# *Cascading classifier application for topology prediction of TMB proteins*

Hassan B. Kazemian
*School of Computing and Digital Media*
*London Metropolitan University*
London, UK
h.kazemian@londonmet.ac.uk

Cedric Maxime Grimaldi
*School of Computing and Digital Media*
*London Metropolitan University*
London, UK
rx8879@gmail.com

***Abstract*—This paper is concerned with the use of a cascading classifier for trans-membrane beta-barrel topology prediction analysis. Most of novel drug design requires the use of membrane proteins. Trans-membrane proteins have key roles such as active transport across the membrane and signal transduction among other functions. Given their key roles, understanding their structures mechanisms and regulation at the level of molecules with the use of computational modeling is essential. In the field of bioinformatics, many years have been spent on the trans-membrane protein structure prediction focusing on the alpha-helix membrane proteins. Technological developments have been increasingly utilized in order to understand in more details membrane protein function and structure. Various methodologies have been developed for the prediction of TMB (transmembrane beta-barrel) proteins topology however the use of cascading classifier has not been fully explored. This research presents a novel approach for TMB topology prediction. The MATLAB computer simulation results show that the proposed methodology predicts transmembrane topologies with high accuracy for randomly selected proteins.**

## I. INTRODUCTION

There are three different computational problems related to trans-membrane beta-barrel and they can be identified and classified as: Trans-membrane beta-barrel detection and discrimination (from other types of proteins), trans-membrane topology prediction and trans-membrane beta-contacts prediction. A recent paper embarks upon a NN (Neural Network) technique and its comparison with hybrid- two-level NN-SVM (Support Vector Machines) methodology to classify inter-class and intra-class transitions to predict the number and range of beta membrane spanning regions. The computer simulation results demonstrate a significant impact and a superior performance of NN-SVM tests with a 5 residue overlap for signal protein over NN with and without redundant proteins for prediction of trans- membrane beta barrel spanning regions [1]. The efforts to beta-barrel topology prediction have been overshadowed and the accuracy of prediction could be improved. Early work on protein secondary structure prediction goes back to 1950's when Pauling et al. [2] explored some reasoning for the creation of alpha-helix and beta-strand local conformations. Chou and Fasman discovered that individual amino acids prefer a certain type of secondary structure. They developed the Chou-Fasman method in 1974 that is described in two different papers [3] and [4]. The aim of this paper is to evaluate the performance of a cascading classifier in the prediction of TMB topologies. The results of this paper would represent a potentially important advance in the prediction of beta-barrel trans-membrane protein topology using computational tools. Datasets used for trans-membrane beta-barrel proteins are usually of small size. A recent publication [5] evaluates the performance of various machine learning techniques based on small datasets with varying dimensionalities. From their study, they concluded that KNN (k-nearestneighbors), SVM and linear discriminant have the best predictive accuracy on small datasets. One of the latest methods used for predicting trans-membrane beta-barrel topologies is BOCTOPUS [6]. Three support vector machines are used to predict the local structural preferences for a residue. A HMM (Hidden Markov Model) model is used to obtain a topology model for a protein. In 2016, Hayat et al. introduced BOCTOPUS2 [7], an improved version of BOCTOPUS. The correct topology is predicted correctly in 69% of the proteins with BOCTOPUS2. It is more than 10% improvement compared to BOCTOPUS, the earlier method. The barrel domain identification as well as topology identification and prediction of the orientation of residues in trans-membrane beta-strands can be obtained with their latest method. Neighbors-based classification is a type of non-generalizing learning or instance-based learning. There is no attempt for the construction of a general internal model, but it simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point. The data class of the query point is obtained from the data class which has the most representatives within the nearest neighbors of that point.

KNN can be used for both classification and regression predictive problems. The KNN algorithm find the k-nearest neighbors by measuring the distance between the unknown sample and the training data samples. One parameter that is hidden from sight is the distance function. It is a key component of the KNN algorithm. It can have a strong effect on performance out of a KNN algorithm. The most common

distance function for numeric attributes is the Euclidian. It is symmetric, spherical and treats all dimensions equally. It has some down side as it is not always optimal for what needs to be done. It is sensitive to extreme differences in single attribute. It cannot be used for categorical data. Hamming distance function is used for categorical data. KNN algorithm is easy to understand and easy to implement classification technique. It can perform well in many situations.

Deep learning is a rapidly evolving field. A complete survey on the application of deep learning techniques in mining biological data has been provided in a recent article [23]. Positive results using deep learning towards trans-membrane beta-barrel topology prediction could also provide a major advance in bioinformatics. A secondary structure predictor called DNSS based on deeper neural networks [8] has been used and it achieved a prediction accuracy of 80.7%. It was however not targeted specifically at transmembrane beta-barrel topology prediction. This research takes the membrane beta barrel topology prediction further by applying novel techniques such as DNN, KNN and SVM as part of a cascading classifier. The computer simulation results show new results for TMB topologies prediction using a cascading classifier

## II. MACHINE LEARNING APPROACHES

The cascading classifier presented as part of this paper consists of combinations of various machine learning techniques including KNN, DNN and SVM. The model allows to use two methods or even three methods. Use KNN =1 or 0, useSVM = 1 or 0 and useNN = 1 or 0 are the list of parameters. Each machine learning techniques have their own characteristics that are summarized in this chapter.

### *A. K-Nearest Neighbors (KNN)*

Among all machine learning algorithms, Nearest Neighbors algorithms are the simplest. Neighbors-based classification is a type of non-generalizing learning or instance-based learning. KNN is particularly well suited for multi-modal classes as well as applications in which an object can have many class labels. Datasets used for trans-membrane beta-barrel proteins are usually of small size. As indicated in the introduction, KNN have one of the best predictive accuracy on small datasets. In the MATLAB implementation presented as part of this paper, KNN is one of three machine learning techniques that can be used as part of the cascading classifier within a combination. KNN is created using the `fitcknn` function. `fitcknn` is part of the statistics toolbox. At the input layer, a sliding encoding window will be used on each amino acid sequence. Prediction is based on the topology characteristic of the central residue in the window. A binary array of size 20 is used to encode each window position at the start of the implementation. Several model parameters can be modified such as the tie-breaking algorithm (`'BreakTies'`), the Nearest Neighbor search method (`'NSMethod'`), k value (`'NumNeighbors'`), maximum data points in node (`'Bucketsize'`), tie inclusion flag (`includeTies'`), distance (`'Distance'`) and exponent (`'exponent'`).

### *B. Deep Neural Network (DNN)*

Deep learning is a rapidly evolving field and positive results using this method towards trans-membrane beta-barrel topology prediction could provide a major advance in bioinformatics. Deep neural networks have become popular machine learning tools in recent years. In a recent paper, Heffernan et al. [8] achieved a secondary structure prediction accuracy of 82% by using a deep learning neural network. Recurrent neural networks provide successful results when applied to secondary structure prediction [09], [10]. Deep neural networks are able to learn complex patterns. A secondary structure predictor called DNSS based on deeper neural networks [11] achieves a prediction accuracy of 80.7%. This predictor is not targeted specifically at trans-membrane beta-barrel topology prediction. Hinton et al. define a deep neural network as a feed-forward artificial neural network that has more than one layer of hidden units between the inputs and its outputs [12]. For the MATLAB implementation presented as part of this paper, when a DNN is used part of a combination, `patternnet` is used for the creation of a pattern recognition neural network and `hsize` corresponds to the size of the hidden layer. An input layer, a hidden layer and an output layer are used to define the neural network. At the input layer, a sliding encoding window is used on each amino acid sequence.

### *C. Support Vector Machines (SVM)*

Another machine learning technique used as part of the cascading classifier is SVM. In the field of machine learning, a Support Vector Machine is a supervised learning technique that can be used for both classification and regression. It was first introduced in a 1992 article [13] in which the author introduced a training algorithm. In high-dimensional space, SVMs have a good generalization and the small number of training samples does not have an impact [14]. When we compare with empirical risk minimization principle that is used by a vast majority of neural networks, better results are obtained [15]. The goal is to minimize an upper bound of the generalization error by maximizing the margin between the data and the separating hyperplane [16]. SVMs don't over generalize in general whereas the neural networks can lead to over generalization often [17]. The performance of SVM depends largely on the kernels chosen. The best kernel of choice for a specific problem has to be researched. Smola et al. [18] gave an explanation of the relation between the standard regularization theory and the SVM kernel method. Other problems of SVMs, for the training and testing phases include size and speed. For a similar generalization performance, other neural networks are faster than SVMs [19].

## III. DATA PREPARATION

### *A. Datasets*

There are a number of databases that are available and are repositories for the structures and sequences of trans-membrane proteins. TOPDB (Topology data bank of transmembrane proteins) contains the comprehensive list of trans-membrane

proteins with topology information [20]. It has a total of 4190 trans-membrane proteins obtained from the literature and from public databases available on internet. The beta-barrel TOPDB entries can be downloaded directly from the website. The topdb_bp.txt file contains 123 TMB sequences. The BOCTOPUS2 dataset is the second dataset used for the implementation. It is the dataset that was used for the training/testing of the software/predictor BOCTOPUS2 which is a trans-membrane beta-barrel topology prediction tool [6]. It is available on the website of BOCTOPUS2 server. The BOCTOPUS2 dataset consists of 42 TMB sequences.

### *B. Data collection*

For BOCTOPUS2 dataset, two files available from the server.`boctopus2_crossvalidation_dataset.xlsx` and is named `boctopus2Labels.txt`.This file was created using the observed topologies data available in `boctopus2_dataset_sequenceannotation.txt`. Pore-facing (p) and lipid-facing (l) labels were manually replaced with M in order to have an i, o, M profile labels for each sequence. For the TOPDB dataset, the data file was manually curated. The observed topology represented with an X corresponds to the signal peptide. For the implementations, the signal peptide was ignored. The process of curation was similar to the BOCTOPUS2 dataset. The `topdb_bp.txt` was divided into two separate files `TOPBPLabels.txt` and `TOPBPSequence.txt`.

### *C. Data pre-processing*

In order to train the cascading classifier, datasets needed to be created and formatted for MATLAB. Structure arrays were created with a small program. A 1x42 structure array with 3 fields (header, sequence, and topology) was created for BOCTOPUS2 and a 1x123 structure array was created for the TOPDB dataset. The name of the structure arrays and the load files for the implementation are called `TOPBPdataset.mat` and `Boctopus2dataset.mat`.The fields include ‘header’ which corresponds to the annotation of a given protein sequence, ‘sequence’ which represents the protein sequence and ‘topology’ which represents the predicted topology.

## IV. COMPUTER SIMULATION IMPLEMENTATION AND RESULTS

The computer simulation was executed in MATLAB. MATLAB® is a fourth-generation programming language and a computing. It is easy to use and combines computing visualization and coding in the same environment. It has the capability to solve technically complex problems such as the formulations of matrices.

### *A. Creating and training the cascading classifier*

Data division for the cascading classifier is defined by the model. Data division is based on two parameters (tsPart1 and tsPart2). The first parameter defines how many data will be used in total for testing (This corresponds to tsPart1). So, if we choose tsPart1=0.8, it means 80% of data will be used for training and 20% for testing. From the selected 80%, we can decide which part will be used by the first layer and which part will be used by the second layer. This is defined by tsPart2.For tsPart1=0.8 and tsPart2=0.5, if we have a total a 100 data, 80 will be used for training in total. Among them, 50%=40 will be used for the first layer and 50%=40 will be used for the second layer. When not using DNN, the fraction of the first level can be set manually.

The model consists of two levels. Several selected models will be trained at the first level. The selected algorithms (KNN, SVM or DNN) are trained to predict the values of the class. In case two or more classifiers are selected, a second level SVM classifier is trained to predict the value of the class based on the probability predicted by those 2 or more models at the first level. This is a cascading classifier as the output of the first layer corresponds to the input of the second layer. In case, only one model is initially selected at the first level, the second layer classifier will not be trained.

The model allows to use a single method, 2 methods or even three methods. KNN =1 or 0, SVM = 1 or 0, and DNN = 1 or 0 are the list of parameters. If a parameter is equals to 0, it will not be used in the cascading classifier. All combinations are possible except for combination when all of them are equal to 0. If only one of them is set to 1, the application will work as one classifier. The selected classifier (equals to 1) will be trained and will show the results. If at least 2 parameters are set to 1, it means that several classifiers will be trained and then combined together by one more probability classifier. When multiple classifiers are chosen, a second level SVM will be trained.

### *B. Mondifying the cascading classifier parameters*

The model has four different basic configurations. The parameters of each machine learning algorithms can be modified.

When KNN has been used as part of the cascading classifier, various parameters have been modified. The classifier performs better with more than one neighbor and the best results are reached when the k value equals 8. 'BreakTies' corresponds to tie-breaking algorithm. The default value is 'smallest' but values such as 'nearest' or 'random' have been used. Nearest neighbor search method has been coded as a pair consisting of 'NSMethod' and 'kdtree' or 'exhaustive'. Distance metrics used include 'euclidean', 'cityblock', 'chebychev' or 'minkowski’. ‘minkowski’ corresponds to the Minkowski distance. The default exponent is 2. ’Bucketsize’ was also modified. It corresponds to the maximum data points in node.50 is the default value. ‘includeTies’ is another parameter that has been modified. It can be set to true or false. When the value is ‘true’, the model takes into consideration all nearest neighbors with a distance equal to the k-th smallest distance in the output arguments. ‘exponent’ corresponds to the Minkowski distance exponent and can be added to one of the sub-parameters of a

KNN. It is only applicable when ‘Distance’ is specified as ‘Minkowski’.

When DNN has been used as part of the cascading classifier, various parameters have been modified. An input layer, a hidden layer and an output layer are used to define the deep neural network. At the input layer, a sliding encoding window was used on each amino acid sequence. 20 binary bits is the most common distributed encoding method. During the computing, various hidden layer sizes have been used from 2 to 1000. Various training algorithms have been used with the computation in order to evaluate the accuracy of prediction. Trainscg (scaled conjugate gradient) is the default training algorithm available in MATLAB. Other available trainings include Trainrp (resilient backpropagation algorithm), Traincgb (Conjugate Gradient with Powell/Beale Restarts), Traincgf (Fletcher-Powell Conjugate Gradient) in which weight and bias values are updated according to the conjugate gradient backpropagation with Fletcher-Reeves updates [21] and Traincgp (Polak-Ribiére Conjugate Gradient Training algorithm) in which weight and bias values are updated according to the conjugate gradient backpropagation with Polak-Ribiére updates. Trainscg provides the best results. The transfer function, logsig, allows the signals received from the input layer to be transformed in each hidden layer, was used and give better results than the hyperbolic tangent sigmoid transfer function,Tansig(N). Various functions were used for the data division. If 'dividerand' is selected for net.divideFcn, the data will be randomly divided into three sets. The ratio that is used by default is 0.7/0.15/0.15. It corresponds to the ratio for training, testing and validation. The data is randomly divided so that 70% of the samples are assigned to the training set, 15% to the validation set, and 15% to the test set. ‘divideblock’ will divide the data into contiguous blocks. ‘divideind’ will divide the data by index. The data division is an automatic process that happens when the network is trained. It will divide the data into a training set, validation set and testing set. All types of division functions were used during the implementation in order to evaluate the differences in predictive accuracy. To train a neural network, some measure of error between computed outputs and the desired target outputs of the training data is needed. The most common measure of error is called mean squared error. The mean squared error (MSE) of an estimator calculates the squared error. Sum squared error (SSE) performance function was also used. The trends of the training, validation, and test errors as training iterations pass are displayed with function plotperform. When one of several conditions defined in net.trainParam is met, the training process will stop. It can be for example that the number of epochs, referred as repetitions, is obtained. Also, it can be that the maximum amount of time is achieved, the performance is minimized to the goal, the performance gradient falls below min_grad or the validation performance has increased more than max_fail times since the last time it decreased (when using validation).

When SVM has been used as part of the cascading classifier, various parameters have been modified. 'SaveSupportVectors' is one of the properties of a SVM that has been modified. 'KernelFunction' can also be modified. 'SaveSupportVectors' can be 'true' or 'false' and 'KernelFunction' can be 'linear 'or 'rbf' or 'polynomial'. 'BoxConstraints' is a parameter than can be added and modified. 'BoxConstraints' is characterized by the pair that consist of 'BoxConstraints' and a positive scalar. For one-class learning, it is set to 1. An example of syntax would be 'BoxConstraints', 100. When this parameter is increased, the SVM classifier will have less support vectors. When this parameter is increased, the training duration is longer. Few runs have been executed with box constraints ranging from 1 to 1000. 'CacheSize' is another parameter that can be modified. 'CacheSize' is characterized as a pair that consist of 'CacheSize' and 'maximal' or a positive scalar. When 'CacheSize' is 'maximal', it keeps enough memory to be able to hold the entire m-by-m Gram matrix. When 'CacheSize' is a positive scalar, it keeps CacheSize megabytes of memory for the training of the classifier. For large problems, it is better to specify enough cache size. The default value is 1000. An example of syntax would be: 'CacheSize', 'maximal'.'IterationLimit' is another parameter that can be modified. It’s defined as the maximal number of numerical optimization iterations. 'IterationLimit' is characterized as the pair that consist of 'IterationLimit' and a positive integer. It returns a model that is trained even if the optimization routine does not converge. Mdl.ConvergenceInfo does contain convergence details. When the iteration limit is very low or very high, the optimization is slowed down. When the iteration limit is too tight, the algorithm spends too much time doing optimization of the dual variables of a single example. 'ClipAlphas' is another parameter that can be modified. It is characterized as a pair that consist of 'ClipAlphas' and either 'true' or 'false'. If 'false', the software will not modify the alpha coefficients during the optimization. 'ClipAlphas' can affect SMO and ISDA convergence. 'Solver' is another parameter that can be modified. It is specified as the comma-separated as 'Solver' and either 'ISDA' or 'L1QP' or 'SMO'. The default is 'ISDA', if 'OutlierFraction' is set to a positive value and in the case of two-class learning. It will be 'SMO' otherwise. 'SMO' refers to Sequential Minimal Optimization. 'ISDA' refers to Iterative Single Data Algorithm. 'OutlierFraction' corresponds to the proportion of outliers in the data used for training. It is characterized as a pair that consist of 'OutlierFraction' and a numeric scalar ranging from 0 and 1. SVMs can be affected by outliers and methods have been developed to mitigate the effects of outliers on SVMs. 'DeltaGradientTolerance' is another parameter that can be modified. It is written as a pair that consist of 'DeltaGradientTolerance' and a nonnegative scalar. 'DeltaGradientTolerance' is equivalent to the tolerance for the gradient difference between upper and lower violators obtained by SMO or ISDA. 'GapTolerance' is another parameter that can be modified. It is written as a pair that consist of 'GapTolerance' with a nonnegative scalar. It is equivalent to the feasibility gap tolerance that is obtained by SMO or ISDA. When the value is equal to 0, then MATLAB does not use the feasibility gap tolerance when checking for optimization convergence. 'KernelOffset' is another parameter that can be modified. It is written as a pair that consist of 'KernelOffset' with a nonnegative scalar. MATLAB will add 'KernelOffset' to each element of the Gram matrix. If the solver is SMO, the default value is 0. It’s 0.1 if the solver is ISDA. 'KernelScale'

is another parameter that can be modified. It is written as a pair that consist of 'KernelScale' with a positive scalar or 'auto'. MATLAB will divide all elements of the predictor matrix X by the value of 'KernelScale' and then the software will apply the appropriate kernel norm in order to compute the Gram matrix. If it is written 'auto' MATLAB will select an appropriate scale factor with the use of a heuristic procedure. 'KKTTolerance' is another parameter that can be modified. It is written as a pair that consist of 'KKTTolerance' and a positive scalar. KKTTolerance is equivalent to Karush-Kuhn-Tucker complementary conditions violation tolerance. If 'KKTTolerance' is equal to 0, then MATLAB will not use the KKT complementary violation tolerance in order to check for optimization convergence. If the solver is SMO, the default value is 0 otherwise it's 1e-3 if the solver is ISDA. 'Numprint' is another parameter that can be modified. It is written as a pair that consist of 'Numprint' with a nonnegative integer. It is corresponding to the number of iterations between optimization diagnostic message output. If 'Verbose', 1 and 'Numprint', Numprint are used then the software will display all optimization diagnostic message from SMO and ISDA every Numprint iteration in the command window. 'Verbose' is another parameter that can be modified. It is written as a pair that consist of 'Verbose' with either 0, 1 or 2. It is corresponding to the verbosity level. It is controlling the amount of optimization information that the software will display in the command window and will be saving as the structure Mdl.ConvergenceInfo.History. 'PolynomialOrder' is another parameter that can be modified. It is written as a pair that consist of 'PolynomialOrder' and a positive integer. It is corresponding to the polynomial kernel function order. The default value is 3. 'ShrinkagePeriod' is another parameter that can be modified. It is written as a pair that consist of 'ShrinkagePeriod' with a nonnegative integer. It is corresponding to the number of iterations between movement of observations from active to inactive set. The software will not shrink the active set if it has a value of 0. Convergence can be speeded up with shrinking when the support vector set is much smaller than the number of data in the training dataset. 'Standardize' is another parameter that can be modified. It is written as a pair that consist of 'Standardize' and 'true' or 'false'. It is corresponding to a flag to standardize the predictor data. MATLAB will center and scale each column of the predictor data (x) by the weighted column mean and standard deviation if it is set as 'true'. The software will not standardize the data that is contained in the dummy variable columns and that is generated for categorical predictors. MATLAB will train the classifier using the standardized predictor matrix, if it is set as 'true'. The unstandardized data will be stored in the classifier property x. For the function fitecoc, 'Cost' is another parameter that can be modified. It is written as a pair that consist of 'Cost' and a square matrix or structure. It is corresponding to the misclassification cost.

The best results are obtained when KNN, SVM and DNN are used at layer 1 and SVM is used at layer 2.

### *C. Results of the cascading classifier*

Multiple runs have been executed in MATLAB using different parameters configurations. Various ratio combinations such as 50:50, 60:40, 70:30, 75:25 and 80:20 have been used for the split between training and testing data in tsPart1. Best results for the cascading classifier were obtained using a split with 80% of data used for training and 20% used for testing in tsPart1 and 42% in tsPart2. Best results are obtained with parameters configured to a window size of 65, Bits encodings of 50, Hidden layer size of 50, logsig transfer function, scaled conjugate gradient for the training function, 'Sum' Performance Function and 'Dividerand' data division for the DNN part of the cascading classifier and k-value of 8, exhaustive nearest neighbor search method, random tie-breaking algorithm for the KNN part of the cascading classifier. For the SVM part of the cascading classifier, a polynomial kernel function, 'SaveSupportVectors' is equivalent to 'true' have been used as well as default values for the box constraint, cache size, Solver, tolerance to gradient difference, feasibility gap tolerance, Maximal number of optimization iterations, kernel offset parameter, kernel scale, Karush-Kuhn-Tucker complementarity conditions violation tolerance, v parameter for one-class learning, number of iterations between optimization diagnostic message output, expected proportion of outliers in training data, Polynomial kernel function order, number of iterations between movement of observations from active to inactive set, flag to standardize predictor data and verbosity level. It is interesting to note that an increase in the amount of data does produce better results. Two runs were executed using the same parameters with different dataset available: TopBP Dataset (1x123) and BOCTOPUS2 Dataset (1x42). Accuracy is 63.6% when using BOCTOPUS2 Dataset and 72.8% when using TopBP Dataset.

The function assessPerformance is used to display performance in the form or ROC curves, confusion matrix and bar. The Receiver Operating Characteristic (ROC) is a plot of the true positive rate (sensitivity) versus the false positive rate (1 - specificity). A confusion matrix is a table that is predominately used to describe the performance of a classification model or classifier on a set of test data for which the true or actual values are known. Confusion matrix allows visualization of the performance of a classifier. Topologies predictions can be evaluated in more detail by calculation of assorted quality indices as well. A confusion matrix plot for the target and output data is returned with Plotconfusion (targets, outputs). In the confusion matrix, each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. It provides a visualization of the performance of the algorithm. Fig.1 represents the confusion matrices for the cascading classifier.


Fig. 1. Confusion Matrix

The receiver operating characteristic for each output class is plotted with Plotroc (targets, outputs). When the curve goes to the left and top edges of the plot it means that the classification is better. The sensitivity measures the proportion of actual positives that are correctly identified as such. The false positive is also known as the fall-out. Fall-out is closely related to specificity and is equal to (1 - specificity).

The ROC curve is thus the sensitivity as a function of the fall-out. A perfect predictor would be described at 100% sensitive. The closer the ROC curve is to the upper left corner (100% sensitivity, 100% specificity), the higher the overall accuracy. Fig.2 represents the ROC curves for KNN compared with DNN.

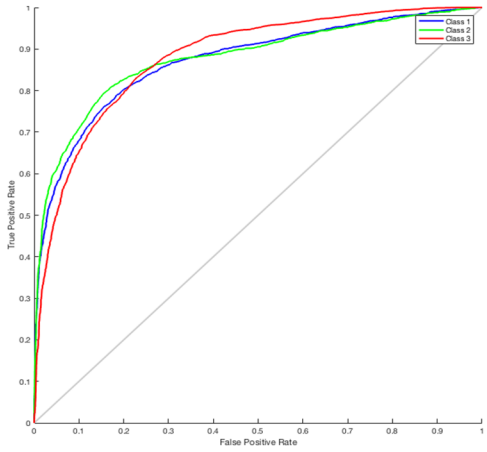


Fig. 2. ROC curve

Topologies predictions can be evaluated in more detail by calculation of assorted quality indices [22]. Those indices demonstrate if a topology is accurately predicted and whether there was over-prediction or under-prediction. Fig.3 represents the correctly predicted positions in percentage as observed and predicted. `Bar (x,y)` is the MATLAB function that was used to create the bar graphs.

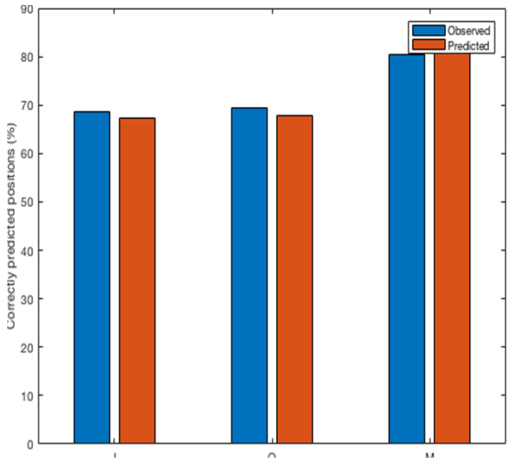


Fig. 3. Quality indices performance

## V. CONCLUSION

This paper discusses the applications of KNN and DNN for TMB topology prediction. Training and testing was performed on curated TOPDB and BOCTOPUS2 datasets. The model allows to use a single method, two methods or even three methods that is using KNN =1 or 0, SVM = 1 or 0, and DNN = 1 or 0. The computer simulation results using a dataset including 42 TMB sequences respectively reveal a TMB topology prediction accuracy of 75.2%. The accuracy of 75.2% is for one scenario combination where layer one is SVM, KNN and DNN, and layer two is SVM. The output of layer one is the input of layer two in the cascading classifier. This represents a significant improvement in the prediction of beta-barrel trans-membrane topology prediction

## ACKNOWLEDGMENT

This research work has been carried out at School of Computing and Digital Media, London Metropolitan University.

## REFERENCES

[1] H. Kazemian, S. A. Yusuf, K. White, and C. M. Grimaldi, “NN approach and its comparison with NN-SVM to beta-barrel prediction” *Expert Systems with Applications*, vol. 61, pp. 203–214, Nov. 2016

[2] L. Pauling and R. B. Corey, “The Pleated Sheet, A New Layer Configuration of Polypeptide Chains” *Proceedings of the Natlional Academy of Sciences U. S. A*, vol. 37, no. 5, pp. 251–256, May 1951

[3] P. Y. Chou and G. D. Fasman, "Conformational parameters for amino acids in helical, β-sheet, and random coil regions calculated from proteins" *Biochemistry*, vol. 13, no. 2, pp. 211–222, Jan. 1974

[4] P. Y. Chou and G. D. Fasman, "Prediction of protein conformation" *Biochemistry*, vol. 13, no. 2, pp. 222–245, Jan. 1974

[5] S.Sharma and V. Sharma, "Performance of Various Machine Learning Classifiers on Small Datasets with Varying Dimensionalities". *Circulation in Computer Science*, Vol.1, no.1, pp. 30-35, Jul. 2016

[6] S. Hayat and A. Elofsson, "BOCTOPUS: improved topology prediction of transmembrane β barrel proteins" *Bioinformatics*, vol. 28, no. 4, pp. 516–522, Feb. 2012

[7] S. Hayat, C. Peters, N. Shu, K. D. Tsirigos, and A. Elofsson, "Inclusion of dyad-repeat pattern improves topology prediction of transmembrane β-barrel proteins" *Bioinformatics*, vol. 32, no. 10, pp. 1571–1573, Jan. 2016

[8] R. Heffernan *et al.*, "Improving prediction of secondary structure, local backbone angles, and solvent accessible surface area of proteins by iterative deep learning" *Scientific reports*, vol. 5, p. 11476, Jun. 2015

[9] G. Pollastri, D. Przybylski, B. Rost, and P. Baldi, "Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles" *Proteins*, vol. 47, no. 2, pp. 228–235, May 2002

[10] A. Daniel, "Prediction of Protein Secondary Structure using Long Short-Term Memory. Technical report, Halmstad University, pp 1-35, Jan. 2003

[11] M. Spencer, J. Eickholt, and J. Cheng, "A Deep Learning Network Approach to Ab Initio Protein Secondary Structure Prediction" *IEEE/ACM Transactions on Computational Biology. Bioinformatics*, vol. 12, no. 1, pp. 103–112, Jan. 2015

[12] G. Hinton *et al.*, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups" *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov. 2012

[13] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers" in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, USA, pp. 144–152, Jul. 1992

[14] K. Jonsson, J. Kittler, Y. P. Li, and J. Matas, "Support vector machines for face authentication" *Image and Vision Computing*, vol. 20, no. 5–6, pp. 369–375, Apr. 2002

[15] J. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "Face recognition using feature optimization and /spl nu/-support vector learning" in *Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop (IEEE Cat. No.01TH8584)*, pp. 373–382, Sep. 2001

[16] S. Amari and S. Wu, "Improving Support Vector Machine Classifiers by Modifying Kernal Functions" *Neural Networks*, vol. 12, no. 6, pp. 783–789, Jul. 1999

[17] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997

[18] A. J. Smola, B. Schölkopf, and K. R. Müller, "The connection between regularization operators and support vector kernels" *Neural Networks*, vol. 11, no. 4, pp. 637–649, Jun. 1998

[19] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998

[20] G. E. Tusnády, L. Kalmár, and I. Simon, "TOPDB: topology data bank of transmembrane proteins" *Nucleic Acids Research*, vol. 36, no. Database issue, pp. D234–239, Jan. 2008

[21] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients" *The Computer Journal*, vol. 7, no. 2, pp. 149–154, Jan. 1964

[22] W. Kabsch and C. Sander, "How good are predictions of protein secondary structure?" *FEBS Letters*, vol. 155, no. 2, pp. 179–182, May 1983

[23] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, "Applications of Deep Learning and Reinforcement Learning to Biological Data" *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2063–2079, Jun. 2018