POLITECNICO DI TORINO

Binary Classification of Protein Functions

Candidate
Simone Lucio Cannavó

Febbraio 2024

Abstract

In this article, we explore the task of protein function prediction using machine learning techniques. Leveraging data from the UniProt database, we construct and evaluate three classification models: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest. Through rigorous validation and parameter optimization, we identify the Random Forest as the most promising model for predicting protein function. We present validation results, parameter optimization using Grid Search, and performance evaluation on a test dataset. The Random Forest model demonstrates strong predictive accuracy, precision, recall, and F1-score, confirming its effectiveness in protein function prediction tasks.

Contents

1 Introduction 3
- 1.1 Protein . . . 3
  - 1.1.1 Training,Validation and Test . . . 4
- 1.2 Preprocessing . . . 4
  - 1.2.1 Features . . . 4
  - 1.2.2 One hot Encoding . . . 5
  - 1.2.3 Padding . . . 5
  - 1.2.4 PCA . . . 6

2 Classification Models 9
- 2.1 Theory Of models . . . 9
- 2.2 Random Forest . . . 9
- 2.3 KNN . . . 10
- 2.4 SVM . . . 10
- 2.5 Validation . . . 11

3 Results 13
- 3.1 Results . . . 13
  - 3.1.1 Validation Set Results . . . 13
  - 3.1.2 Confusion Matrices . . . 13
  - 3.1.3 Test Set Results . . . 15

4 Conclusion 17

Chapter 1

Introduction

In the realm of molecular biology and bioinformatics, predicting protein function stands as a central and captivating challenge. Proteins play a fundamental role in regulating a wide array of biological processes within cells, and understanding their functions is of greatest importance for comprehending cellular mechanisms and developing targeted therapies.

This work revolves around this very goal: developing a classification model capable of predicting protein function based on their molecular attributes. Starting from data sourced from the prestigious Uniprot database, we've constructed a robust machine learning framework to tackle this challenge.

Through a rigorous methodology involving data partitioning into training, validation, and test sets, we've trained and evaluated three classification models: K-Nearest Neighbors (KNN), Random Forest, and Support Vector Machine (SVM). Employing grid search techniques, we've fine-tuned the models' parameters to maximize their predictive performance.

1.1 Protein

From the UniProt database, 1000 protein sequences were selected, with 500 associated with transport function and 500 associated with transducer function.

Proteins with a transport function are responsible for the movement of molecules or ions across cellular membranes or within the cell. This process is crucial for various cellular activities such as nutrient uptake, waste removal, and signal transduction. Transport proteins facilitate the movement of substances across biological membranes using various mechanisms, including active transport, passive diffusion, and facilitated diffusion.

On the other hand, proteins with a transducer function are involved in signal transduction pathways. These proteins transmit signals from the cell surface to the cell's interior, where they regulate cellular responses such as gene expression, metabolism, and cell growth. Signal transduction pathways are essential for cells to respond to external stimuli such as hormones, growth factors, and environmental signals, allowing them to adapt and survive changing conditions.In this article, our aim is to develop a model capable of distinguishing between two types of protein functions using sequence structure.

This structure includes both the order of amino acids composing the sequence and the type of amino acids present.

Therefore, the analysed predictive model, given a set of protein sequences, can identify whether a given sequence is associated with the transport function or the transducer function.

1.1.1 Training,Validation and Test

We started with 1000 protein sequences, of which 500 were used to construct the training dataset. In this dataset, we trained three classification models: Random Forest, SVM, and KNN.

Subsequently, we split the remaining 500 sequences into two sets of 300 and 200 sequences, respectively, for validation and testing. We then proceeded with grid search to optimize the parameters of the Random Forest and SVM models, focusing on models that showed the best performance.

After identifying the best parameters during the validation phase, we tested the two optimized models on the test set. In total, the training set had 500 observations, while the validation and test sets had 300 and 200 observations for validation and 200 observations for testing, respectively.

1.2 Preprocessing

1.2.1 Features

Initially, we considered using the frequency of each amino acid in the sequence and the length of the sequence itself as attributes, leading to a vector of 23 dimensions for each sequence. However, this approach did not yield the desired results.

Subsequently, we adopted a more detailed approach to capture the information contained in protein sequences. We created a dataframe where each row represents a protein sequence, and the number of columns is determined by the maximum length among all sequences present in the dataframe, multiplied by the number of amino acids, which is 22.

For example, if we have three sequences with lengths of 10, 20, and 30 respectively, the dataframe will have three rows and 660 columns (30 * 22).

Each cell in this dataframe represents the presence or absence of a particular amino acid at a specific position in the sequence. So, if an amino acid is present at a certain position, the value of the corresponding cell will be 1; otherwise, it will be 0. This scheme allows for a comprehensive and detailed representation of protein sequence features, preparing them for subsequent analysis and modeling.

The technique employed here is that of one hot encoding, which will be discussed in the next chapter.

1.2.2 One hot Encoding

Preprocessing of protein sequences is essential to prepare the data for analysis and modeling. Initially, it's important to consider that protein sequences are composed of a series of amino acids, each of which is represented by a specific letter. Here are the 22 letters of the amino acids:

$$A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y.$$

However, treating the sequences directly in this manner is impractical for analysis. Therefore, it's necessary to appropriately transform the observations so that we can construct a suitable dataframe for subsequent modeling phases, such as classification.

To face this challenge, the technique of one hot encoding has been chosen. This technique involves representing each amino acid in the sequence with a binary vector of length equal to the total number of amino acids. Each vector represents an amino acid, and only one entry will be set to 1 (indicating the presence of the corresponding amino acid), while all others will be set to 0.

For example, let's consider the protein sequence "ACD". Using one hot encoding, this sequence is represented as a matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For example the first cell corresponds to ammno acid A in the first position, the second is amino acid C in first position.In this matrix, each row represents an amino acid, while the columns represent the 22 possible amino acids in the order listed above. Thus, the sequence "ACD" is represented by three rows, each with a 1 in the column corresponding to the amino acid present in the sequence and zeros in all other columns.

One hot encoding allows us to represent protein sequences in a way that can be used as input in machine learning algorithms, enabling them to recognize and learn patterns within these sequences.

In our practical implementation, we concatenated each row of this matrix into a single row, creating a unique vector for each sequence.

1.2.3 Padding

In our dataframe, each protein sequence is represented as a vector of variable length, where each position in the vector corresponds to an amino acid. However, to ensure that all sequences have the same length, we employ the padding technique.

Padding involves adding special values, known as "padding tokens," to the end of shorter sequences so that they match the length of the longest sequence in the dataframe. This ensures that each sequence is uniformly aligned and can be processed consistently by machine learning algorithms.

In our case, we chose to use 0 as the padding value. Therefore, if a sequence is shorter than the maximum length, zeros will be appended to the end of the sequence until it reaches the desired length.

This process allows us to efficiently handle variable-length sequences in the dataframe, ensuring that they are all of the same size and thereby facilitating data analysis and modeling.

1.2.4 PCA

Clearly, a significant issue, both statistically and computationally, is the number of features present. Protein lengths vary considerably, with sequences extending beyond 500 amino acids. Considering that the number of features equals the length of the longest sequence multiplied by the number of amino acids (22), we are dealing with enormous dimensionality. The problem is both computational and statistical.

Having a large number of features poses a computational challenge because it increases the complexity and computational cost of training and evaluating machine learning models. With a high-dimensional feature space, algorithms may require more memory and processing power, leading to longer training times and slower performance. This can be particularly problematic when working with large datasets or limited computational resources.

Additionally, the presence of numerous features can lead to statistical challenges such as overfitting. Overfitting occurs when a model learns to capture noise or random fluctuations in the training data, rather than the underlying patterns or relationships. With a high-dimensional feature space, there is a greater risk of overfitting because the model may become too complex and adapt too closely to the training data, resulting in poor generalization to unseen data.

Therefore, it's crucial to carefully consider feature selection and dimensionality reduction techniques to address both computational and statistical challenges associated with high-dimensional data. In this context, we decided to use Principal Component Analysis (PCA) to reduce the dimensionality of our feature space. PCA is a technique that transforms the original variables into a new set of variables, called principal components. These components are a linear combination of the original variables and are ordered such that the first explains the maximum variance in the data, followed by the second, and so on. To select the optimal number of principal components, we observed the Proportion of Variance Explained (PVE) by each principal component. PVE provides a measure of the amount of variance in the original data contained in each principal component, allowing us to choose the appropriate number of principal components to strike a balance between model complexity and explained variance.Below is a graph where the x-axis represents the number of principal components and the y-axis represents the total variance of the data. The original number of features is 138731 and the structure is: It can be seen from the graph that with 150 components, the explained variance is already quite high, indicating a significant reduction in the problem despite having more than 100,000 variables,the choice of number of features is 150.

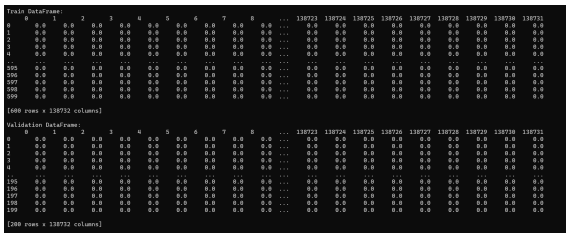


Figure 1.1. Original dataset

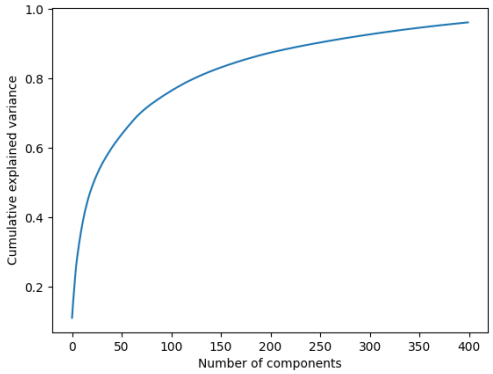
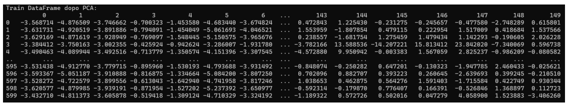


Figure 1.2. Cumulative variance vs number of components

After pca dataset is:



Train DataFrame dopo PCA:

	0	1	2	3	4	5	6	...	143	144	145	146	147	148	149
0	-3.568714	-4.876509	-3.746662	-0.700323	-1.453580	-4.683440	-3.674824	...	0.472843	1.225430	-0.231275	-0.245657	-0.477580	-2.748289	0.615801
1	-3.631731	-4.920519	-3.891886	-0.794091	-1.454049	-5.061693	-4.046521	...	1.553959	-1.807854	0.479115	0.222954	1.517009	0.418684	1.537566
2	-3.629169	-4.871619	-3.928949	-0.769097	-1.548445	-5.150575	-3.965676	...	0.238557	-1.681754	1.275459	1.479434	1.142293	-0.190605	2.026228
3	-3.384412	-3.750163	-3.002355	-0.425924	-0.942624	-3.286007	-1.931780	...	-3.782166	13.588536	-14.207221	15.813412	23.842020	-7.340069	0.596738
4	-3.490463	-4.089944	-3.492516	-0.713779	-1.350574	-4.151396	-3.307545	...	-4.572880	9.950942	-0.003383	1.567059	2.825237	-0.986209	-0.080582
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
595	-3.531438	-4.912770	-3.779715	-0.895960	-1.530193	-4.793688	-3.931492	...	-0.848074	-0.250282	0.647201	-0.130323	-1.947785	2.460433	-0.025621
596	-3.593367	-5.051187	-3.910888	-0.816875	-1.334664	-5.084200	-3.807250	...	0.702096	0.882707	0.393223	0.260645	-2.639693	0.399245	-0.210510
597	-3.528272	-4.722579	-3.899556	-0.613043	-1.642940	-4.741958	-3.817246	...	1.038653	0.462875	0.564276	1.591403	-1.715584	0.422749	0.930344
598	-3.620577	-4.879985	-3.939191	-0.871954	-1.527202	-5.237392	-3.650977	...	-0.592314	-0.179870	0.776407	0.166391	-0.526846	1.368897	0.112723
599	-3.432710	-4.811373	-3.605878	-0.519418	-1.309124	-4.710329	-3.324192	...	-1.189322	0.572726	0.502016	0.047279	4.058900	1.523883	-3.406260

Figure 1.3. datset after PCA

Chapter 2

Classification Models

2.1 Theory Of models

2.2 Random Forest

The Random Forest is a machine learning algorithm that relies on the construction of a set of decision trees. These trees are trained on random subsets of the original data, using a technique called bootstrap.

Bootstrap is a statistical technique involving the random extraction of samples with replacement from the original dataset. This means that each sample can be selected multiple times and other samples may not be selected at all. In practice, different subsets of data are created, each of which may contain duplicate or missing observations from the original.

Each decision tree in the Random Forest is trained on one of these data subsets. Additionally, during the construction of each tree, only a random subset of the available variables is selected to create a split at each node of the tree. Typically, a number of variables equal to $\sqrt{p}$, where p is the total number of variables in the original dataset, is chosen.

This approach leads to different decision trees, each trained on a different data subset and using only a random subset of the available variables. Consequently, the trees in the Random Forest are independent of each other, which results in greater diversity among the models.

This diversity is important because it reduces the correlation between the models and leads to greater robustness and generalization of the ensemble. If all the trees were similar, the ensemble might not significantly improve performance compared to a single tree. By using the Random Forest, however, models are obtained that are different from each other, each with its own data splitting characteristics.

In conclusion, the Random Forest is preferred over bootstrap because it utilizes the bootstrap approach to create data subsets, but adds diversity to the decision trees by using random variable selection during training. This results in a more robust and generalizable ensemble of models.

2.3 KNN

The main idea of K-Nearest Neighbors (KNN) is to classify an instance based on the class membership of its nearest neighbors. More formally, given a dataset $\tau = \{(x_i, y_i)\}_i^n$, where $y_i \in \{0, ..., c-1\}$ represents the class memberships and α is a new instance to be classified, KNN operates as follows:

1. Calculate the distance between the instance α and all other points in the dataset. The Euclidean distance is often used, but other distance metrics can also be utilized.

2. Select the k points in the dataset that are closest to α based on the calculated distance in the previous step. These points form a subset called $\sigma(\alpha)$, containing the pairs (x_i, y_i) of the k nearest points along with their respective class memberships.

3. Assign α the class label that is most common among the points in $\sigma(\alpha)$. In other words, the most represented class among the k neighbors of α will be the class label assigned to α.

In summary, KNN classifies an instance α based on the classes of its k nearest neighbors in the dataset. The choice of the parameter k is crucial and can impact the model's performance: smaller values of k may lead to higher variance in results, while larger values may lead to higher bias.

2.4 SVM

Support Vector Machine (SVM) is a supervised learning model used for classification and regression tasks. Formally, given a training dataset $\{(x_i, y_i)\}_{i=1}^n$, where x_i is a feature vector and y_i is the corresponding class label, SVM aims to find an optimal hyperplane $w \cdot x + b = 0$ that maximally separates points of different classes.

The hyperplane is defined by the linear combination $w \cdot x + b$, where w is a weight vector and b is the bias term. The distance between the hyperplane and the nearest point of each class, known as support vector, should be maximized.

Mathematically, the objective of SVM can be formulated as an optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to the constraints:

$$y_i(w \cdot x_i + b) \geq 1, \quad \forall i = 1, \ldots, n$$

where $\| \cdot \|^2$ represents the squared Euclidean norm, y_i are the class labels taking values of either $+1$ or -1, and x_i are the feature vectors.

SVM can be extended to address nonlinear problems by using kernel functions. The idea is to map the data into a higher-dimensional space where the data can be linearly separated. This allows SVM to find complex separation hyperplanes even for nonlinearly separable data.

In summary, SVM is a mathematical model that seeks to find the optimal hyperplane to maximize the separation between different classes in the feature space, making it a versatile and powerful method for classification and regression.

2.5 Validation

During the validation phase of our machine learning model, we conducted a comparative analysis among three widely used algorithms: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest. The objective was to identify which of them performed best on our validation dataset.

After training each model on the training set and evaluating their performance on the validation set, we observed that the Random Forest exhibited the best predictive accuracy. However, to further optimize the Random Forest's performance, we decided to perform a Grid Search.

Grid Search is a technique used to identify the optimal combination of a model's parameters. Essentially, it's a systematic process that explores a predefined grid of parameters and evaluates the model's performance on each combination. This allows us to identify the parameters that maximize the model's performance.

We used Grid Search to optimize the parameters of the Random Forest, which include:

- **Criterion ('criterion': 'gini')**: The Gini criterion is used to measure the purity of decision tree splits within the Random Forest ensemble. It seeks to minimize impurity during tree construction.
- **Max Depth ('max_depth': 5)**: It limits the maximum depth of trees in the ensemble. This helps prevent overfitting by ensuring that trees do not become overly complex.
- **Max Features ('max_features': 'sqrt')**: Specifies the maximum number of features to consider when looking for the best split at each tree node. In this case, the square root of the total number of features is considered.
- **N_estimators ('n_estimators': 100)**: Indicates the number of trees in the forest. A higher number of trees can lead to a more robust model less prone to overfitting.

After determining the optimal parameters through Grid Search, we retrained the Random Forest model using these parameters and evaluated its performance on the test set. This provided us with an accurate estimate of the model's performance on previously unseen data, allowing us to reliably assess its effectiveness.

In conclusion, parameter optimization through Grid Search played a crucial role in improving the performance of our Random Forest model, enabling us to achieve more accurate and reliable results during the testing phase.

Chapter 3

Results

3.1 Results

3.1.1 Validation Set Results

In this subsection, we present the results obtained by evaluating all three models K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest on the validation dataset. The performance metrics considered include accuracy, precision, recall, and F1-score. Table 3.1 summarizes the results:

Model	Accuracy	Precision	Recall	F1-Score	Class
KNN	0.64	0.61	0.75	0.68	0
KNN	0.64	0.68	0.53	0.60	1
SVM	0.59	0.67	0.35	0.46	0
SVM	0.59	0.56	0.83	0.67	1
Random Forest	0.76	0.73	0.81	0.77	0
Random Forest	0.76	0.79	0.70	0.74	1

Table 3.1. Performance on validation Set

As shown in Table 3.1, the Random Forest model outperforms both KNN and SVM in terms of accuracy, precision, recall, and F1-score on the validation set. Therefore, we selected the Random Forest model for further analysis and evaluation.

3.1.2 Confusion Matrices

Additionally, we generated confusion matrices for each model to visually inspect the performance in more detail. A confusion matrix is a table used to assess the performance of a classification model. It compares the actual and predicted values and is typically organized as follows:

$$\begin{bmatrix} \text{True Negative (TN)} & \text{False Positive (FP)} \\ \text{False Negative (FN)} & \text{True Positive (TP)} \end{bmatrix}$$

Each cell in the matrix represents a particular outcome of the classification. For example: - True Positive (TP): Correctly predicted positive instances. - True Negative (TN): Correctly predicted negative instances. - False Positive (FP): Incorrectly predicted positive instances. - False Negative (FN): Incorrectly predicted negative instances.

Confusion matrices are valuable for understanding the types of errors a model makes and calculating performance metrics such as accuracy, precision, recall, and F1-score. So for the validation test we have:

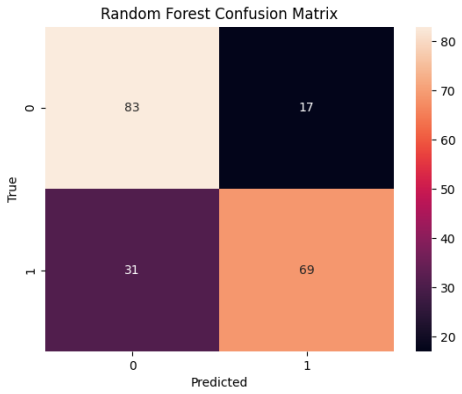


Figure 3.1. Random Forest Confusion Matrix on Validation set

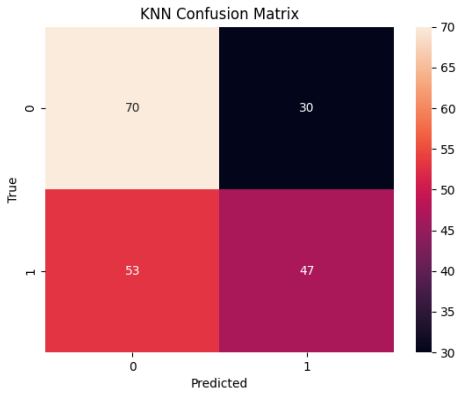


Figure 3.2. KNN Confusion Matrix on Validation Set

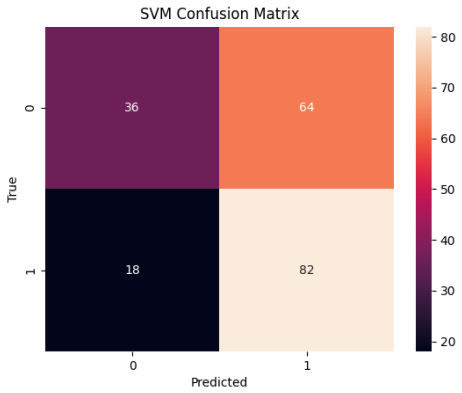


Figure 3.3. SVM Confusion Matrix on Validation Set

3.1.3 Test Set Results

In this subsection, we present the results obtained by evaluating the selected Random Forest model on the test dataset. The performance metrics considered are the same as those used for the validation set. Table 3.2 summarizes the results:

- **Accuracy**: This metric measures the proportion of correctly classified instances out of the total instances in the dataset. It provides an overall assessment of the model's predictive performance.
- **Precision**: Precision represents the ratio of correctly predicted positive observations to the total predicted positives. It measures the model's ability to avoid false positives.
- **Recall**: Recall, also known as sensitivity or true positive rate, quantifies the proportion of actual positive instances that are correctly predicted by the model. It measures the model's ability to capture all positive instances.
- **F1-score**: The F1-score is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall, making it useful for evaluating models when there is an imbalance between classes.

Model (Random Forest)	Accuracy	Precision	Recall	F1-score	class
Random Forest	0.78	0. 75	0.83	0.79	0
	0.78	0.81	0.73	0.77	1

Table 3.2. Performance Metrics on Test Set

Also in this case we can see the confusion Matrix of Random Forest, but finally only on Test Set:

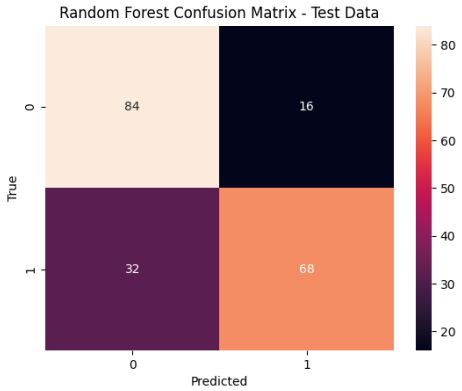


Figure 3.4. Random Forest's Confusion Matrix on Test set

Chapter 4

Conclusion

In conclusion, this study has highlighted the remarkable performance of Random Forest in predicting protein function using machine learning approaches. Through rigorous validation and parameter optimization, Random Forest has proven to be a reliable and accurate model for this task, outperforming K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) in terms of accuracy, precision, recall, and F1-score. The accuracy and reliability of Random Forest in distinguishing between transport and transducer protein functions indicate the potential of this approach in the field of bioinformatics and molecular biology. These findings provide a solid foundation for further investigations and developments in protein function prediction and biological data analysis using machine learning techniques.