Lab Notebook – SHENG HAN

# Project summary:

This project aim at predicting whether a given protein sequence contains a signal peptide or not. The input data were provided by our bioinformatics professor at KTH. Both probabilistic models (Naive Bayes) and Linear models (SVM and logistic regression) are trained to run prediction. Note that predict the exact position of signal peptide and classify given protein sequence into correct regions is out of scope for this introductory project.

## 10-12-2017 – Project initialisation

**Event/findings:** Initial python coding for this project has been done. The initial code aims to parsing raw data (obtained from course Canvas) into python data types. The following step is performed on the raw data to transform into Machine learning ready data.

1. Open and read each data file
2. Parse FASTA format data into python dictionaries
3. Use Count vectoriser to vectorise raw sequence strings into vectors
4. Tested N-grams on cross-validation sets, found 3 to 4 n-gram provide best average accuracy on validation set.
5. Split into train/test set, initial split ration is 0.8/0.2

Using the train and test set, 2 Naïve Bayes models and logistic regression is trained.

The result is stored under /results/10-12-2017. Please refer to the result folder for images

**Errors:**

1. The fasta parser from Bio package does not filter out commenting lines, this results an error in the training data. The solution was build our own FASTA reader.

**Lessons learned:**

1. Double checking the training data is important, invalid input data will produce incorrect results.

**What’s next?**

1. Implement SVM
2. Separate and clean existing python into several modules
3. Improve documentation and comments
4. Write code to visualise outcomes
5. Write automated scripts that train data and generates results
6. Choose two proteomes and write scripts/code to predict on the proteomes

## 12-12-2017 – SVM

**Event/findings:** Build and trained basic neural network and SVM classifiers. I found a simple neuron network performs similar as other none neural network approaches. To keep the scope small, I decided to not use complex models such as deep neural networks.

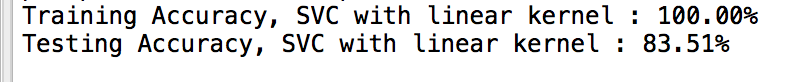
SVM with linear kernel performed the same as logistic regression. (83% on test-set on Non-tm)

**Errors:**

1. RBF kernel does not work well in high dimension; with 3 and 4 n-grams, the input X has a shape of 2362 \* 144374 which produced near random results, shown below, note they are based on non-TM data

**Lessons learned:**

1. Run RBF kernel in lower dimensions, replaced with 1 and 2 n-gram, which produced similar results as linear kernel in 3 and 4 n-gram, shown as below, based on non-TM data.

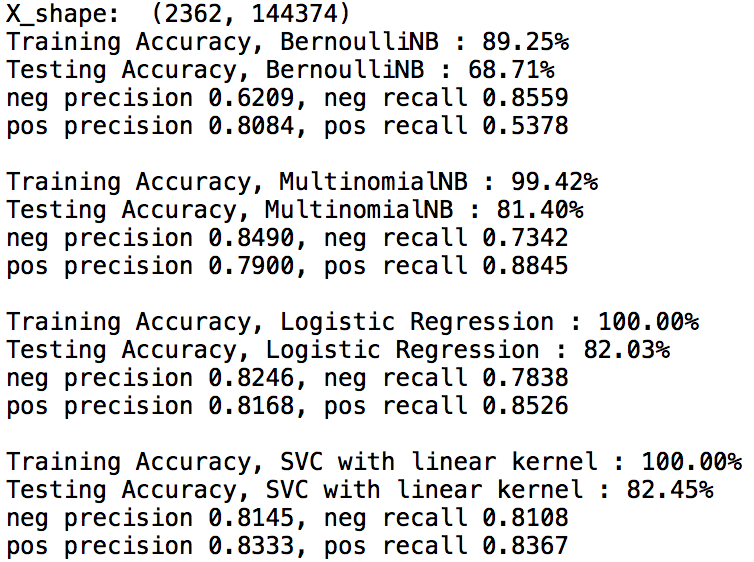


**What’s next?**

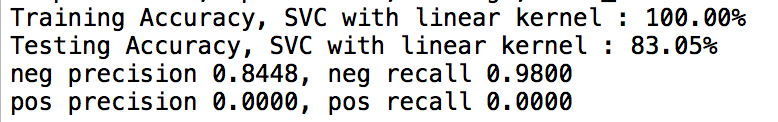
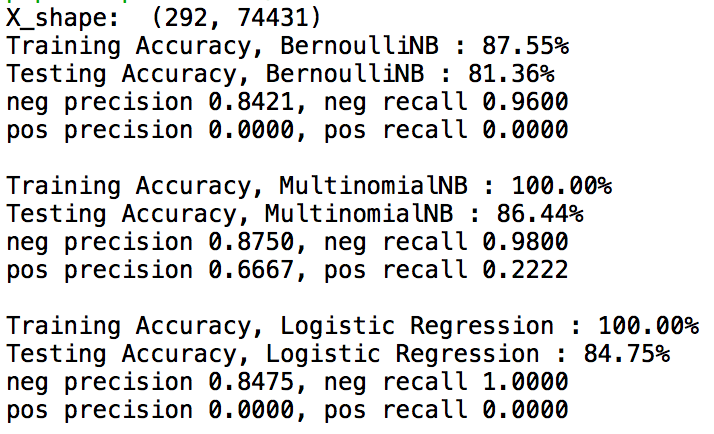
1. Add precision and recall into results
2. Separate and clean existing python into several modules
3. Improve documentation and comments
4. Write code to visualise outcomes
5. Write automated scripts that train data and generates results
6. Choose two proteomes and write scripts/code to predict on the proteomes

## 14-12-2017 – Precision and recall

**Event/findings:** Precision and recall is added as part of matrix to measure the performance.

* **Results of Non-TM samples:** 

Both logistic regression and SVM with learning kernel performed similar with SVM gaining an slide edge in terms of negative recall and positive precision.

* **Results for TM samples:**

All classifiers except multinomial performed poorly on predicting and training on TM samples. Although accuracy is high, this can be accounted by the fact that only 15% of training data (36/233) are positive samples.

**Lessons Learned:**

1. Accuracy is not the final matric in determining the performance of classifiers. The truth table or precision and recall score shows a more accurate reflection on the true performance.

**What’s next?**

1. Separate and clean existing python into several modules
2. Improve documentation and comments
3. Write code to visualise outcomes
4. Write automated scripts that train data and generates results
5. Try training on a mixed TM and non-TM dataset.
6. Choose two proteomes and write scripts/code to predict on the proteomes

## 15-12-2017 – Saving results

**Event/findings:** Add model\_performance\_test and model\_f-score\_test. The model performance test performs model evaluation based on n-fold cross-validation. The result is saved as a bar chart in pdf.

The Model f-score test evaluates f-beta scores, precision and recalls for all 4 models, and construct a table that is then saved in a pdf file automatically.

We found logistic regression performed the best during our model performance test. This result is consistent with f-score test. SVM have a slightly higher f-score results, but it had a greater variance during our performance test thus less reliable.

**Lessons learned:**

* I’ve learnt how to produce charts and tables using matplotlib.
* N-fold cross validation takes significant longer time to finish when n-gram has an upper range above 3. This can take up to 10 minutes.

**What’s next?**

1. Improve documentation and comments
2. Write automated scripts that train data and generates results
3. Choose two proteomes and write scripts and code to predict on the proteomes

## 22-12-2017 – Command line interface

**Event/findings:** Using argparse to process interface with command line. Added comments for the remaining functions.

**Lessons learned:**

* I have increased my confidence in working with argparse module

**What’s next?**

1. Choose two proteomes and write scripts and code to predict on the proteomes

## 23-12-2017 – Prediction on 2 proteomes

**Event/findings:** I decided to use Logistic regression (since it perform the best and easy to train) on 2 proteomes.

* The results of prediction are saved in txt file, formatted as [accession],[1/0] where 1 indicate this accession includes an signal peptide and 0 otherwise.
* The program also allows customise threshold, default to 0.5

**What’s next?**

1. Enjoy Christmas
2. Split into multiple modules
3. Help on the report