Binary Classifier: Was this protein sequence randomly shuffled?

Data Mining (CS-235) course Project by Aleksandr Levchuk
Submitted on March 17, 2011

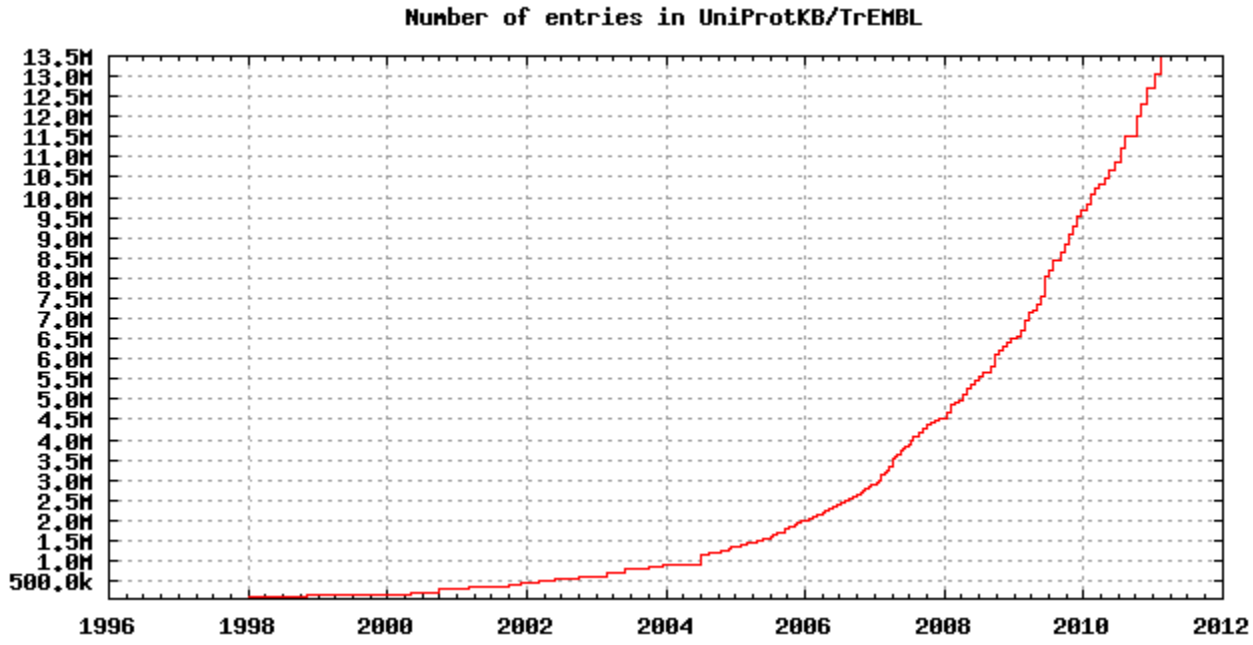
Introduction

How can a computer program tell whether an unknown protein sequence underwent a random rearrangement of all of it's letters? This is a toy problem which I solve for learning purposes. My solution works by looking up all 6-mers of the unknown sequence and adding-up the ratios of overrepresentation of the 6-mers in a database of know protein sequences.

The Data

For training and verification of my computer program (classifier) I used UniProt - the world's largest protein sequence database. TrEMBL makes up the major part of Unitport. The number of protein sequences added to TreEMBL doubles every year.

Figure 1A: As it appears on the TrEMBL February 2011 release stats web page.



I used the Feburary 2011 release. TrEMBL in this release contains 13,499,622 sequences, comprising 4,340,005,282 letters. SwissProt is a highly curated and manually annotated set of protein sequences. In this release SwissProt contains 525,207 sequences and 185,522,689 letters in addition to TrEMBL.[3] I combined TrEMBL and SwissProt to obtain the set that is known as UniProt and used the combined set in my analysis. The total amount of data used in my analysis was 14,024,829 sequences, comprising 4,525,527,971 letters.[4]

Data Availability

The dataset in FASTA format is at http://biocluster.ucr.edu/~alevchuk/cs235-proj/downloaded/. UniProt is distributed under Creative Commons Attribution-NoDerivs License so anyone has the freedom “to copy, distribute, display and make commercial use of these databases [...].” [1, 2]

Analysis Availability

All of the analysis described in this report is freely available in R, Ruby, and Unix shell code. Download or browse at https://github.com/alevchuk/nmercount-classifier

Data Preparation

I concatenated the two FASTA files of TrEMBL and SwissProt into one file and changed the text from multiple-lines-per-sequence to one-line-per-sequence format using a simple convert-fasta-to-tab Ruby script.

One Unix shell command in GNU Bash performs the data preparation:

```
zcat UniProt_2011_02_{trembl,sprot}.fasta.gz | ./convert-fasta-to-tab > UniProt_2011_02.tab
```

Use of Random Numbers

The random number generator in R is made reproducible by running set.seed(1) before any series of randomized operations. There are only two randomized operations in my analysis - data sampling and sequence shuffling. In both cases R’s `sample` function is used. When taking samples from the UniProt data I recorded the line numbers from the original February 8th release.

Data-mining of 10% sample of sequences. Remaining 90% for checking.

To separate the data that will be used to build the Biological-vs-Shuffled classifier, I took a random sample from Uniport. This is done by the 010-choose-random-10-percent-sample-ids R script, and the 020-extract-10-percent-sample Ruby script. The 020 script stores the 10% and the remaining 90% of sequences into two separate files.

Amino acid composition

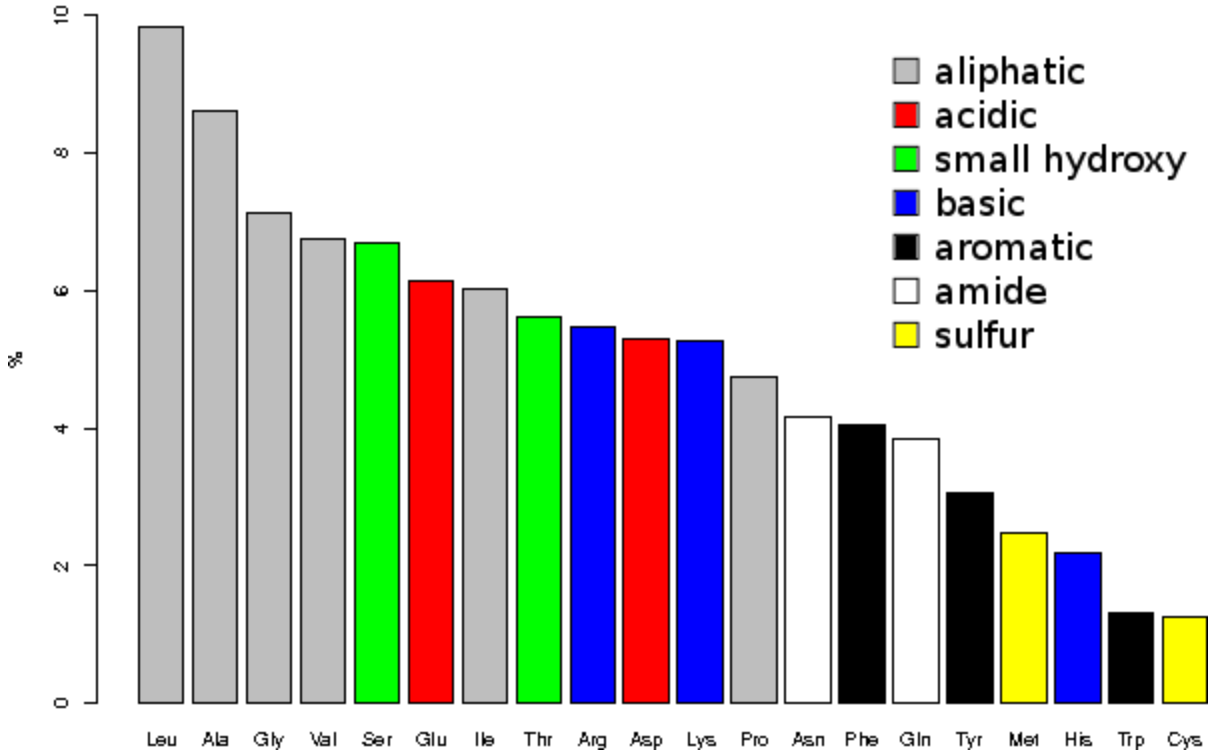
In the simplest case, random sequences will have about the same number of each letter. On the other hand, biological sequences have a skewed and unique distribution of letters frequencies (a.k.a. Amino acid composition). The skewed distribution can be seen in **Table 1** and **Figure 1B**.

Table 1 (Taken from t the TrEMBL February 2011 release stats web page)

Composition in percent for the complete database

Ala (A)	8.61	Gln (Q)	3.84	Leu (L)	9.83	Ser (S)	6.69
Arg (R)	5.46	Glu (E)	6.13	Lys (K)	5.27	Thr (T)	5.62
Asn (N)	4.15	Gly (G)	7.12	Met (M)	2.48	Trp (W)	1.31
Asp (D)	5.30	His (H)	2.19	Phe (F)	4.04	Tyr (Y)	3.05
Cys (C)	1.26	Ile (I)	6.03	Pro (P)	4.73	Val (V)	6.74

Figure 1B (Taken from the TrEMBL February 2011 release stats web page)



To distinguish a biological sequence from a random sequence with a normally distributed alphabet one can simply count how often each letter occurs in the sequences under question. If the distribution is similar to the normal distribution then classify the sequence as Random. If the distribution is more similar to the distribution in **Figure 1B** then classify the sequence as Biological. Such a solution is very easy program. I speculate that it would have an accuracy rate higher than 90% and a false positive rate lower than 10%.

The problem that I'm researching is more challenging then the one that I just described. My research problem is to distinguish a biological sequence from a random sequence with a alphabet distribution that is identical the the biological sequence.

Shuffling the letters in a protein Sequences

I generated a synthetic dataset which had the same distribution of letters frequencies as in UniProt. Specifically, I took every sequence X in UniProt and ran sample(X) in R. This rearranges (shuffles) the letters making the order of letters random but keeping all letters the same as in the original sequence. For every sequences in 14 million of the UniProt sequences I made a corresponding shuffled sequence.

Of the shuffled sequences, 10% were used for building the classifier. The remaining 90% were used to estimate the accuracy of the classifier.

Instead of using the synthetic dataset for building the classifier, a mathematical model for random sequences could have been used. Hoverer, it is not clear to me if the mathematical modeling approach would have maintained the simplicity in the analysis that I was able have. It is also not clear it the approach would significantly improve accuracy of the classifier.

The R script 030-shuffle takes the one-sequence-per-line formatted file with 10% of the UniProt sequences and outputs the same sequences, in the same order, and in the same format but the letters in each sequence are re-arranged.

N-mer Frequency Counting

To understand what is different between the UniProt sequences and shuffled sequences I looked at how many times the combination letters AA, AB, AC, …, ZZ occur in the sequences. In this step I did not differentiate between individual sequences.

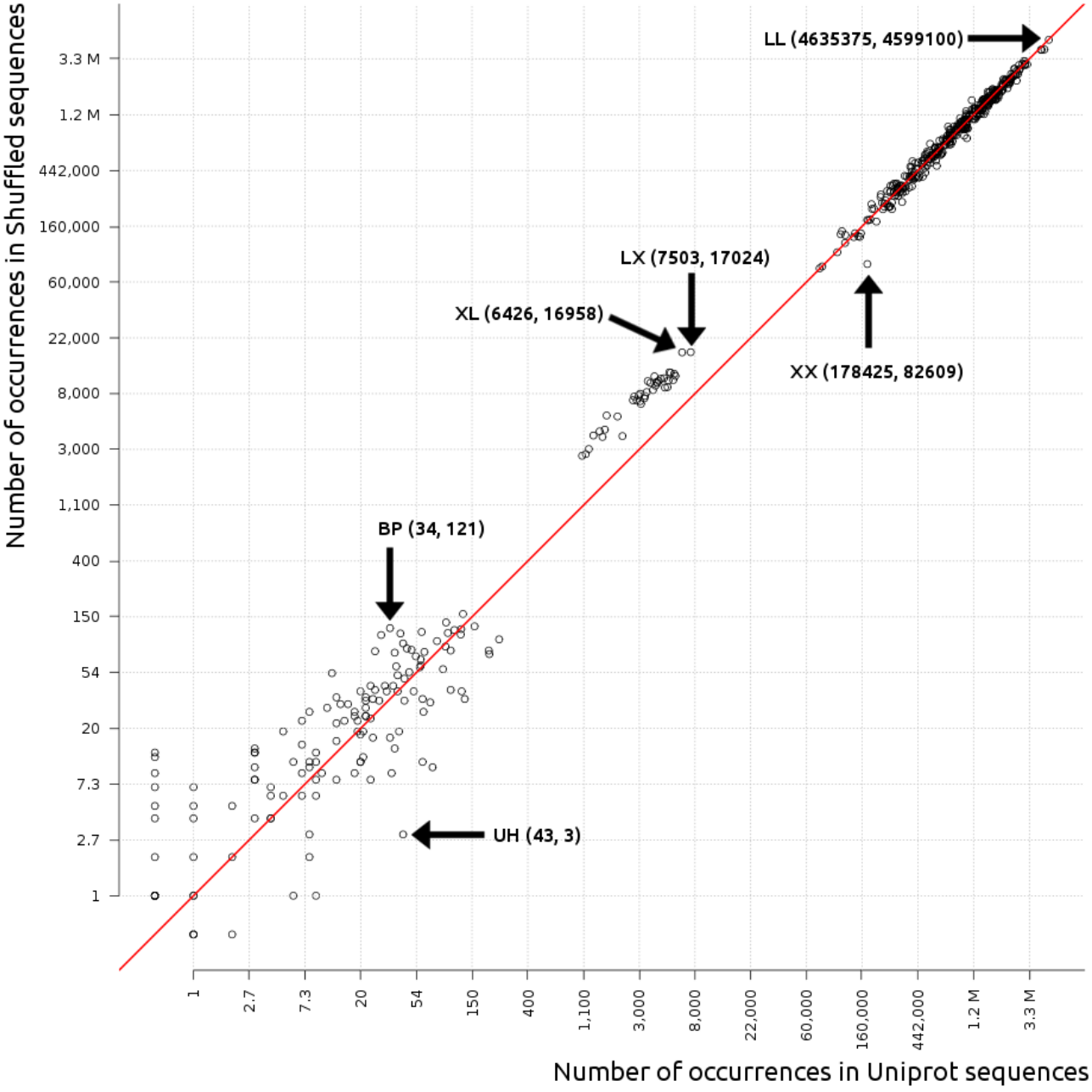
In the simplest way, I slided a 2-letter window from left to right for every sequence, accounting the number of occurrences of the combinations AA, AB, AC, ..., ZZ. As a result having the counts of all the 2-letter combination in all of the 1.4 million the sequences in the 10% UniProt sample. I applied the same analysis to the corresponding set of 1.4 million Shuffled sequences and got a different table of frequency counts. I repeated the analysis for 3-mers, 4-mers, 5-mers, 6-mers, and 7-mers. For each N-mer I got 2 different frequency counts.

The Ruby script 040-get-pattern-frequencies takes one-sequence-per-line formatted file and one argument <runlength> determining which N-mer to do the analysis for. The output looks like this:

```
OG      1
UV      1
…       ...
AA      4280084
LL      4635375
```

Figure 2: Frequency of all 2-mers in UniProt vs. Shuffled sequence pools

Each bubble represents a 2-letter pattern (e.g. “AA”). The x-axis shows how many times the 2-letter pattern occurred in all of the 1.4 million sequences of a 10% UniProt sample. On y-axis are the counts for the same pool of sequences where letters in each sequence were shuffled.



N-mers that are overrepresented and underrepresented in UniProt

In **Figures 2 through 6**, a pattern of letters (an N-mer) placed close to the diagonal is occurring in Uniprot and Shuffled sequences about the same number of times. On the other hand, the further away the N-mer from the diagonal line the greater is the difference between that N-mer occurring in UniProt vs. the Shuffled sets.

Being bellow the diagonal line means the N-mer occurs more frequently in UniProt but less frequently in the shuffled UniProt sequences. Bubbles above the line represent N-mers that occur less in Uniprot but more often in the Suffled set.

Figure 3: Frequency of all 3-mers in UniProt vs. Shuffled sequence pools

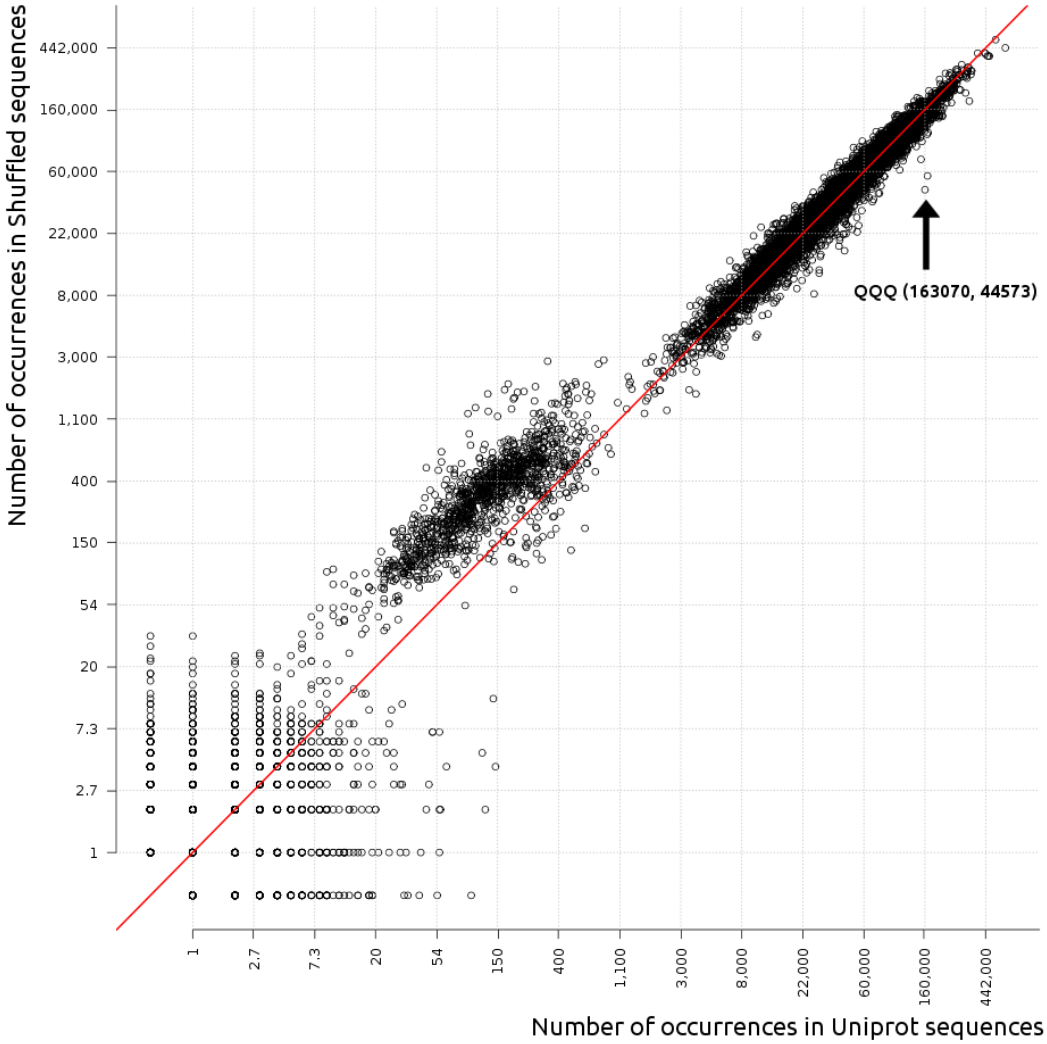
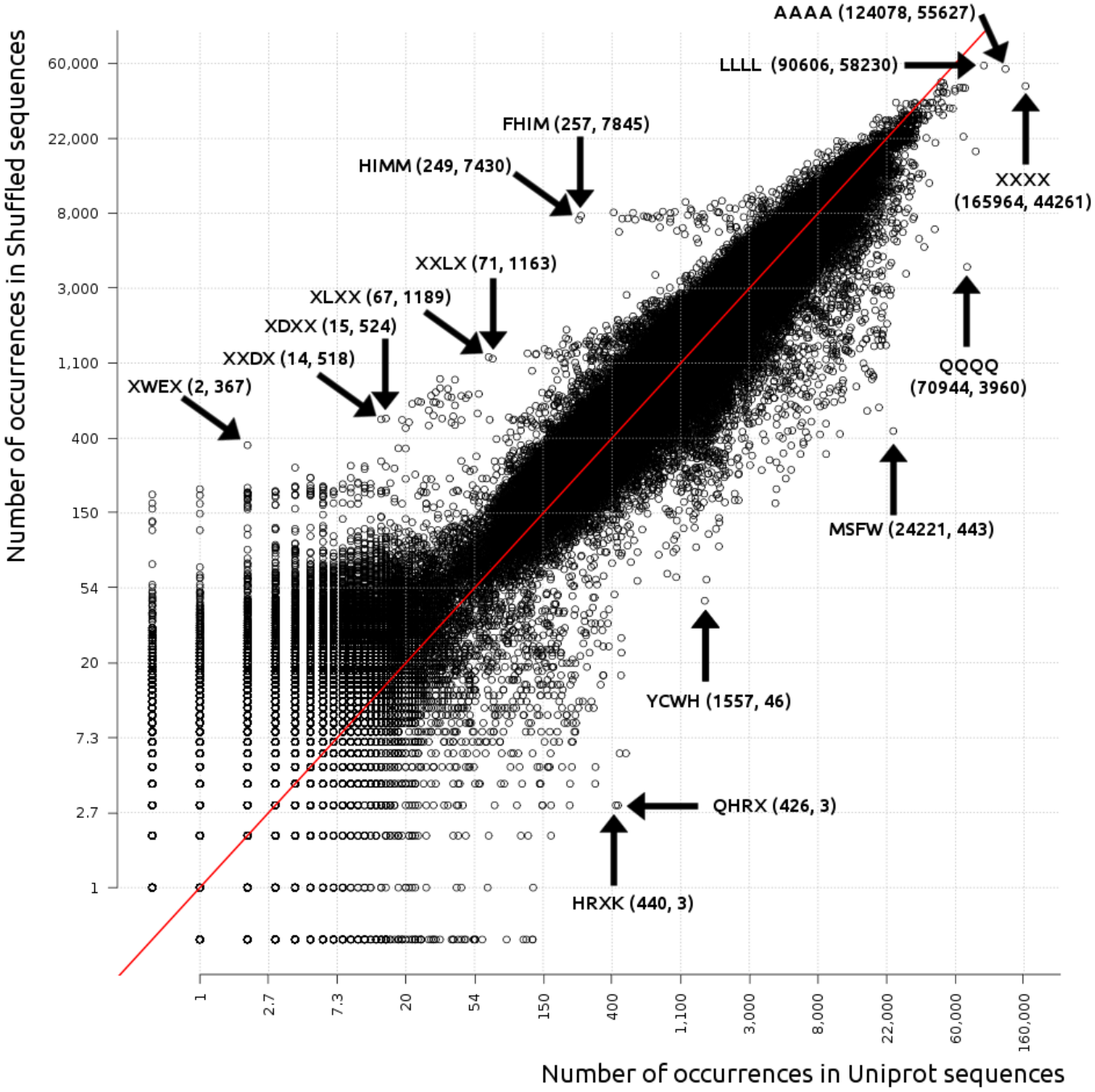
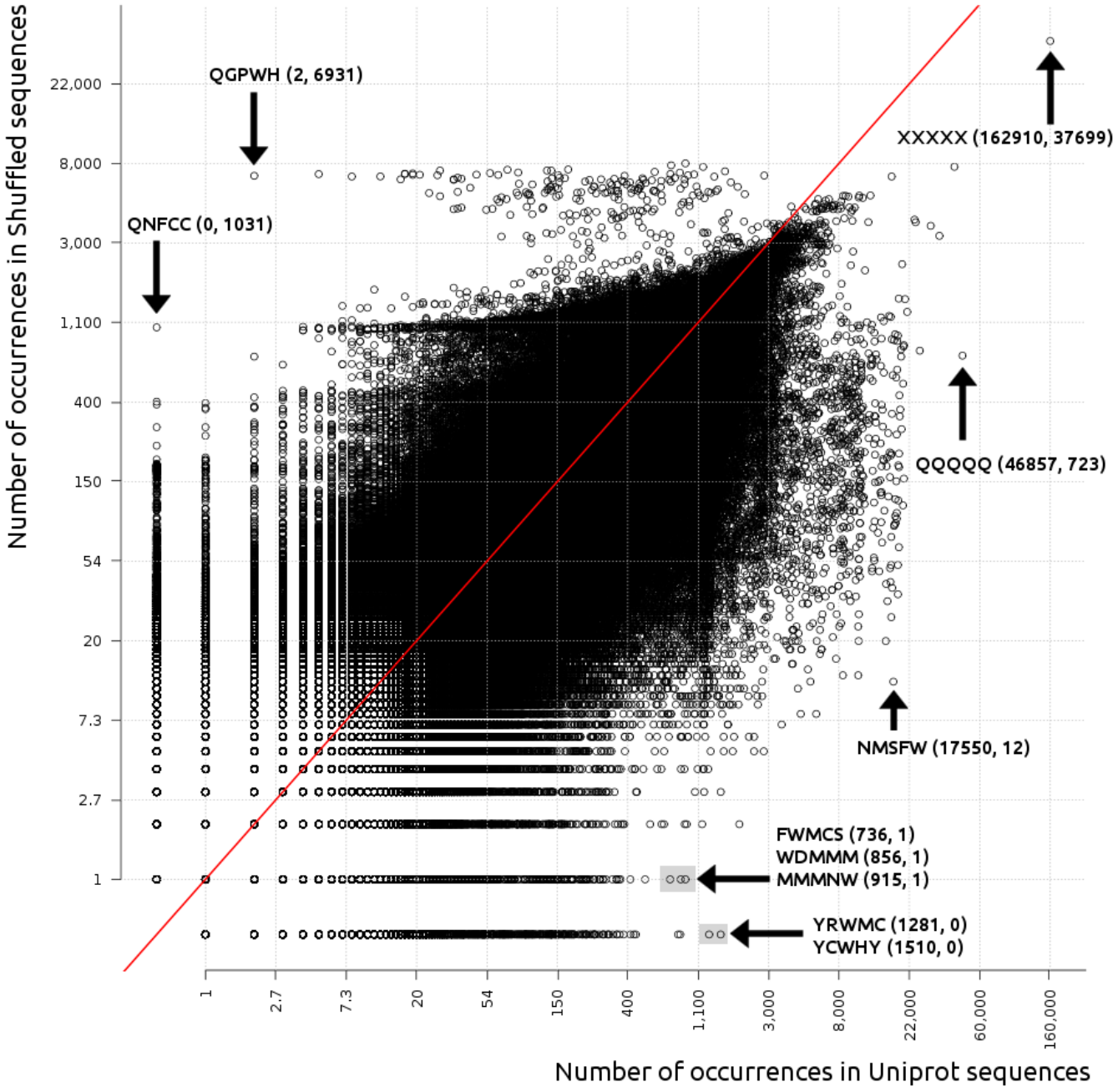


Figure 4: Frequency of all 4-mers in UniProt vs. Shuffled sequence pools



I labeled some of the bubbles showing which particular N-mer the bubble corresponds to and the exact (X, Y) coordinates. The particular bubbles to be labeled were chosen arbitrarily.

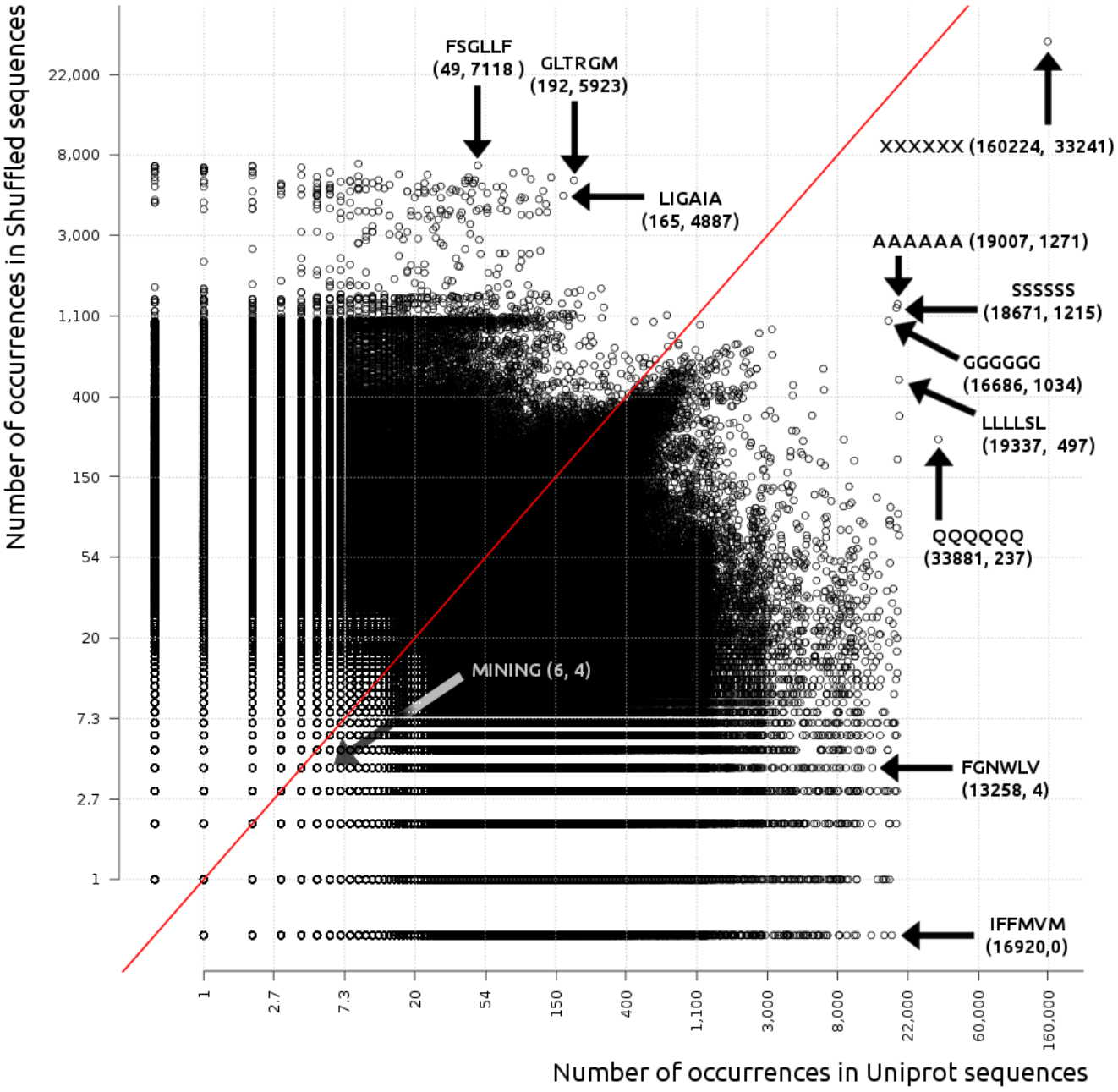
Figure 5: Frequency of all 5-mers in UniProt vs. Shuffled sequence pools



In **Figures 2 through 6**, the straight lines in the the bottom and in the left are visible because the frequency counts are descrete. For example, I was only able to observe N-mers with frequency 1 or 2, but none in between 1 and 2. The lines start to blend into a seemingly continuous space because the the X and Y axis are plotted in log-scale. Log-scale was used to reveal more details.

In **Figures 5 and 6,** however, straight lines of bubbles are visible at approximately Y=1000. This visual pattern, among other things, was unexpected. I did not do analysis to find out the cause.

Figure 6: Frequency of all 6-mers in UniProt vs. Shuffled sequence pools



In **Figure 6** the 6-mer “MINING” is one of 208,959 different 6-mers at point the point (4, 6).

Classifying unknown sequences

The classifier can predict whether an unknown protein sequence underwent a random rearrangement of all of it's letters. It works by looking up all N-mers of the unknown sequence and adding-up the ratios of overrepresentation of the N-mers in a database of know protein sequences. All N-mer are easily obtained by taking a window of size N and sliding it across the entire sequence, one letter at each step.

For each N-mer encountered the overrepresentation ratio is computed by looking up the two frequency tables (Shffled and UniProt) and dividing the larger count by the smaller. If Shffled was the larger one then the ration is negated.

The final sum of ratios will be positive if more frequency ratios are encountered where UniProt was larger. The sum will be negative if more frequency ratios were encountered where Shffled was larger. The sum of ratios will be close to 0 if the two types of frequencies were equal in strength.

The script 060-classify takes the output of 040-get-pattern-frequencies. It takes two frequency tables as arguments: one for the UniProt frequencies and one for the Shuffle frequencies. Then it takes a list of unknown sequences as input. As sequences are read, the classifier outputs one number per each sequence. This number is the sum of ratios as described above.

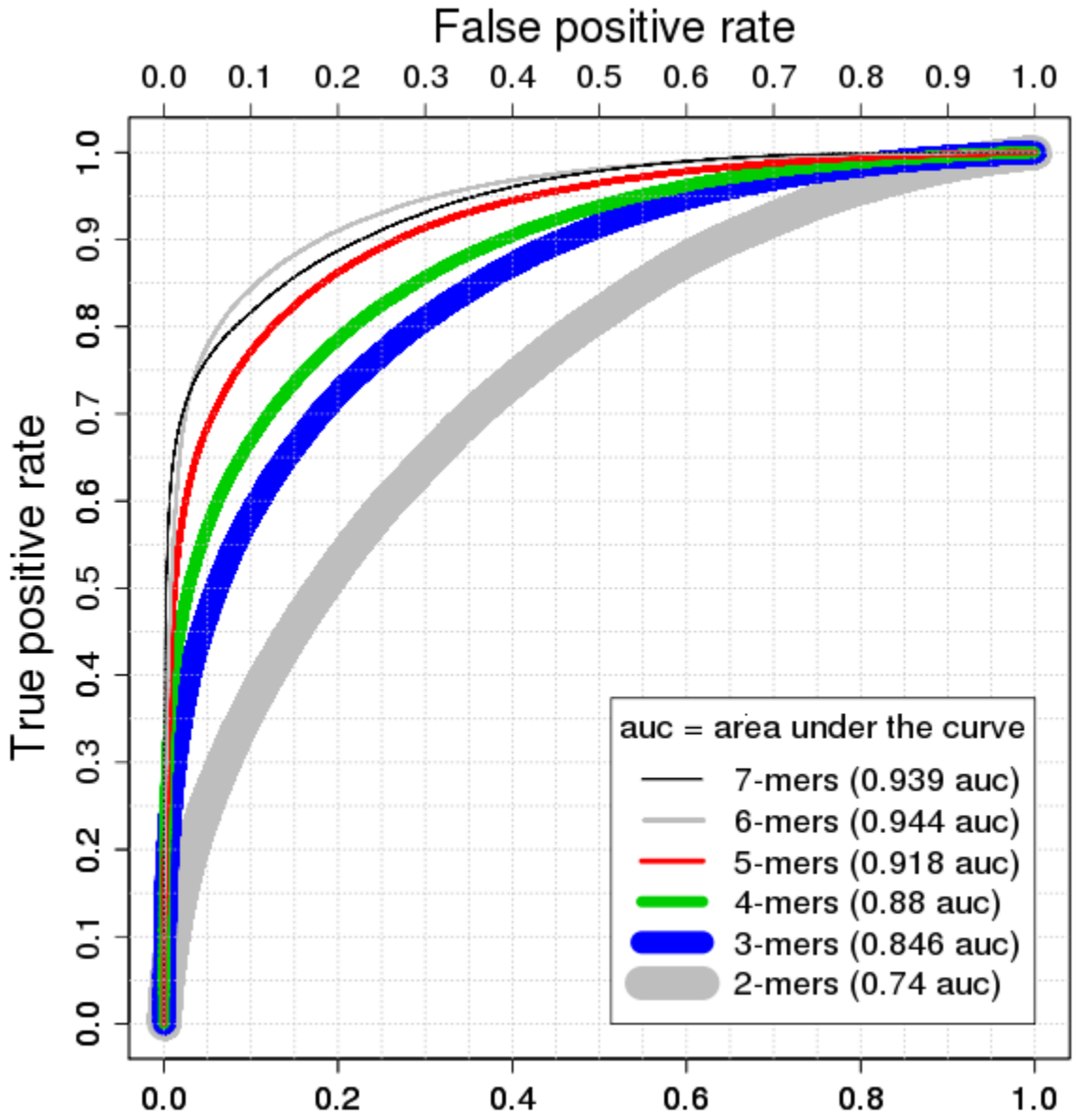
Results

The simplest way to get the answers from the classifier is to look at the sign of the output ratios. If the number is positive then the classifier calms the protein sequence Not Shuffled. If the ratio is negative then the calm for the sequence is Shuffled.

Larger absolute values of the ratios mean the higher confidence of the classifier. So, instead of looking at signs of ratios, I can set a threshold. For example, ratios above -10.0, stands for "Not Shuffled" and ration below -10.0 stand for "Shuffled". It this example, "Not Shuffled" answers will occur more frequently then in the positive/negative classifier. Changing the threshold will change the balance between true-positive rate (correct "Not shuffled" answer) and the false-positive rate of the answers (incorrect "Not shuffled" answer).

I plotted the true-positive rate against the false-positive rate of my classifier. The R script 070-plot-roc takes the output of 060-classify as input. The script uses ratios and the information of which answers are correct to calculate the accuracy rates as a function of different thresholds.

Figure 7: Receiver Operating Characteristic (ROC) plots of 7 different N-mers for the biological vs. shuffled sequence classifier



The above ROC curve was build using 1% of the 90% of the Uniprot sequences. These are the sequences chosen randomly for verification and never been seen by the classifier. Initially, I used only 1% of them for speed purposes. Later, I ran the analysis on the entire set of the verification sequences (90% of UniProt). Surprisingly the ROC results came out almost identical:

7-mers (0.939 auc) I did not have time to try 5-mers and 6-mers.	4-mers (0.881 auc) 3-mers (0.848 auc) 2-mers (0.742 auc)

Conclusions

When increasing the N-mer size from 2 to 6 the simple N-mer counting classifier's performance improved dramatically.

My simple N-mer counting classification method works best for 6-mers. For the results that produced the ROC in **Figure 7**, at threshold -37.0 the 6-mer classifier achieves 87% true-positive rate while having a 13% false-positive rate. At threshold -54.0 the 6-mer classifier achieves 90% true-positive rate while having a 18% false-positive rate.

The 7-mer classifier will perform worse for most thresholds than the 6-mer classifier with the exception of high thresholds.

The simple classifier has worked well. I tried improving the classifier by considering only the top 10 Uniprot vs Shuffled ratios for a given sequence. I reasoned that the largest Uniprot vs Shuffled ratios are significant and all other ratios are just noise. I was wrong, the ROC curve for the top 10 classifier came out very weak. It seems that, even though the small Uniprot vs Shuffled ratios provide only small evidence individually - when added up, the evidence of the truth grows strong.

There were a few computational challenges due to the size to the data set. All the approaches that I've taken and the solutions are available in the version-controlled source code which is referenced in the **Analysis Availability** section.

Further work

It would be great to try a classifier this combines the evidence of all 2,3,4,5,6, and 7-mers.

Instead of looking at consecutive runs of N letters I tried various spaced seed for 2, 3, and 4-mers but the ROC curves came out very weak. It is still not clear, hovever, if the accuracy can be improved by using spaced seeds for 6-mers and 7-mers.

It would be also great to try to improve the classifier by employing an alphabet reductions. The letters can be combined according to a similarity matrix BLOSUM or amino acid physicochemical properties. We may discover N-mer overrepresentation which is more meaningful.

References

[1] http://www.UniProt.org/help/license
[2] https://creativecommons.org/weblog/entry/5778
[3] http://biocluster.ucr.edu/~alevchuk/cs235-proj/downloaded/uniprot_2011_02_sprot_stats/
[4] http://biocluster.ucr.edu/~alevchuk/cs235-proj/downloaded/uniprot_2011_02_trembl_stats/