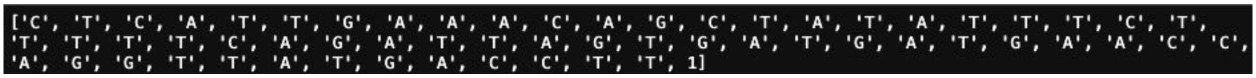
I found this assignment slightly trickier than the last. There was no clear cut way to preprocess, and it took me a number of different attempts to figure out how to preprocess effectively.

My first thought was to analyze the existence of different sequences, and sort of do NLP on that, but this did not prove very effective. I thought that the emergent patterns would arise more naturally this way, but I suppose they did not.

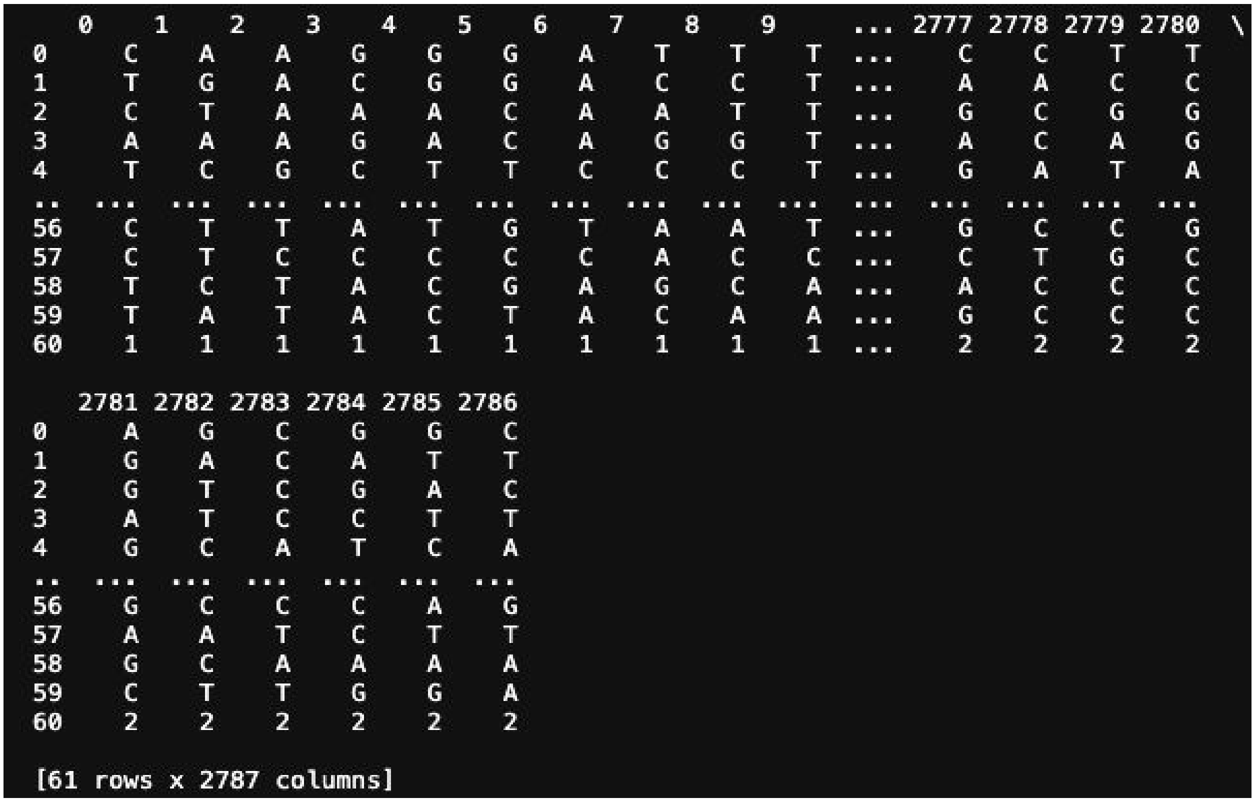
What I eventually ended up doing was breaking it up by letter and turning each nucleotide into a categorical binary variable.

My first pass was to read through an entry, and grab all the letters and the group if it was the training data set. Here is what the first entry looked like at this point.



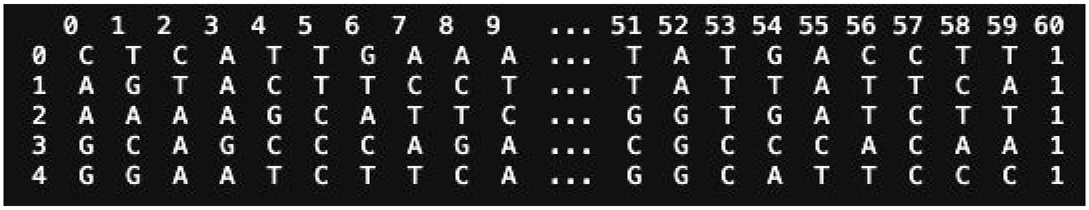
```
['C', 'T', 'C', 'A', 'T', 'T', 'G', 'A', 'A', 'A', 'C', 'A', 'G', 'C', 'T', 'A', 'T', 'A', 'T', 'T', 'T', 'C', 'T',
'T', 'T', 'T', 'T', 'C', 'A', 'G', 'A', 'T', 'T', 'A', 'G', 'T', 'G', 'A', 'T', 'G', 'A', 'T', 'G', 'A', 'A', 'C', 'C',
'A', 'G', 'G', 'T', 'T', 'A', 'T', 'G', 'A', 'C', 'C', 'T', 'T', 1]
```

I then recast it as a Pandas dataframe.



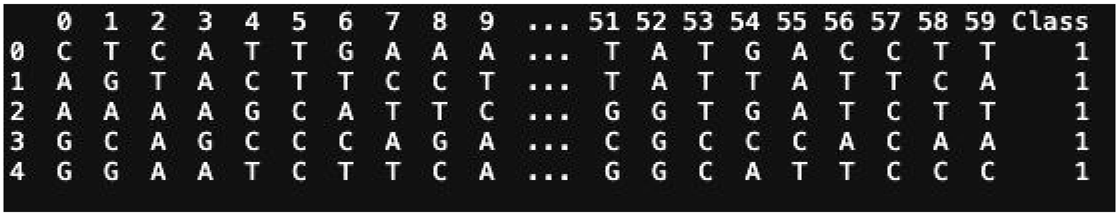
```
     0    1    2    3    4    5    6    7    8    9   ... 2777 2778 2779 2780  \
0    C    A    A    G    G    G    A    T    T    T  ...    C    C    T    T
1    T    G    A    C    G    G    A    C    C    T  ...    A    A    C    C
2    C    T    A    A    A    C    A    A    T    T  ...    G    C    G    G
3    A    A    A    G    A    C    A    G    G    T  ...    A    C    A    G
4    T    C    G    C    T    T    C    C    C    T  ...    G    A    T    A
..  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
56   C    T    T    A    T    G    T    A    A    T  ...    G    C    C    G
57   C    T    C    C    C    C    C    A    C    C  ...    C    T    G    C
58   T    C    T    A    C    G    A    G    C    A  ...    A    C    C    C
59   T    A    T    A    C    T    A    C    A    A  ...    G    C    C    C
60   1    1    1    1    1    1    1    1    1    1  ...    2    2    2    2

   2781 2782 2783 2784 2785 2786
0     A    G    C    G    G    C
1     G    A    C    A    T    T
2     G    T    C    G    A    C
3     A    T    C    C    T    T
4     G    C    A    T    C    A
..  ...  ...  ...  ...  ...  ...
56    G    C    C    C    A    G
57    A    A    T    C    T    T
58    G    C    A    A    A    A
59    C    T    T    G    G    A
60    2    2    2    2    2    2

[61 rows x 2787 columns]
```

This is the wrong way, though, so I transposed it.



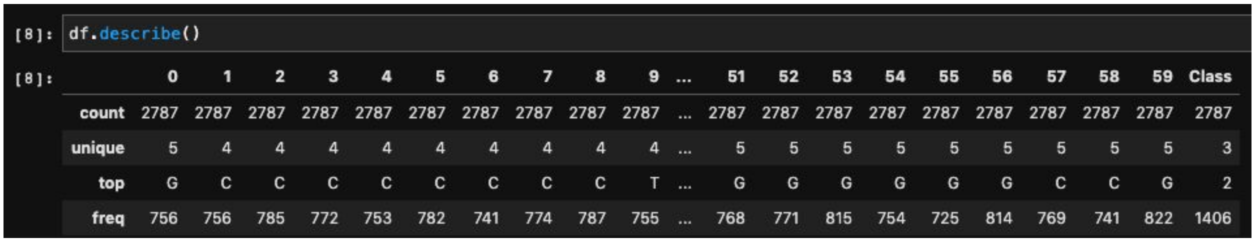
```
   0  1  2  3  4  5  6  7  8  9   ... 51 52 53 54 55 56 57 58 59 60
0  C  T  C  A  T  T  G  A  A  A   ...  T  A  T  G  A  C  C  T  T  1
1  A  G  T  A  C  T  T  C  C  T   ...  T  A  T  T  A  T  T  C  A  1
2  A  A  A  A  G  C  A  T  T  C   ...  G  G  T  G  A  T  C  T  T  1
3  G  C  A  G  C  C  C  A  G  A   ...  C  G  C  C  C  A  C  A  A  1
4  G  G  A  A  T  C  T  T  C  A   ...  G  G  C  A  T  T  C  C  C  1
```

I then renamed the last column to Class if it was the training set, just for convenience.



```
  0  1  2  3  4  5  6  7  8  9  ... 51 52 53 54 55 56 57 58 59 Class
0 C  T  C  A  T  T  G  A  A  A  ...  T  A  T  G  A  C  C  T  T     1
1 A  G  T  A  C  T  T  C  C  T  ...  T  A  T  T  A  T  T  C  A     1
2 A  A  A  A  G  C  A  T  T  C  ...  G  G  T  G  A  T  C  T  T     1
3 G  C  A  G  C  C  C  A  G  A  ...  C  G  C  C  C  A  C  A  A     1
4 G  G  A  A  T  C  T  T  C  A  ...  G  G  C  A  T  T  C  C  C     1
```

Then, I tried to look deeper into the data to see if this would be sufficient for pre-processing.

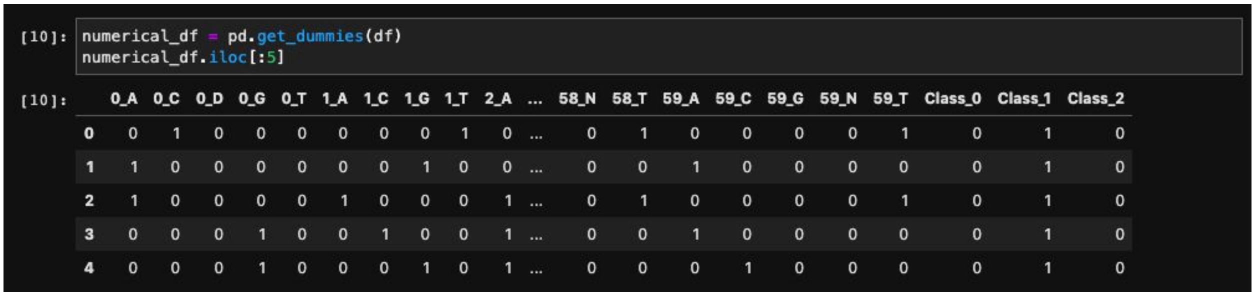


```
[8]: df.describe()
```

[8]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	57	58	59	Class
count	2787	2787	2787	2787	2787	2787	2787	2787	2787	2787	...	2787	2787	2787	2787	2787	2787	2787	2787	2787	2787
unique	5	4	4	4	4	4	4	4	4	4	...	5	5	5	5	5	5	5	5	5	3
top	G	C	C	C	C	C	C	C	C	T	...	G	G	G	G	G	G	C	C	G	2
freq	756	756	785	772	753	782	741	774	787	755	...	768	771	815	754	725	814	769	741	822	1406

However, these statistics are very cursory and don't provide much information. As such, I had Pandas generate dummy variables so that I could more easily standardize what letter is in each position, etc.



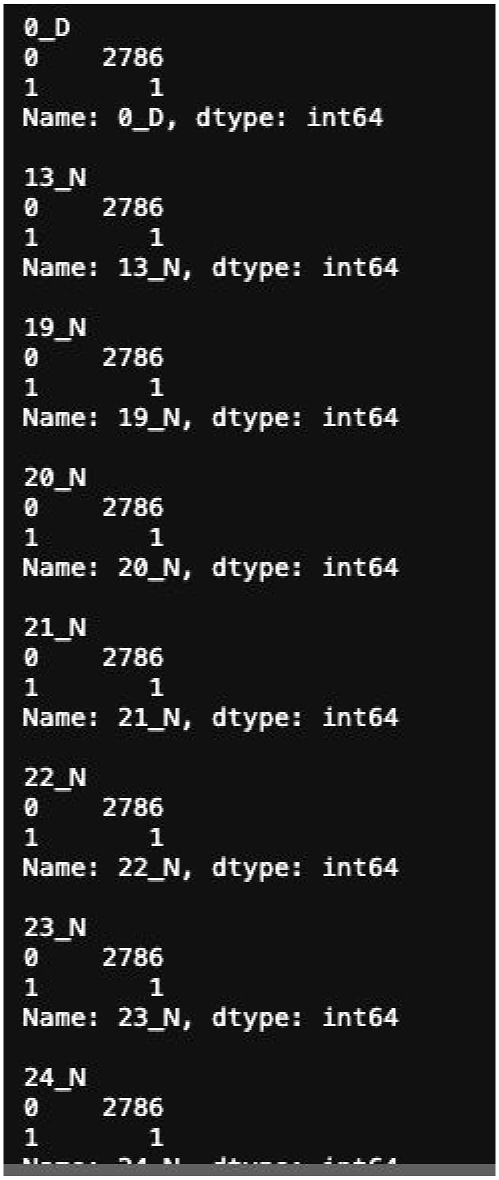
```
[10]: numerical_df = pd.get_dummies(df)
      numerical_df.iloc[:5]
```

[10]:

	0_A	0_C	0_D	0_G	0_T	1_A	1_C	1_G	1_T	2_A	...	58_N	58_T	59_A	59_C	59_G	59_N	59_T	Class_0	Class_1	Class_2
0	0	1	0	0	0	0	0	0	1	0	...	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	0	0	0	1	0	0	...	0	0	1	0	0	0	0	0	1	0
2	1	0	0	0	0	1	0	0	0	1	...	0	1	0	0	0	0	1	0	1	0
3	0	0	0	1	0	0	1	0	0	1	...	0	0	1	0	0	0	0	0	1	0
4	0	0	0	1	0	0	0	1	0	1	...	0	0	0	1	0	0	0	0	1	0

In the process of dummy generation, it split up the class column, so I reconsolidated it and renamed it back to class.

Finally, upon further inspection there were more columns than there should have been. There should only be four times the number of nucleotides. As such, I removed all outliers. When I looked into them, each letter was only included once, so I am going to assume that it is not an immense amount of potential data I could be throwing out.



```
0_D
0    2786
1       1
Name: 0_D, dtype: int64

13_N
0    2786
1       1
Name: 13_N, dtype: int64

19_N
0    2786
1       1
Name: 19_N, dtype: int64

20_N
0    2786
1       1
Name: 20_N, dtype: int64

21_N
0    2786
1       1
Name: 21_N, dtype: int64

22_N
0    2786
1       1
Name: 22_N, dtype: int64

23_N
0    2786
1       1
Name: 23_N, dtype: int64

24_N
0    2786
1       1
Name: 24_N, dtype: int64
```

With this, it was time to actually pick a classifier.

After doing some reading online and going back through the lecture notes, I picked SVM with an RBF kernel as my classifier. I didn’t have time to do a ton of hyperparameter tuning with other models, but it did very well in my testing, and it seemed like a good one to use for this occasion because I wasn’t sure how linear the data would be. The code I used to test different models is included in my submission.

```
Seed = 1

K Nearest Neighbors: 0.761722 (0.021161)
Test--  Nearest Neighbors :  0.7560975609756098

Decision Tree: 0.941627 (0.012803)
Test--  Decision Tree :  0.9368723098995696

Random Forest: 0.509091 (0.024038)
Test--  Random Forest :  0.5265423242467718

Neural Net: 0.944019 (0.015585)
Test--  Neural Net :  0.9540889526542324

AdaBoost: 0.938278 (0.014266)
Test--  AdaBoost :  0.9354375896700143

Naive Bayes: 0.745933 (0.063744)
Test--  Naive Bayes :  0.7216642754662841

SVM Linear: 0.917225 (0.010273)
Test--  SVM Linear :  0.9139167862266858

SVM RBF: 0.958852 (0.012329)
Test--  SVM RBF :  0.9670014347202296
```

Some of the classifiers I used were not covered in class, so I did not include them with my final results. I just experimented with them out of curiosity.

One technique I think I could use in the future to improve the accuracy of my techniques is blending with multiple models that already have >.90% accuracy with 10-fold cross-validation. If I use three ones and two of them agree on a value, I take the one that has two that agree, or something of that sort.