The subcellular location of proteins is an important factor that provides knowledge about their function, the interactions they have with other proteins and their role in regulatory processes at the cellular level (Chou and Shen 2008). This information can be used in the fields of molecular cell biology, proteomics, and drug development(Chou 2015). The sequencing of an organism genome has become a routine task (Kanehisa et al. 2016) and as a result, the amount of protein sequences publicly available has been increasing. However, trying to obtain knowledge from a protein sequence is a process that is both time-consuming and costly, leading to an increasingly wide between protein sequence and knowledge about their subcellular location (Chou and Shen 2008). These factors make a computational solution for this task to be very interesting (Lee, Redfern, and Orengo 2007). This is one challenge that can be addressed through different approaches such as comparative genome analysis (Pellegrini et al. 1999), using the structure of the protein(Laskowski, Watson, and Thornton 2005) or extracting information directly from the amino acid sequence(Cedano et al. 2017). For this coursework, we developed a method to classify eukaryotic protein sequence into one of four possible subcellular location: cytosolic, inside the cell but outside of any organelle, secreted, outside of the cell, nuclear, inside the nucleus of the cell and mitochondrial, in the cell’s mitochondria.

Data

Four Fasta files were provided, each one composed of protein sequences of one type of subcellular location. The dataset was extracted those files and includes 9222 protein sequences. The sequences represent the primary structure of protein, and are strings of 24 possible characters ( 20 for each amino acid and 4 for uncertainties). All of the sequences in the dataset are unique, meaning that there are not two homologous. The number of sequences of each class is shown on TABLE.

Method

The model we implemented to solve this task is divided into two stages: feature extraction and prediction. In the first stage, the model extract information from the amino acid sequence representing it as a vector of features that we then feed to the prediction algorithm for the second stage. Python was used to implement this classifier, alongside with the packages Biopython(Cock et al. 2009) and Scikit-learn(Pedregosa FABIANPEDREGOSA et al. 2011)

Feature Extraction

Features extracted by this model can be divided into: amino acid composition, Chou’s pseudo amino composition and physiochemical properties.

Amino acid composition

Amino acid composition was the feature used by earlier approaches to this task (Cedano et al. 2017). It consists on counting the relative frequency of each amino acid in the sequence. Furthermore, we also computed this composition for two local subsequences, the first and last 50 amino acids.

Pseudo amino acid composition

Since amino acid composition only takes into account the amount of each amino acid that is present in the sequence, information about sequence order and sequence length effects is not preserved(Gao et al. 2005). To address this problem Chou introduced the concept of pseudo amino acid composition. In essence, the pseudo amino composition includes the 20 components of the classical amino acid composition but introduces a number of elements that are dependent on sequence ordering(Chou 2001). Each of these elements computes the order-correlation between the ith most contiguous residues(Chou 2001). In our model, we included 7 of these elements.

Physiochemical properties

The physical and chemical properties of each amino acid were used to compute the aromaticity, isoelectric point and the flexibility of the sequence and include these values as features. Moreover, we clustered the amino acids into X groups - …. – and computed the global and two local group compositions.

Design and implementation of the prediction model

The prediction is made based on a voting mechanism that ensembles predictions made by three different classifiers: random forests, logistic regression and gradient boosting.

Logistic Regression

Logistic regression is a special case of a generalized linear model that can predict a discrete outcome, applying a maximum likelihood estimation after transforming the dependent variable through a logit function(Qi, Bar-Joseph, and Klein-Seetharaman 2006). This classifier performs well when the relationships in the data are linear, however, the contrary happens if there’s a complex nonlinear relationship between variables(Abu-Nimeh et al. 2007).

Random Forest

To define Random Forests we need first to define Decision Trees. Abu-Nimeh defines them as “trees where the nonleaf nodes are labeled with attributes, the arcs out of a node are labeled with each of the possible values of the attribute, and the leaves of the tree are labeled with classifications” . According to Qi, this trees are flexible in the sense that can model nonlinear or nonsmooth relationships.

Random Forest is a technique that generates and ensembles a defined number of classification trees with each growing from a bootstrap subset of the original training set (Peters et al. 2007). This leads them to generate an internal unbiased estimate of the generalization error(Abu-Nimeh et al. 2007).

Gradient Boosting

Boosting is the notion that by combining several weak learns it is possible to obtain a good strong learner. Gradient Tree Boosting (GBT) combines several weak learners, usually Decision Trees. GBT aims to minimize a given differentiable loss function. To perform this minimization each weak is added to the model by stages. In each stage, the decision tree the minimizes the loss function the most is added, following the gradient. In this model, we use the implementation of this classifier made by Chen and Guestrin, XGBoost.

Voting Classifier

To combine the predictions of each individual classifier, we implemented the Voting Classifier from the scikit-learn package. By setting the voting type to “soft” this classifier adds up the predictions made by the composing classifier and outputs the argmax of those

Training and Cross-Validation

To train our model, each of the classifiers was trained individually. Although the models and training procedures of all classifiers were already implemented by the scikit-learn package, some hyperparameter tuning was required in order to adapt the algorithms to our dataset. In the case of Logistic Regression we tuned the parameter “C” – inverse of regularization strength, with a lower value meaning a stronger regularization.

For the Random Forests classifier, we studied different values for the number of trees (n\_estimators) and for the maximum number of features that each tree takes into account (max\_features). While for the former a higher number is generally better, the same thing doesn’t apply to the latter which requires a more careful tuning.

The parameters tuned for XGBoost were indentical to those for Random Forests with the