Zeke Elkins

Final Project Paper

**Introduction**

The data preparation for this project was very extensive. The data itself was originally in .pdb format, which is a generalized format used by cell and biochemical biologists to record information about proteins, including their genetic makeup. The format name, .pdb, stands for “protein data bank.” The issue is, extracting data from this format can be extremely painful and difficult. The broad area of bioinformatics’ main focus is to make the vast amount of information more easily accessible. The area has a long way to go, however.

**Methods**

The .pdb file I obtained is for the protein leptin, a protein that occurs in humans. Its function is to inhibit the feeling of hunger in order to regulate energy balance. My research colleagues Shannon White and Paige Diamond were kind enough to supply me with the file, which I copied into a .txt file named Leptin.txt. This project was mainly inspired by my research project under Professor Timothy Urness and Dr. Matthew Zwier. However, the code I wrote was mainly written by me, with some collaboration on the earlier aspects with Kayla Huff. The main collaboration between Kayla and I was in finding the locations of the true splice sites and arranging them in a list, which was harder than we thought it would be. In addition, Paige Diamond helped us with the initial code that extracted the DNA sequence from the .txt file. I coded everything on my own after that.

Data Preparation

For this particular problem, my goal was to use machine learning to accurately identify strings of 60 DNA base pairs as holding the location of a splice site or not. Splice sites are important because they determine which portion of the DNA sequence is translated into the protein. There are several basic rules for identifying splice sites, but these rules don’t hold up to intense scrutiny, as correct splice sites don’t really follow these rules all of the time. Therefore, it takes a well-trained biologist or graduate student to identify splice sites by eye. As such, it’s extremely valuable to create a program that correctly identifies such sites.

In order to do this, I first had to develop a way to format the data so that I could run it through a machine-learning algorithm. First, I focused on extracting the known splice sites and their respective Open Reading Frames (ORFs), which is the precise sequence of DNA that is translated into a protein. However, after reading some literature, mainly an article written by Zhang et al. (2006), I decided to convert the DNA letters into corresponding numbers. This is known as sparse encoding, and it allows sequences of DNA to be linearly separable. For my data, the DNA base letter A corresponds with 1000, C with 0100, T with 0010, and G with 0001. After converting the base pairs to corresponding codes, I wrote each ORF (comprised of 60 encoded base pairs) to a .csv file. In addition, I added a second column, labeled “orf,” which identified the corresponding sequence as a true ORF (value of 1) or not (value of 0). The true ORFs were labeled as 1.

According to my research colleagues, most splice sites start after the occurrence of ‘AG’ in the DNA base pair sequence. I used a regular expression to identify the location of every ‘AG’ in the DNA sequence. From that information, I then gathered the DNA base pairs extending 30 base pairs on either side of that location, resulting in a false ORF of 60 base pairs. I then converted the base pairs to their respective codes using the sparse encoding method I described above, and added them to the .csv file, labeling them with a 0 since they’re false splice sites. When an “AG” occurred within the first 30 base pairs of the extracted DNA sequence, or within the last 30 base pairs of the extracted DNA sequence, I extended the number of base pairs on one side of “AG” to cover the lack of base pairs on the ending or starting side of the DNA sequence. So if an AG occurred in location 15, I would take the first 15 base pairs, and then 45 more in the opposite direction for a total of 60 base pairs. My biology colleagues say that these potential splice sites are definitely not splice sites, because splice sites don’t normally occur at the beginning or ending of the DNA sequence.

**Machine-Learning Algorithm and Results**

After extracting the entire necessary DNA and encoding it appropriately, I finally turned to the machine algorithms. I extracted the information from the newly made .csv file and split it into a testing set and a training set. Then, I passed it through a Linear SVC machine-learning algorithm using the encoded sequence as the predictor and the ORF binary column as the predicted outcome. I computed the average accuracy of the machine-learning algorithm over the course of 100 trials. The average accuracy score the algorithm received was 99.5%. I was supremely impressed with this score, but then I thought of a potential reason the accuracy was so extremely high: most of the data in my dataset was a false splice site. The leptin protein sequence only contains 6 known splice sites. As a result, the majority of the encoded sequences were not splice sites – about eighteen thousand of them.

**Discussion**

Sadly, due to the extreme amount of data preparation, I was not able to expand upon my results. Going forward, I would like to do several things. First off, I would like to introduce new unknown encoded DNA sequences to the learning algorithm in order to test its ability to accurately predict novel data from other similar proteins. The paper I read by Zhang et al. (2006) discussed implementing a custom-made Naïve Bayes kernel for a Support Vector Machine. I would like to attempt my own custom-made Naïve Bayes kernel. I also want to try using either a deep-text machine learning algorithm or a Multinomial Naïve Bayes algorithm on the non-encoded DNA sequences. However, I’ve read that MNB algorithms do not work as well because building up a dictionary on four random repeating base pairs is extremely convoluted and not very efficient. Finally, I would like to beef up the current training and testing data sets to include more correctly identified ORF sites that may potentially bring down the accuracy scores of the machine-learning algorithms due to adding more variability amongst the current data (making it less linearly separable).

Finally, the machine-learning algorithm could potentially have only guessed that every test example was not a splice site, and it still could predict the outcome accurately with percentages in the 90s, since 100 – (6/18000) is roughly 99.9996. However, since the percentage I got was 99.5%, I assume that the algorithm did indeed predict some examples were splice sites. To resolve this issue, I need more positive examples.

**References**

http://cs.olemiss.edu/~ychen/publications/journal/zhang\_esa06.pdf