are shown in Figures from 7.22 to 7.35.

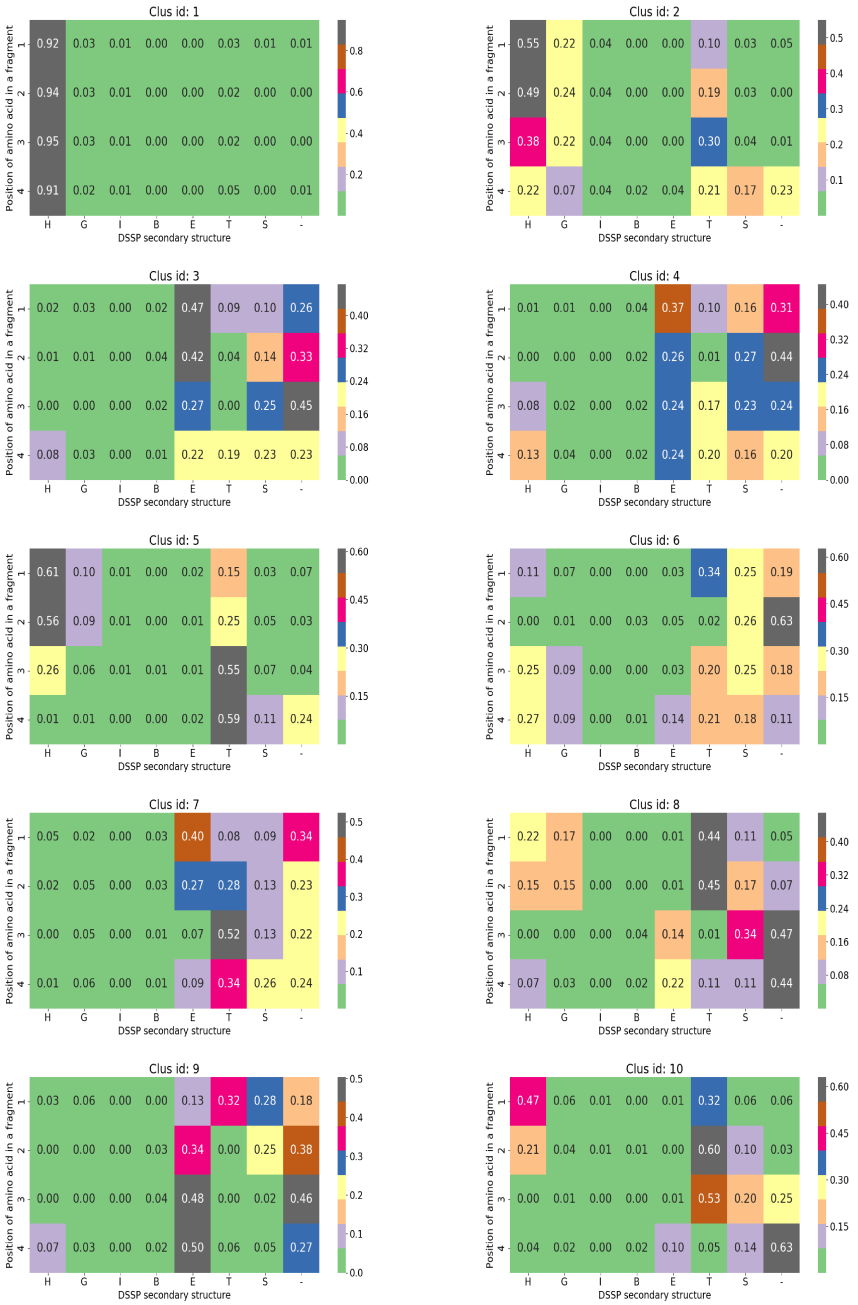

Figure 7.22: Frequency of DSSP states per amino acid position in clusters of SA **4_20** for clusters with id from 1 to 10

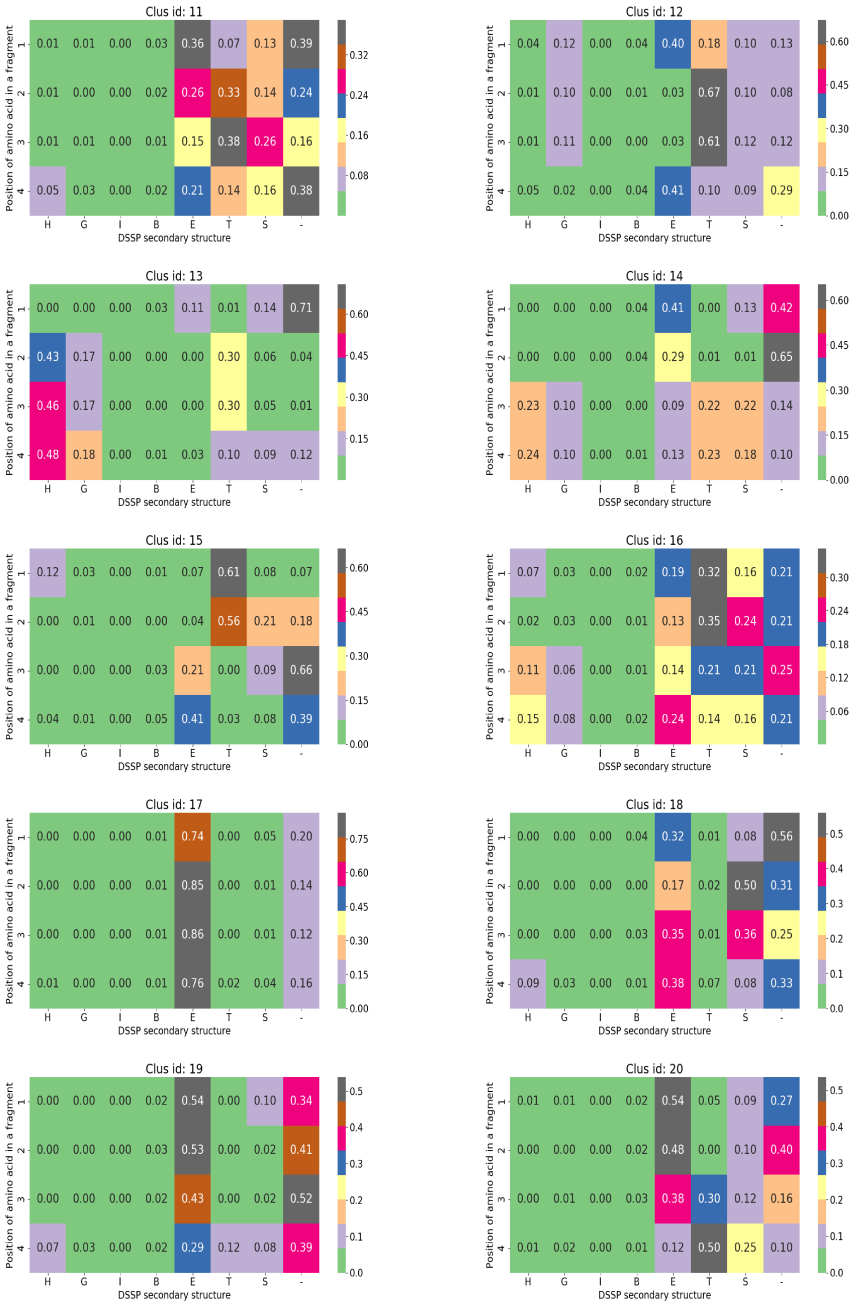

Figure 7.23: Frequency of DSSP states per amino acid position in clusters of SA **4_20** for clusters with id from 11 to 20

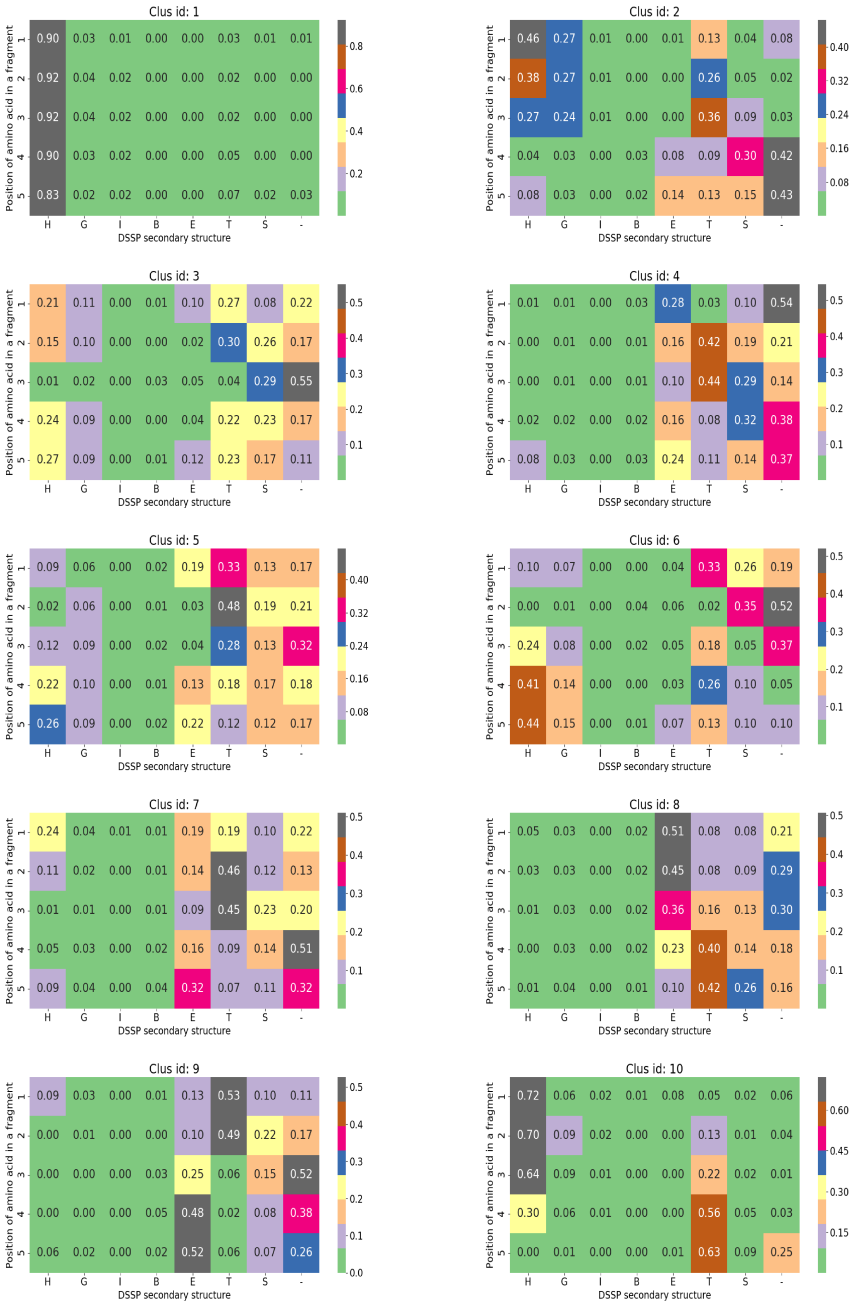

Figure 7.24: Frequency of DSSP states per amino acid position in clusters of SA **5_20** for clusters with id from 1 to 10

Figure 7.25: Frequency of DSSP states per amino acid position in clusters of SA **5_20** for clusters with id from 11 to 20

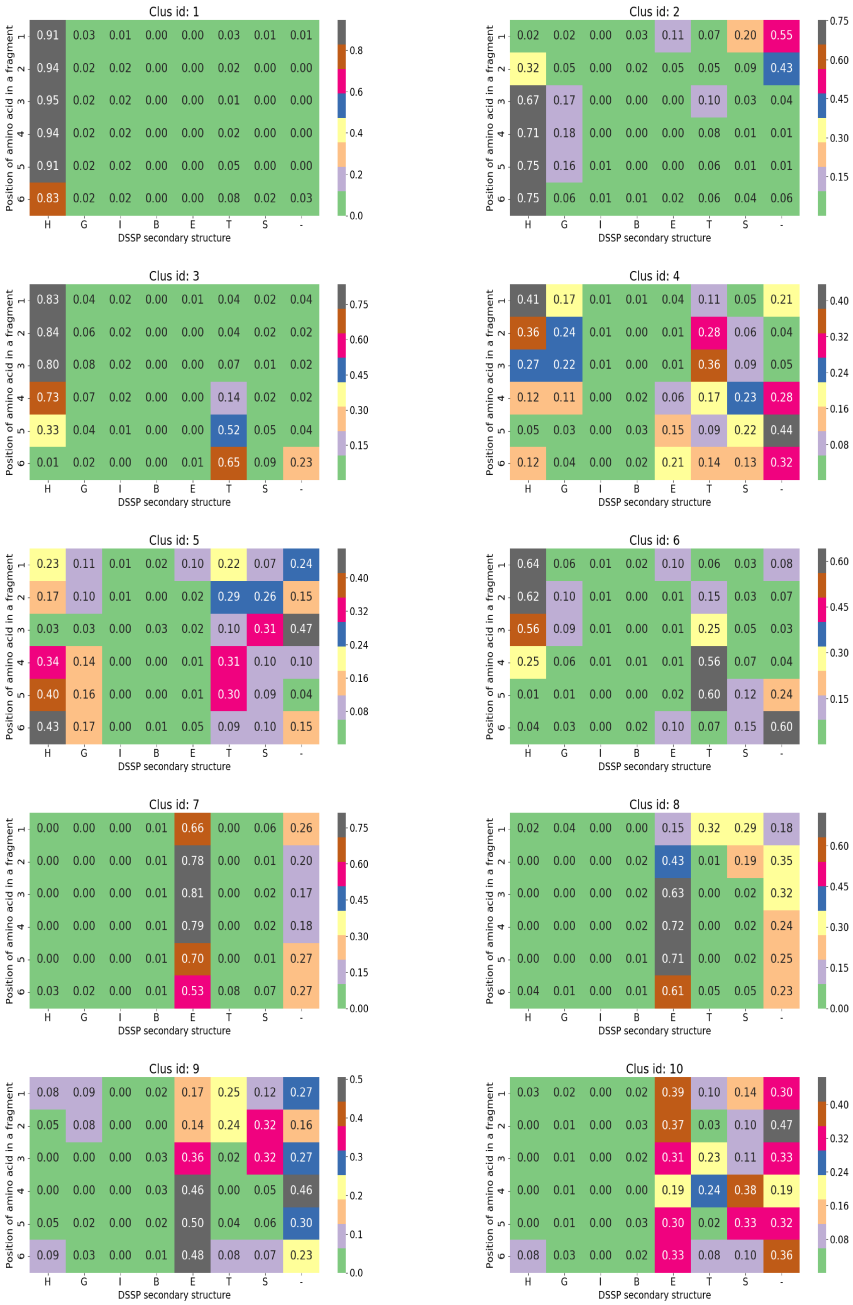

Figure 7.26: Frequency of DSSP states per amino acid position in clusters of SA **6_20** for clusters with id from 1 to 10

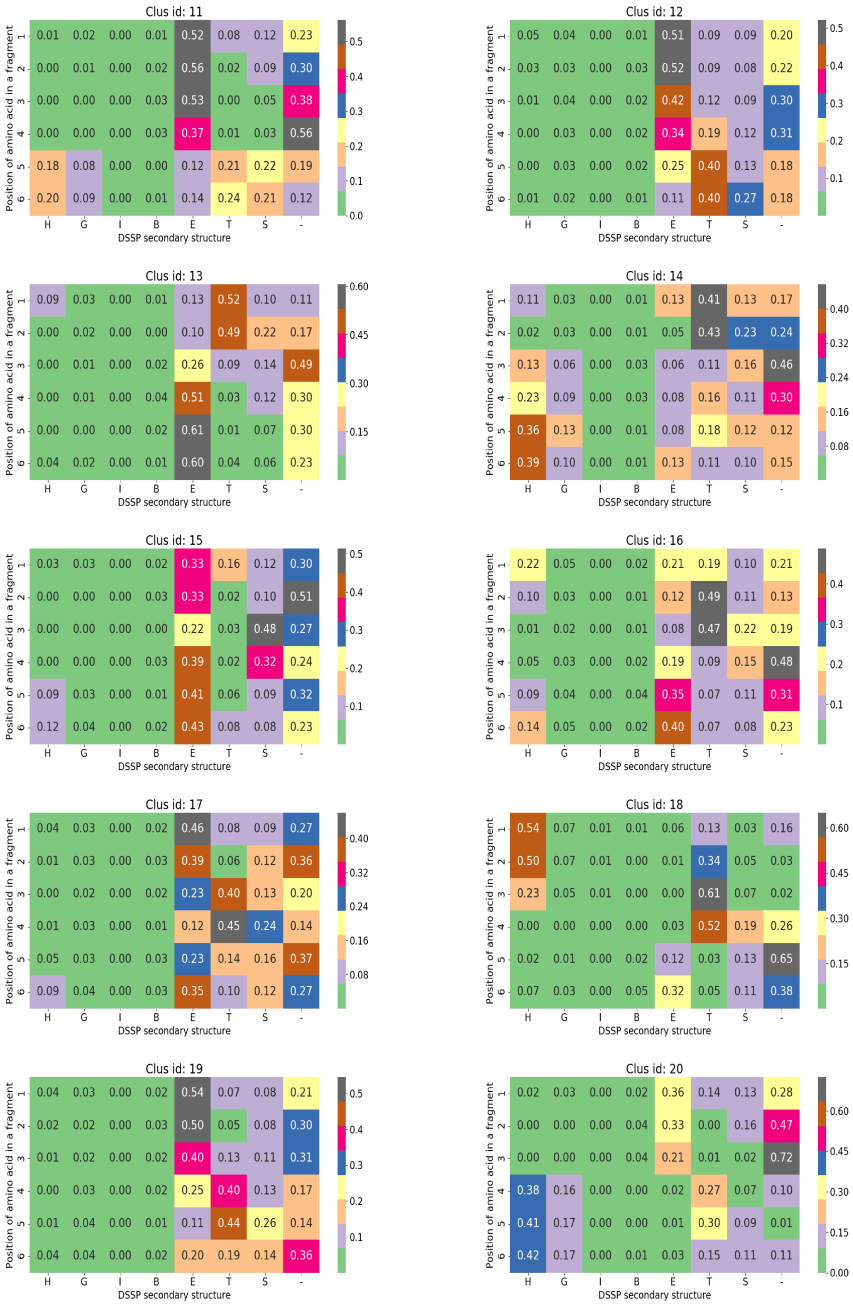

Figure 7.27: Frequency of DSSP states per amino acid position in clusters of SA **6_20** for clusters with id from 11 to 20

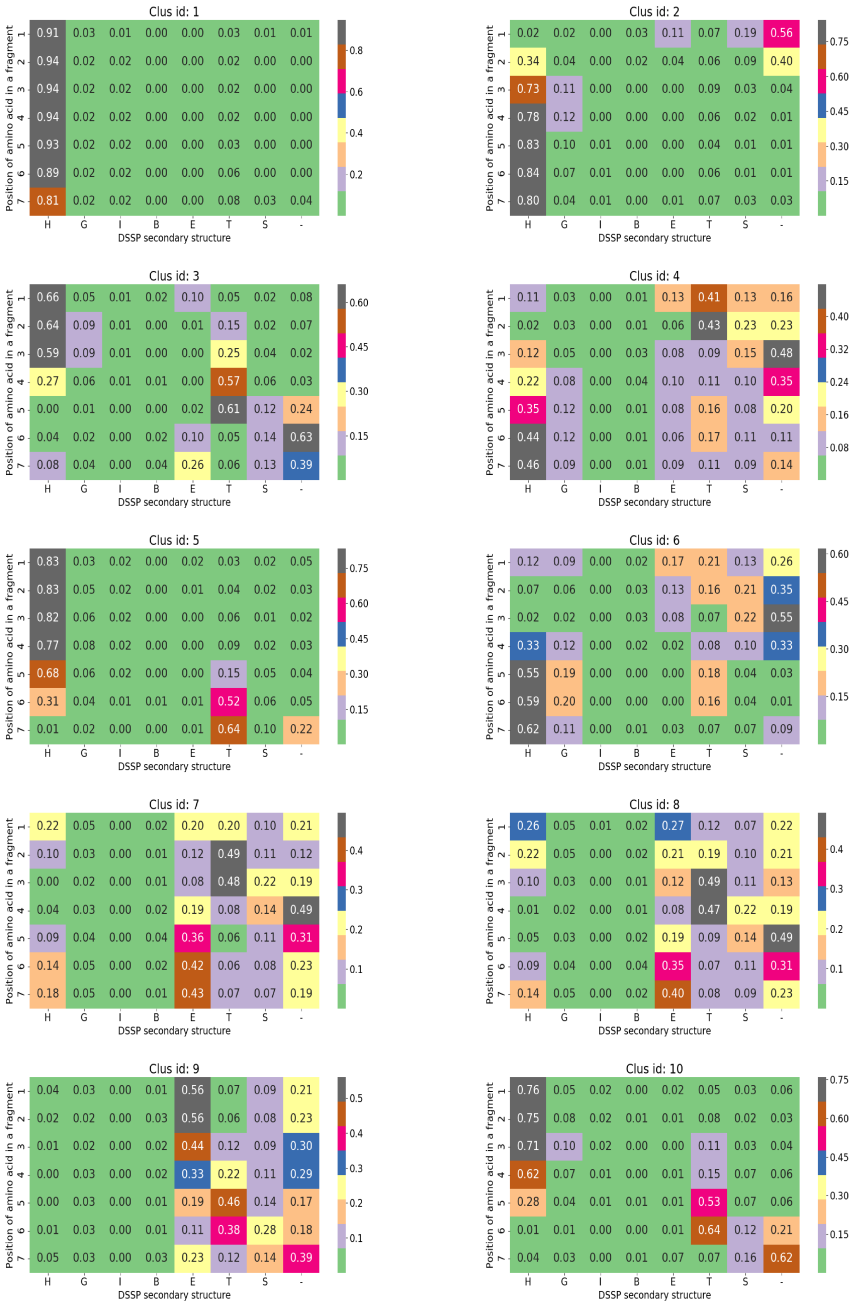

Figure 7.28: Frequency of DSSP states per amino acid position in clusters of SA **7_20** for clusters with id from 1 to 10

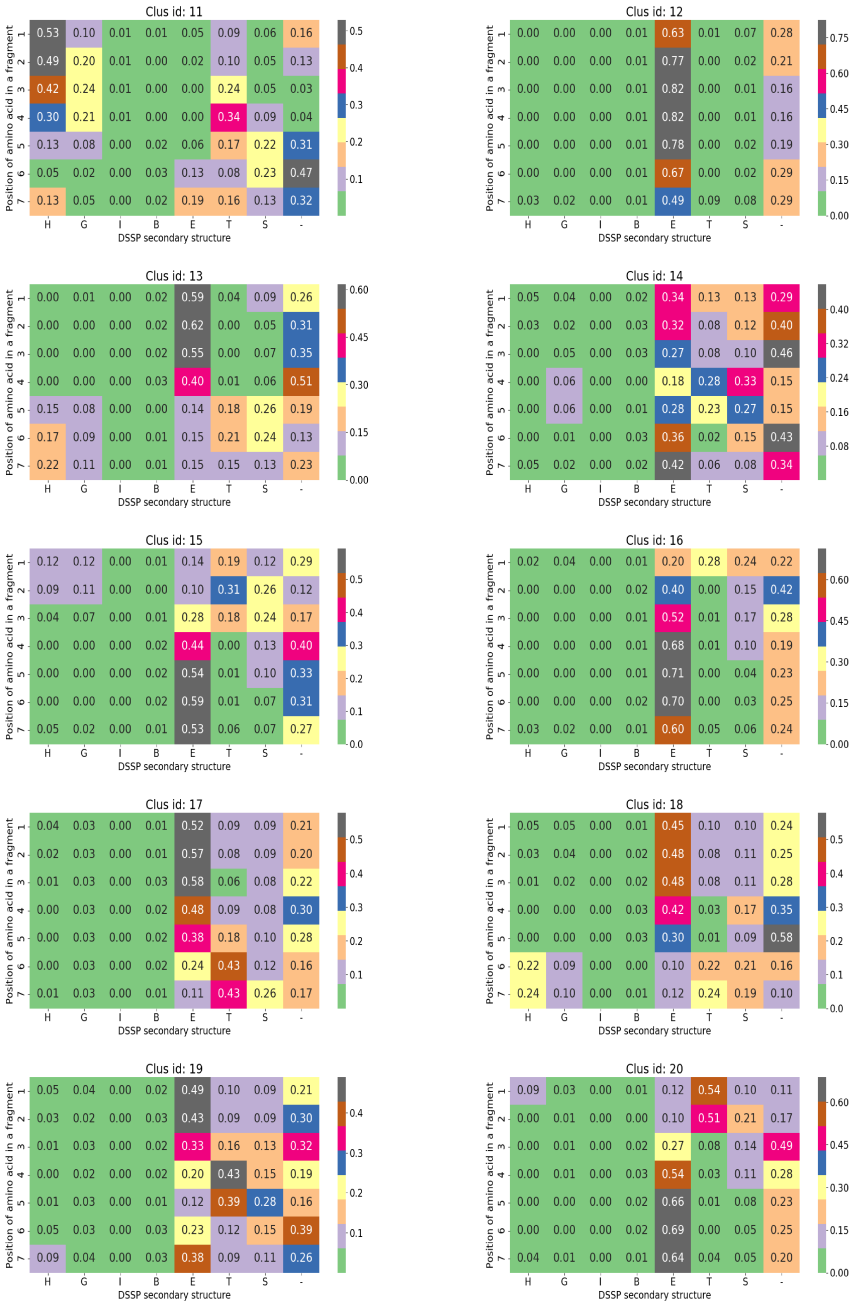

Figure 7.29: Frequency of DSSP states per amino acid position in clusters of SA **7_20** for clusters with id from 11 to 20

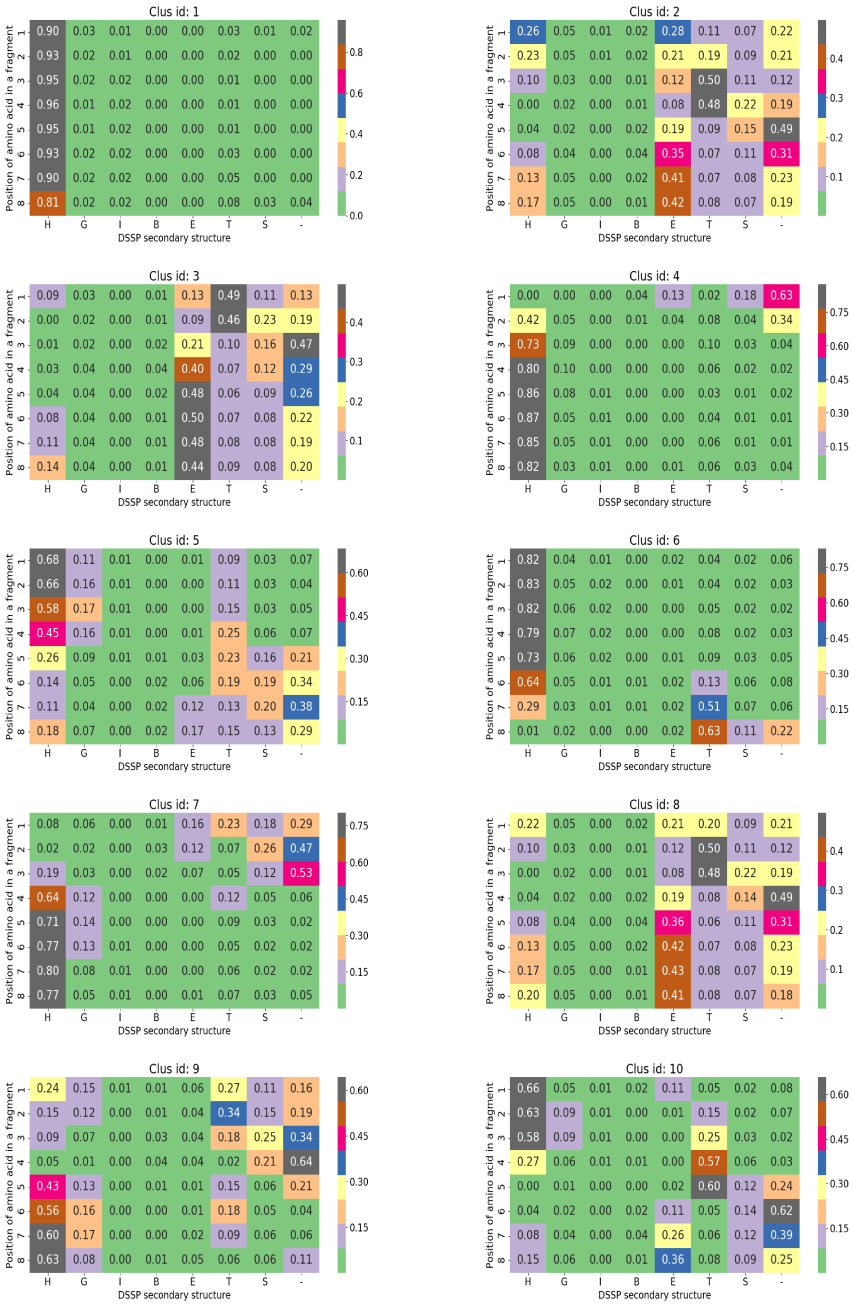

Figure 7.30: Frequency of DSSP states per amino acid position in clusters of SA **8_20** for clusters with id from 1 to 10

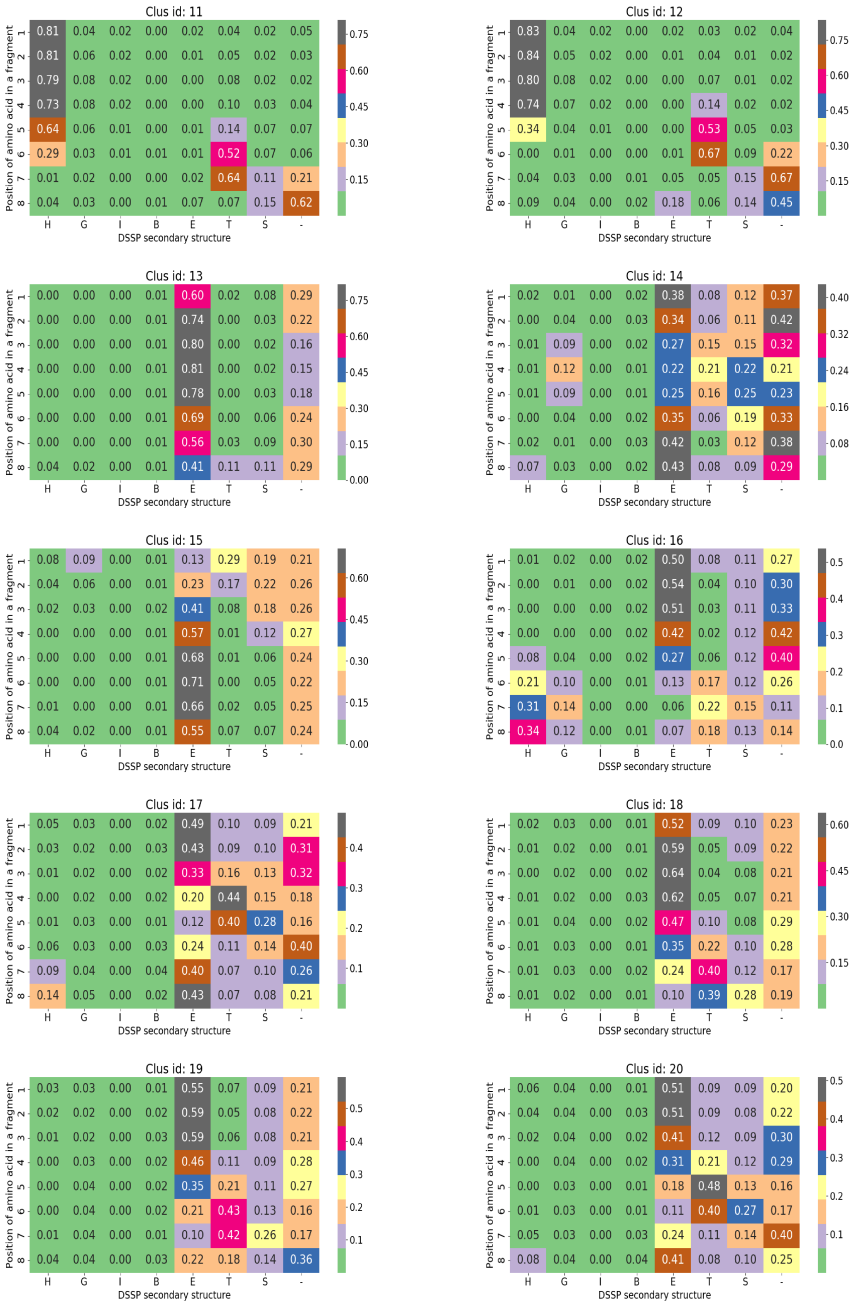

Figure 7.31: Frequency of DSSP states per amino acid position in clusters of SA **8_20** for clusters with id from 11 to 20

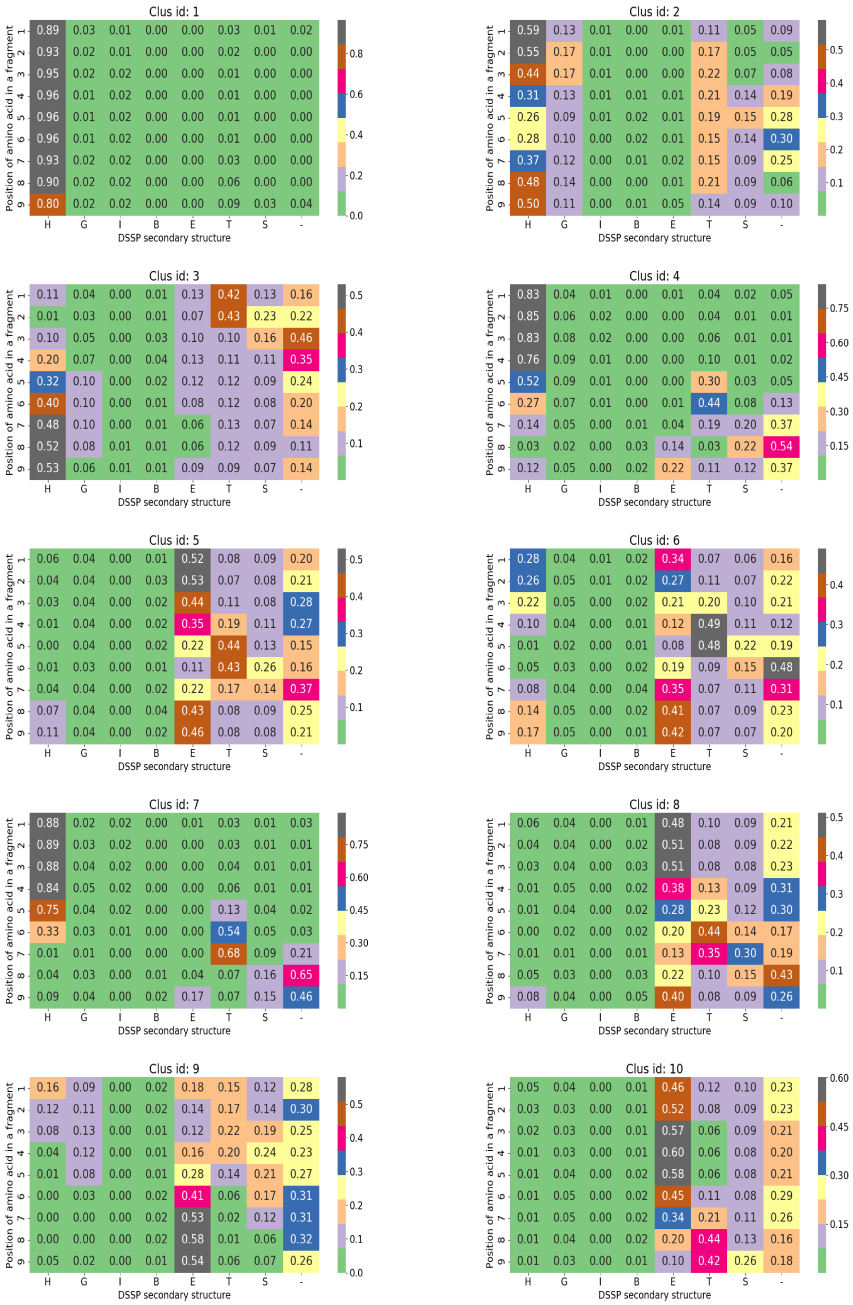

Figure 7.32: Frequency of DSSP states per amino acid position in clusters of SA **9_20** for clusters with id from 1 to 10

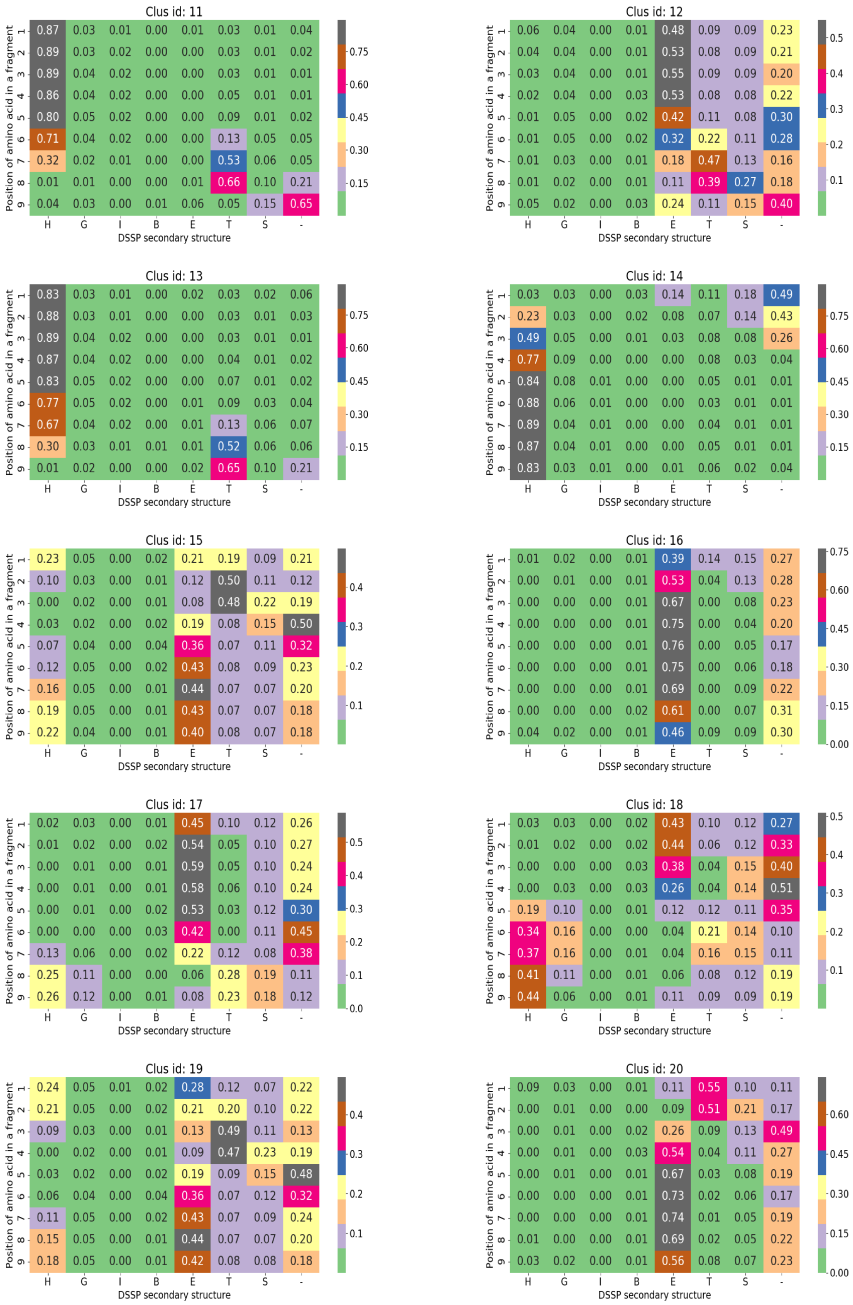

Figure 7.33: Frequency of DSSP states per amino acid position in clusters of SA **9_20** for clusters with id from 11 to 20

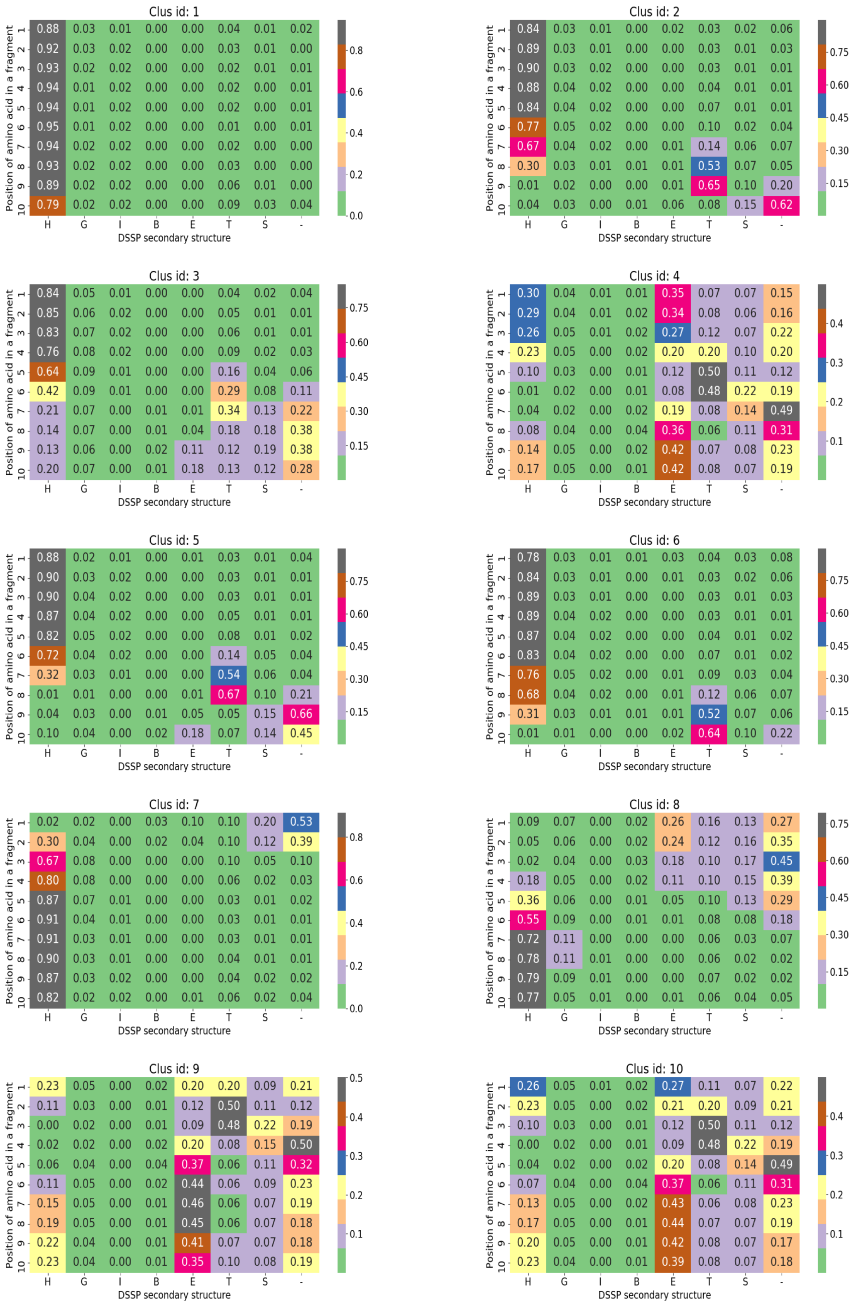

Figure 7.34: Frequency of DSSP states per amino acid position in clusters of SA **10_20** for clusters with id from 1 to 10

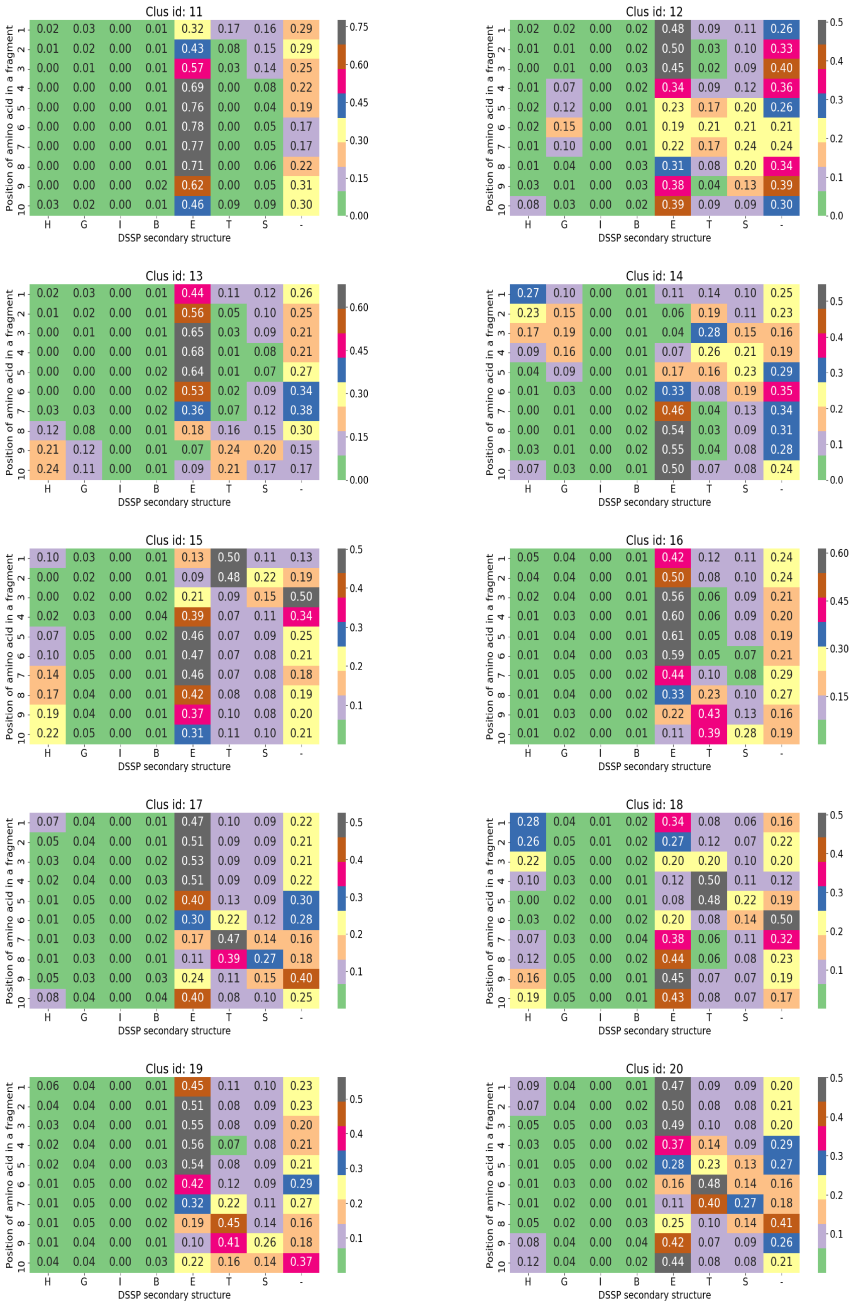

Figure 7.35: Frequency of DSSP states per amino acid position in clusters of SA **10_20** for clusters with id from 11 to 20

Bibliography

[1] UniProtKB/Swiss-Prot protein knowledgebase release 2021_01 statistics. `https://web.expasy.org/docs/relnotes/relstat.html`. Accessed: 9 March 2021.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] C. C. Aggarwal. *Data Mining*. Springer International Publishing, 2015.

[4] B. Alberts, A. Johnson, and J. Lewis et al. *Molecular Biology of the Cell*. Garland Science, 2002.

[5] W. R. Atchley, J. Zhao, A. D. Fernandes, and T. Drüke. Solving the protein sequence metric problem. *PNAS*, 102(18):6395–6400, 2005.

[6] A. E. Badaczewska-Dawid, A. Kolinski, and S. Kmiecik. Computational reconstruction of atomistic protein structures from coarse-grained models. *Computational and Structural Biotechnology Journal*, 18:162–176, 2020.

[7] H. Berman, K. Henrick, and H. Nakamura. Announcing the worldwide Protein Data Bank. *Nat Struct Mol Biol*, 10(12):980, 2003.

[8] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[9] L. Breiman, J. Friedman, C. J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[10] C. Bystroff and D. Baker. Prediction of local structure in proteins using a library of sequence-structure motifs. *J Mol Biol*, 281(3):565–77, 1998.

[11] C.C. Chang and C.C Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[12] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[13] T. Chiu, D.P. Fang, J. Chenand Y. Wang, and C. Jeris. A Robust and Scalable Clustering Algorithm for Mixed Type Attributes in Large Database Environment. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 263–268, New York, NY, USA, 2001. Association for Computing Machinery.

[14] J. G. Voet D. Voet. *Biochemistry*. Wiley, 2010.

[15] A. G. de Brevern, C. Etchebest, and S. Hazout. Bayesian Probabilistic Approach for Predicting Backbone Structures in Terms of Protein Blocks. *PROTEINS: Structure, Function, and Genetics*, 41:271–287, 2000.

[16] Q. Dong, X. Wang, L. Lin, and Y. Wang. Analysis and prediction of protein local structure based on structure alphabets. *Proteins: Structure, Function, and Bioinformatics*, 72(1):163–172, 2008.

[17] M. Dudev and C. Lim. Discovering structural motifs using a structural alphabet: Application to magnesium-binding sites. *BMC Bioinformatics*, 8(106), 2007.

[18] D. Eisenberg. The discovery of the α-helix and β-sheet, the principal structural features of proteins. *Proceedings of the National Academy of Sciences*, 100(20):11207–11210, 2003.

[19] A. Eliasy and J. Przychodzen. The role of AI in capital structure to enhance corporate funding strategies. *Array*, 6:100017, 2020.

[20] G. Erdős and Z. Dosztányi. Analyzing Protein Disorder with IUPred2A. *Current Protocols in Bioinformatics*, 70(1):e99, 2020.

[21] A. K. Dunker et al. Intrinsically disordered protein. *J Mol Graph Model*, 19:26–59, 2001.

[22] A. P. Joseph et al. A short survey on protein blocks. *Biophys Rev*, 2(3):137–147, 2010.

[23] F. Pedregosa et al. *Documentation for Scikit-learn: Machine Learning in Python.*

[24] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] G. Faure et al. iPBAvizu: a PyMOL plugin for an efficient 3D protein structure superimposition approach. *Source Code Biol*, 5, 2019.

[26] I. Vetrivel et al. Knowledge-based prediction of protein backbone conformation using a structural alphabet. *PLoS ONE*, 12(11):e0186215, 2017.

[27] J. Barnoud et al. PBxplore: a tool to analyze local protein structure and deformability with Protein Blocks. *bioRxiv*, 2017.

[28] K. Peng et al. Optimizing long intrinsic disorder predictors with protein evolutionary information. *Journal of Bioinformatics and Computational Biology*, 3:35–60, 2005.

[29] M. Agathocleous et al. Protein Secondary Structure Prediction with Bidirectional Recurrent Neural Nets: Can Weight Updating for Each Residue Enhance Performance? In Harris Papadopoulos; Andreas S. Andreou; Max Bramer, editor, *6th IFIP WG 12.5 International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, volume AICT-339 of *Artificial Intelligence Applications and Innovations*, pages 128–137, Larnaca, Cyprus, October 2010. Springer.

[30] M. V. Akhila et al. A structural entropy index to analyse local conformations in intrinsically disordered proteins. *Journal of Structural Biology*, 210(1):107464, 2020.

[31] P. Romero et al. Sequence complexity of disordered protein. *PROTEINS: Structure, Function, and Genetics*, 42:38–48, 2001.

[32] R. Linding et al. Protein disorder prediction: implications for structural proteomics. *Structure*, 11:1453–1459, 2003.

[33] S. C. Lovell et al. Structure validation by $C\alpha$ geometry: ϕ, ψ and $C\beta$ deviation. *Proteins: Structure, Function, and Bioinformatics*, 50(3):437–450, 2003.

[34] S.F. Altschul et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 09 1997.

[35] X. Wu et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, 2008.

[36] C. Etchebest, C. Benros, S. Hazout, and A. G. de Brevern. A structural alphabet for local protein structures: improved prediction methods. *Proteins*, 59(4):810–27, 2005.

[37] F. Chollet et al. *Keras*, 2015. Software available from keras.io.

[38] J. S. Fetrow, M. J. Palumbo, and G. Berg. Patterns, structures, and amino acid frequencies in structural building blocks, a protein secondary structure classification scheme. *Proteins*, 27(2):249–71, 1997.

[39] Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[40] F.A. Gers, N.N Schraudolph, and J. Schmidhuber. Learning Precise Timing with Lstm Recurrent Networks. *J. Mach. Learn. Res.*, 3(null):115–143, March 2003.

[41] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[42] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, Berlin, Heidelberg, 01 2012.

[43] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[44] R. Heffernan, Y. Yang, K. Paliwal, and Y. Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for

improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017.

[45] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[46] International Business Machines Corporation. *IBM InfoSphere Warehouse. Creating mining models with Intelligent Miner Modeling.*

[47] International Business Machines Corporation. *IBM SPSS Modeler 18.2 Algorithms Guide* .

[48] International Business Machines Corporation. SPSS Modeler.

[49] D. R. Jandrlić, G. M. Lazić, N. S. Mitić, and Mi. D. Pavlović. Software tools for simultaneous data visualization and T cell epitopes and disorder prediction in proteins. *Journal of Biomedical Informatics*, 60:120–131, 2016.

[50] A. Jelović. RepeatsPlus — program for finding motifs and repeats in data sequences. *Journal of Bioinformatics and Computational Biology*, 2021.

[51] A. M. Jelovic, N. S. Mitic, S. Eshafah, and M. V. Beljanski. Finding Statistically Significant Repeats in Nucleic Acids and Proteins. *J Comput Biol*, 25(4):375–387, 2018.

[52] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section B*, 32:922–923, 1976.

[53] W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

[54] G. Karypis. YASSPP: better kernels and coding schemes lead to improvements in protein secondary structure prediction. *Proteins*, 64(3):575–86, 2006.

[55] G. V. Kass. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 29(2):119–127, 1980.

[56] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, 2014.

[57] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[58] R. Kolodny, P. Koehl, L. Guibas, and M. Levitt. Small libraries of protein fragments model native protein structures accurately. *J Mol Biol*, 323(2):297–307, 2002.

[59] M.A. Kramer. Autoassociative neural networks. *Computers and Chemical Engineering*, 16(4):313–328, 1992. Neutral network applications in chemical engineering.

[60] R. Linding, R. B. Russell, V. Neduva, and T. J. Gibson. GlobPlot: Exploring protein sequences for globularity and disorder. *Nucleic Acids Res*, 31(13):3701–3708, 2003.

[61] M Y. Lobanov and O. V. Galzitskaya. The Ising model for prediction of disordered residues from protein sequence alone. *Physical Biology*, 8(3):035004, 2011.

[62] M. Y. Lobanov, I. V. Sokolovskiy, and O. V. Galzitskaya. IsUnstruct: prediction of the residue status to be ordered or disordered in the protein chain by a method based on the Ising model. *J Biomol Struct Dyn*, 31(10):1034–1043, 2013.

[63] M. Basaldella et al. *Bidirectional LSTM Recurrent Neural Network for Keyphrase Extraction*, pages 180–187. 01 2018.

[64] A. G. de Brevern M. M. Maljković, N. S. Mitić. Prediction of Structural Alphabet Protein Blocks Using Data Mining. submitted, manuscript ID: ci-2021-006267.

[65] N. Srinivasan M. Tyagi, A. de Brevern and B. Offmann. Protein structure mining using a structural alphabet. *Proteins: Structure, Function, and Bioinformatics*, 71(2):920–937, 2008.

[66] C. Micheletti, F. Se, and A. Maritan. Recurrent Oligomers in Proteins: An Optional Scheme Reconciling Accurate and Concise Backbone Representations in Automated Folding and Design Studies. *PROTEINS: Structure, Function, and Genetics*, 40:662–674, 2000.

[67] B. Mészáros, G. Erdős, and Z. Dosztányi. IUPred2A: context-dependent prediction of protein disorder as a function of redox state and protein binding. *Nucleic Acids Research*, 46(W1):W329–W337, 2018.

[68] S. C. Nyburg. Some uses of a best molecular fit routine. *Acta Crystallographica Section B*, 30(1):251–253, 1974.

[69] B. Offmann, M. Tyagi, and A. de Brevern. Local Protein Structures. *Current Bioinformatics*, 2:165–202, 2007.

[70] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[71] R. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[72] G. N. Ramachandran, C. Ramakrishnan, and S. Viswanathan. Stereochemistry of polypeptide chain configurations. *J Mol Biol*, 7:95–9, 1963.

[73] H. Rangwala, C. Kauffman, and G. Karypis. svmPRAT: SVM-based Protein Residue Annotation Toolkit. *BMC Bioinformatics*, 10:439, 2009.

[74] J. S. Richardson. The Anatomy and Taxonomy of Protein Structure. volume 34 of *Advances in Protein Chemistry*, pages 167–339. Academic Press, 1981.

[75] M. Rooman, J. Rodriguez, and S. Wodak. Automatic definition of recurrent local structure motifs in proteins. *J Mol Biol*, 213(2):327–336, 1990.

[76] J. Schuchhardt, G. Schneider, J. Reichelt, D. Schomburg, and P. Wrede. Local structural motifs of protein backbones are classified by self-organizing neural networks. *Protein Eng*, 9(10):833–42, 1996.

[77] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[78] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *VLDB Conference*, pages 544–555, 1996.

[79] PN. Tan, M. Steinbach, A. Karpatne, and V. Kumar. *Introduction to Data Mining (2nd Edition)*. Pearson, 2nd edition, 2018.

[80] R. Thomson, T.C. Hodgman, Z. R. Yang, and A. K. Doyle. Characterizing proteolytic cleavage site activity using bio-basis function neural networks. *Bioinformatics*, 19(14):1741–1747, 2003.

[81] CH Tung, JW Huang, and JM. Yang. Kappa-alpha plot derived structural alphabet and BLOSUM-like substitution matrix for rapid search of protein structure database. *Genome Biol*, 8, 2007.

[82] R. Unger, D. Harel, S. Wherland, and J. L. Sussman. A 3D building blocks approach to analyzing and predicting structure of proteins. *PROTEINS: Structure, Function, and Genetics*, 5:355–373, 1989.

[83] I. Walsh, A. J. M. Martin, T. Di Domenico, and S. C. E. Tosatto. ESpritz: accurate and fast prediction of protein disorder. *Bioinformatics*, 28(4):503–509, 2012.

[84] G. Wang and R. L. Dunbrack Jr. PISCES: a protein sequence culling server. *Bioinformatics*, 19:1589–1591, 2003.

[85] G. Wang and R. L. Dunbrack Jr. PISCES: recent improvements to a PDB sequence culling server. *Nucleic Acids Res*, 33:W94–W98, 2005.

[86] C.M. Wilmot and J.M. Thornton. β-Turns and their distortions: a proposed new nomenclature. *Protein Engineering, Design and Selection*, 3(6):479–493, 05 1990.

[87] Z. R. Yang, R. Thomson, P. McNeil, and R. M. Esnouf. RONN: the bio-basis function neural network technique applied to the detection of natively disordered regions in proteins. *Bioinformatics*, 21(16):3369–3376, 06 2005.

[88] O. Zimmermann and U. H. E. Hansmann. LOCUSTRA: Accurate Prediction of Local Protein Structure Using a Two-Layer Support Vector Machine Approach. *Journal of Chemical Information and Modeling*, 48(9):1903–1908, 2008.

Biografija autora

Mirjana Maljković rođena je 13. novembra 1986. godine u Kanberi, Australija. Osnovnu školu i gimnaziju završila je u Beogradu. Školske 2005/2006. godine upisala je osnovne akademske studije na Matematičkom fakultetu u Beogradu, smer Informatika, i diplomirala je školske 2007/2008. godine sa prosekom 9,00.

Školske 2008/2009. godine upisala je master akademske studije na Matematičkom fakultetu u Beogradu, smer Informatika. Master akademske studije završila je 2010. godine odbranom master rada pod naslovom „Dizajn i implementacija podsistema za nastavničke beleške sistema Studinfo" pod rukovodstvom prof. dr Saše Malkova. Prosečna ocena na master studijama je 9,77.

Školske 2010/2011. godine je upisala doktorske akademske studije na Matematičkom fakultetu, smer Informatika. Položila je sve ispite predviđene planom i programom doktorskih studija sa prosečnom ocenom 10,00.

Od 2009. godine zaposlena je na Matematičkom fakultetu Univerziteta u Beogradu kao saradnik u nastavi na Katedri za računarstvo i informatiku; od 2011. godine kao asistent u nastavi na Katedri za računarstvo i informatiku, a od 2018. godine kao asistent praktične nastave na Katedri za računarstvo i informatiku. Do sada je učestvovala u izvođenju vežbi iz sledećih predmeta: Relacione baze podataka, Programiranje baza podataka, Razvoj softvera, Istraživanje podataka i Istraživanje podataka 1.

Osnovne oblasti interesovanja su joj baze podataka, istraživanje podataka i bioinformatika.

Прилог 1.

Изјава о ауторству

Потписани-а _____

број индекса _____

Изјављујем

да је докторска дисертација под насловом

_____

_____

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, _____

_____

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора _____

Број индекса _____

Студијски програм _____

Наслов рада _____

Ментор _____

Потписани/а _____

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, _____

_____

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

_____

_____

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, _____

_____