Predicting signal peptides using machine learning

Jonas Andersson

Computer Science and Communications, KTH Royal Institute of Technology, Stockholm, Sweden

* jonand8@kth.se

Abstract

We test three different classifiers against a test set of data to shed light which classifier is best suited for signal peptide prediction. The classifiers are then used on two proteomes to infer signal peptide sequences. The classification pipeline combines N-gram language modeling with a vector space model using TF-IDF term weighting. Increased accuracy is obtained when the model is trained on the general location of the signal peptide chain.

Introduction

The localization of proteins is mostly determined by a biological trafficking system. Proteins are synthesized on cytoplasmic ribosomes and have several possible destinations such as the nucleus, mitochondria, or peroxisomes. Signal peptides direct proteins to their proper destination after synthesis, keeping them from arriving at the wrong cellular location. Signal peptides, short chains of amino acids, act as postal codes for their destination. These sorting signals, a N-terminal signal peptide, are then removed or cleaved off. If the protein does not have this signal peptide then it remains in the cytoplasm. The structure of the signal peptide consists of a positively charged N-region (1-5 residues long), a hydrophobic h-region (7-15 residues long), and finally a polar C-region (3-7 residues long). The cleavage point is in the C-region, but the location varies between proteins [1].

There exist a number of computational methods for predicting these sorting signals using distinctly different approaches. It is useful to locate these signals as it can potentially add insight into protein translocation, how chromosomes are rearranged. Some examples of application domains for signal prediction include drug discovery and gene therapy [1]. Numbering among these computational approaches are neural networks(NNs), support vector machines(SVMs), hidden markov models (HMMs), decision trees, and ensemble learning methods. These algorithms are encompassed in field of computer science called machine learning. Machine learning (ML) algorithms learn and make decisions from data and comprises a broad spectrum of approaches; much of the work done on signal prediction is approached as a classification problem. In classification, a probability distribution is drawn over the classes, and we choose the class that has the highest probability [2].

$$\hat{c} = \arg\max_{c \in C} \Pr(c|x)$$

Materials and methods

Materials

The software used in this project includes:

- Python 3.6
- Biopython: Parsing the data
- Sci-kit learn: Classifier, vectorization, feature selection, and parameter grid search libraries
- Jupyter Notebook: Prototyping

Data Preprocessing

The training data is FASTA-like with each sequence followed by an annotation line. The annotation line marks the location of the n-, h-, and c- regions. The annotation line is removed during preprocessing. The class information and sequence string is stored in a hash ready for modeling.

Data Modeling

The training data is represented using a language model called N-grams. N-gram models are a probabilistic language model which predict the next word or symbol from the previous N-1 words or symbols. Since we are using amino acid residues in place of words, this will instead predict the next residue from the previous N-1 residues.

N-gram models are trained on a corpus which is a computer readable collection of text or speech [2]. The training data is a corpus which contains 2654 examples. The sequence lengths in the corpus range from 16 residues to 4563 residues. With most sequences having a length between 16 and 1000 residues in length as seen in figure 1.

Once the data is counted, it is vectorized into a vector space model where the sequences are converted into vectors of features representing the amino acids that appear in a sequence. The value of each feature is the term weight and is a function of the terms frequency in the sequence.

The collection of sequences characterized as vectors can be viewed as a sparse matrix of weights where $w_{i,j}$ is the weight of the residue N-gram i in sequence j. This matrix is called a term-by-document matrix [2].

The term weighting is not only a function of the frequency but also its inverse frequency. The inverse document frequency, IDF, term weight assigns higher weights to more discriminative terms. Discriminative terms occur in only a few of the sequences.

$$idf_i = \log \frac{N}{n_i}$$

We combine IDF with term frequency, TF, to obtain tf-idf weighting.

$$w_{i,j} = tf_{i,j} \times idf_i$$

TF-IDF prefers words that are frequent in the current sequence but rare overall in the collection [2].

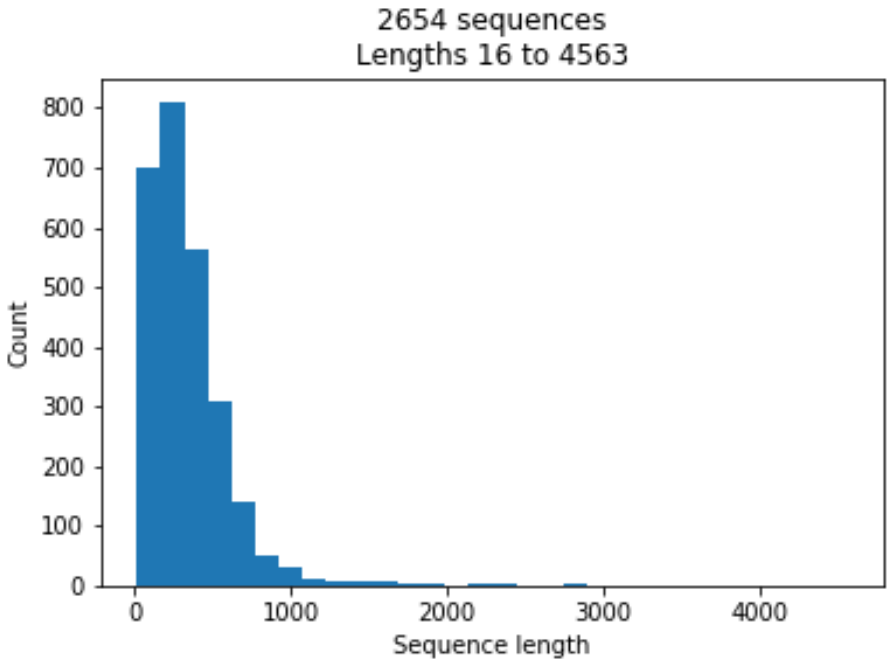

Fig 1. Training Data Length Histogram

Training and Test Sets

The probabilities of the N-gram model come from the corpus it is trained on. The parameters of a statistical model are trained on a set of data and then the model is applied to new unseen data to determine performance. The unseen data set is often called a test set. Using this method we can also evaluate model parameters, such as different orders of N in an N-gram model. Ideally, a models training set should be as large as possible without making the test set unrepresentative. In practice this is typically 80% training, 10% development, and 10% test [2]. This project does not use a development set so the division will be 80% training and 20% test.

Classifiers

Naive Bayes

In general, a classifier can be said to be a function that finds the class C_k that is most probable given the observed data:

$$p(C_k|x_1, \ldots, x_n) \tag{1}$$

Using Bayes Theorem, we get

$$p(C_k|x_1, \ldots, x_n) \propto p(C_k)p(x_1, \ldots, x_n|C_k) \tag{2}$$

which can be expanded to the following using the chain rule of conditional probabilities:

$$p(x_1, \ldots, x_n|C_k) = p(x_1|x_2 \ldots, x_n, C_k) \ldots p(x_n|C_k) \tag{3}$$

The conditional probabilities in equation 3 can be hard to compute. Naive Bayes makes the assumption that the data are independent of each other given the class. We thus get

$$p(x_i|x_{i+1} \ldots, x_n, C_k) = p(x_i|C_k) \tag{4}$$

Equation 3 then simplifies to

$$\begin{gathered} p(x_1, \ldots, x_n|C_k) = p(x_1|C_k) \ldots p(x_n|C_k) \\ \Rightarrow \\ p(C_k|x_1, \ldots, x_n) \propto p(C_k)p(x_1|C_k) \ldots p(x_n|C_k) \end{gathered} \tag{5}$$

Compressing this, the Naive Bayes classifier finds the most likely class $\hat{y}$,

$$\hat{y} = \arg\max_k p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{6}$$

$p(C_k)$ can be estimated by counting the frequencies of the classes in the training set. Similarly, the conditional probabilities $p(x_i|C_k)$ are the frequencies of each residue n-gram in the different classes. To avoid a product of zero if a residue n-gram has never appeared in a class, one can do *smoothing*, which slightly changes all probabilities to avoid zeros.

Naive Bayes is often employed as a good baseline method, yielding results that are sufficiently good for practical use [2]. As a baseline, this classifier is expected to perform the worst, but with at least 70 percent accuracy.

Support Vector Machines

Support Vector Machines are a case of a maximal margin classifier that can draw non-linear decision boundaries. Support vector machines are intended for a binary classification setting, such as the problem posed in this project. SVMs make use of a kernel to transform the input space into a higher dimensional space for the purpose of finding the separating hyperplane with maximum margins. There are a number of different non-linear kernels to use, a popular choice is the radial kernel [5] which takes the form of

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{x'j})^2) \tag{7}$$

When given an observation that has a large euclidean distance from a training observation then the sum $\sum_{j=1}^{p}(x_{ij} - x_{x'j})^2$ will be large, but the equation 7 will be small. This means that it will not affect the classification of points closer to the training data. Support vector machines are considered one of the best 'out of the box' classifiers [5]. In this experiment will do a grid search to find the kernel which performs the best on the held out data set. It is hypothesized that this will be the strongest classifier of the 3 chosen.

Random Forests

Random Forests are a collection of decision tree's. Decision tree's suffer from high variance, because when the training data is split randomly and fit to a decision tree, the results could be different based on the split [5]. However with random forests, each time a split is considered, a random sample of m predictors is chosen instead of the full set of predictors p. A rule of thumb is that the sample $m \approx \sqrt{p}$ [5]. Sampling only a subset of the total predictors mitigates the effect of very strong predictors. In a standard tree these strong predictors would be candidates for the top split because they maximize the information gain. Random forests decorrelate the tree's by only considering a subset of the predictors. If $m = p$ then this is known as bagging.

Stafford Noble's Guidelines

The guidelines given in 'A Quick Guide to Organizing Computational Biology Projects' [3] were adhered to for the most part with some small deviations. The deviations were largely due to time constraints. The spirit of the document was followed.

The project directory structure was adhered to as per Stafford Noble. Source code is in a 'src' folder, the doc folder contains the report, data contains the training and proteome data. The bin folder was excluded in favor of a python driver script using argparse. This could have been easily changed into a 'binary' by adding the file to the workstations PATH and removing the extension, but it was deemed unnecessary for the sake of time.

A lab notebook was kept in the form of a markdown file. The markdown file is the summary of hand written notes and ipython notebooks as well as general brainstorming.

Documenting the source code was done with docstrings. This deviated as the project deadline drew near and errors were encountered. Some functions went through several revisions and updating the docstrings in the intermediate state would have been a waste of time. Code was prototyped in an ipython notebook (included in the project source files under the notebooks directory) and then transferred to a python module. The scripts written could be classifed as 'project specific' in the 4 categories defined in Stafford Noble.

The use of version control was followed nearly to the letter. A .gitignore file was used to ignore generated files such as the temporary files generated by python and LaTeX editors. Data was checked in to version control as a time saving measure. This made it easy to pull and push changes when working on different workstations. Normally a function would be used to pull the data from AWS or some other repository purposed for storing data. For the sake of time, data was checked in.

Results and Discussion

Improving a model lies in understanding where mistakes are made. In a classifier this is done with a confusion matrix (also called a contingency table). A confusion matrix for a 2-way classification task is a 2×2 matrix, where cell x, y is the number of times an item with correct classification as x was classified by the model as y.

Looking at the confusion matrix for models trained and tested on transmembrane data in Table 1 we can see that there are not many examples to work with. The high error rate demonstrates the need for collecting more examples or features surrounding transmembrane signal peptides. The results found in the tables are produced with trigrams and a residue length of 35. The SVM is using a linear kernel with a C value of 10. The random forest is using 20 estimators.

Table 1. Confusion matrix for transmembrane data

Naive Bayes			SVM		Random Forest	
	P	**N**	**P**	**N**	**P**	**N**
PP	47	0	47	0	47	0
PN	12	0	8	4	11	1
Accuracy	79.7%		86.4%		81.4%	

In the confusion matrix for non-transmembrane data found in Table 2 we have more examples to work with. The classifier performs better when examining only non-transmembrane peptides compared to transmembrane only and the whole training set. The SVM classifier does particularly well in this setting.

Table 2. Confusion matrix for non-transmembrane data

	Naive Bayes		SVM		Random Forest	
	P	**N**	**P**	**N**	**P**	**N**
PP	187	31	205	13	180	32
PN	12	243	26	229	38	217
Accuracy	90.9%		91.8%		83.9%	

Table 3. Confusion matrix for transmembrane and non-transmembrane data

	Naive Bayes		SVM		Random Forest	
	P	**N**	**P**	**N**	**P**	**N**
PP	214	52	241	25	213	53
PN	17	248	35	230	45	220
Accuracy	87.0%		88.7%		81.5%	

Classifying Proteomes

The classifiers were ran on two complete Proteomes downloaded from BioMarts Ensembl service [4]. The two proteomes are Choloepus hoffmanni (Sloth) and Gallus gallus (Chicken). A summary of the classifiers signal peptide detection rate is shown in Table 4 for Choloepus hoffmanni and Table 5 for Gallus gallus. The total number of protein sequences for each species can be seen in Table 6. Points of interest are in sorted order of positive predictions by classifier. For both proteomes, Naive Bayes predicted the most signal peptides, SVM with a linear kernel the least, and the Random Forest classifier taking the middle.

Table 4. Choloepus hoffmanni positive prediction frequency and percentage of total proteome containing signal peptides

	Positive Predictions	Percentage
Naive Bayes	3493	28.1%
SVM	2080	16.7%
Random Forest	2603	20.8%

Table 5. Gallus gallus positive prediction frequency and percentage of total proteome containing signal peptides

	Positive Predictions	Percentage
Naive Bayes	10250	33.9%
SVM	5982	19.8%
Random Forest	8018	26.5%

Table 6. Total sequence counts

	Sequence Count
Gallus gallus	30252
Choloepus hoffmanni	12435

Conclusion

Using a residue length of 35 and trigrams proved to have the greatest affect on the accuracy on the test set. Small increases in accuracy were obtained afterwards by tuning the classifier parameters. The biggest surprise was the SVM kernel selected by performing a grid search over a set of parameters. The kernel that performed the best was the 'linear' kernel, or no kernel. The second surprising point was how well Naive Bayes performed on the held out set of data. In all cases the SVM classifier outperformed the the Random Forest classifier while being closely tailed by the Naive Bayes classifier in terms of accuracy. In terms of further improvement, it could be argued that the Random Forest classifier was not tuned well enough to outperform what is considered to be a baseline classifier. Though it is also possible that the No Free Lunch theorem is manifesting itself and making it clear that there is no one-size-fits all classifier, and this could hold true for so-called baseline classifiers as well. In future work, it would be useful to investigate other classifiers such as Nearest Neighbors and a various neural network architectures.

References

1. Ojefua EE. Finding Secretion Signals in Protein Sequences. Masters thesis. University of Tampere; 2009.

2. Jurafsky D. Martin HM. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. London: Pearson Education; 2009

3. Noble WS (2009) A Quick Guide to Organizing Computational Biology Projects. PLoS Comput Biol 5(7). 2009: e1000424. https://doi.org/10.1371/journal.pcbi.1000424

4. Zerbino D, Achuthan P, Akanni W, Amode M, Barrell D, Bhai J et al. Ensembl 2018. Nucleic Acids Research. 2017;46(D1):D754-D761.

5. James G et al. An introduction to statistical learning: with applications in R. Springer Texts in Statistics. 2013. DOI 10.1007/978-1-4614-7138-7 9