Machine Learning Engineer Nanodegree Capstone Project and First Kaggle Competition

September 29, 2017

1 Sponsor Memorial Sloan Kettering Cancer Center (MSKCC)

2 The First Step in Oncogenicity Determination - Predict the effect of Genetic Variants on Protein Function form Vast Text-Based Clinical Literature to enable Personalized Medicine

2.0.1 Bill Kapsalis

2.1 I. Definition

2.1.1 Project Overview

Which of the thousands of mutations in a cancer tumor contribute to the tumor growth. This project and Kaggle competition will uses machine learning techniques to evaluate vast amounts of an expert-annotated knowledge base of information about each mutation to determine which of the thousands of mutations cause tumor growth.(1)

My personal motivation to pursue this kaggle challenge is I have a passion for machine learning and I have lost many family members to cancer. Also I have an Masters in Bioinformatics with minimal experience. With the machine learning skills I learned from Udacity this Kaggle challenge may have rekindled my passion away from robotics, education and back toward biology. Only time will tell. The first path is clear, machine learning/deep learning.

2.1.2 Problem Statement

This challenge is a current Kaggle competition sponsored by Memorial Sloan Kettering Cancer Center (MSKCC). When a tumor is sequenced thousand of mutations are detected. To determine the best course of action the few mutations responsible for tumor growth(Drivers) need to be determined. The problem is a clinical pathologist must manually classify each mutation by reading the vast test based clinical literature to determine which mutations cause tumor growth. This is a time consuming process. This valuable information is used to personalize the treatment for the best results.

The first step to determine which mutations cause tumor growth is to determine what the effect of the mutation is on the protein encoded by the gene. Another words, what is the effect on the protein function caused by the mutation?

The categories in this Kaggle challenge were given as generic categories 1 through 9. After researching I found these categories correspond to 9 resulting gene mutation effects on the encoded proteins:

- (1) Likely Loss-of-function
- (2) Likely Gain-of-function
- (3) Neutral
- (4) Loss-of-function
- (5) Likely Neutral
- (6) Inconclusive
- (7) Gain-of-function
- (8) Likely Switch-of-function
- (9) Switch-of-function

These mutation effects the underlying gene/protein functions. In a different competition NIPS 2017(2) will use this classification to determine which mutations cause tumor growth (oncogenic mutations) and which do not. The resulting oncogenic four categories are:

- 'Likely Oncogenic'
- 'Oncogenic'
- 'Likely Neutral'
- 'Inconclusive'

This project and Kaggle contest does not try to predict oncogenicity. Only the gene/protein mutation effects in generic terms, 1 through 9 categories, are investigated in this challenge. A predictive algorithm that could accurately determine effect from the mutation variant would be an initial step in reducing the huge amount of time for a clinical pathologists to manually interpret genetic mutations. This would make precision medicine more of a cost effective and common reality.

Big Picture: - Sequence tumor - Determine mutation variant - Investigate the literature for each mutation variant <—this Kaggle challenge - Determine one of nine generic categories of effect on protein function <———-this Kaggle challenge - Determine one of four categories of oncogenicity - Determine best course of action for specific patient's tumor

This challenge is more abstract than describe here. The 9 classes were only defined as class 1 through 9. The 9 mutation effects listed above were not give and CAN NOT be used in the modeling. This challenge is only to evaluate the clinical evidence 'text' to predict one of the numeric 1 through 9 class categories.

The goal here is to use the text and optional gene/variation 'function' and 'pathway' information to predict one of the 9 numeric categories. The goal is not to use a table that has the above mutation and oncogenic categories to get a perfect predictive algorithm. This was a point of heated contention in the discussion section of this Kaggle challenge. This resulted in people being removed from the leader board and stricter guidelines of external data that can and can not be used. Metrics:

- Quantifiable: The solution to this problem is quantifiable (the problem can be expressed in mathematical or logical terms). Using machine learning techniques to predict one of the nine categories of protein function alteration this is quantifiable. The training set has the correct category so we can make a validation subset and quantify the correctness of the model.

- Measurable: The measurable (the problem can be measured by some metric and clearly observed) metric to determine correctness in this Kaggle competition is log loss. I will use accuracy and log loss for this capstone project.

- Replicable: With the numpy.random.seed() this model will be replicable.

2.1.3 Metrics

This Kaggle completion uses log loss the evaluate the leader board and the final submissions. I will use I will use accuracy and log loss for this capstone project.

I will use, accuracy to to compare this model to the 1 in 9(11.1%). The for possibility of randomly picking one of the nine classes. For the kaggle competition I will submit log loss. My benchmark goal is to have a predictive accuracy better than 11.1%.

The logloss is the negative log-likelihood of the true labels given a probabilistic classifier's predictions. The log loss is -log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp))
F1 = 2 * (precision * recall) / (precision + recall)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
accuracy = (tp + tn)/ (tp + tn + fp + fn)

2.2 II. Analysis

2.2.1 Data Exploration

The files given are as follows:

- 'training_variants' - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the amino acid change for this mutations), Class (1-9 the class this genetic mutation has been classified on)(1)

- 'training_text' - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)(1)

- 'test_variants' - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the amino-acid change for this mutations)(1)

- 'test_text' - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)(1)

After data wrangling:

```
- X_train will have the fields 'ID', 'Gene', 'Variation', 'Text'
- y_train will have the fields 'ID', 'Class'
- X_test will have the fields 'ID', 'Gene', 'Variation', 'Text'
```

- 'Text' This is the expert annotated text specific for each mutation found. As mentioned above "This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature."(2) The text from the different mutations varies in length from 1 to 76782 word in length.

- 'Variant' has one of two formats. It could be words or protein location of the mutation with amino acid change.

'Variation' examples:

- Example 1 - "Truncating Mutations" This variation means the protein was not fully synthesized.
- Example 2 - "Amplification" would indicate a protein was over produced.
- Example 3 - "Fusions" means parts of the protein of are joined improperly.
- Example 4 - "Y599_D600insPAPQIMSTSTLISENMNIA" mean an insertion of amino acids "PAPQIMSTSTLISENMNIA" in to the protein between the amino acid Y(Tyrosine), in position 599, and amino acid D(Aspartic Acid) at position 600.
- Example 5 - "M541L" This is the most common type of Variant. It represents a change of the amino acid in position 541, of the protein. The underlying mutation caused the amino acid M(Methionine) to be changed to L(Leucine).

The data can be down loaded at this link https://www.kaggle.com/c/msk-redefining-cancer-treatment/data

Below shows examples of the data sets. The count of words in the text field are counted and displayed by class. Maximum, minimum, mean and standard deviation in the visualization an anomaly is shown. The number of samples given in the training set are not evenly distributed by class. The 8th and 9th classes are represented less than the other classes. And some sample have a 'Text' length of one.

Also, I did a download dump from biological web databases. This was done two ways, first using biopython to do a search of National Center for Biotechnology Information(NCBI) and get 'Gene Function' information about all the genes in the training and test sets. This is automatic and saved to a variable. Next I will use the website, "Database for Annotation Visualization and Integrated Discovery"(DAVID) site(9) to do a search for Gene function and Gene Pathway information for all the genes in the training and testing set. This was downloaded into a CVS and imported into the notebook as a variable. I tried adding the new text and kept each new text field separate.

```
In [65]: print("Train and Test variants shape : ",train_variants_df.shape, test_var
         print("Train and Test text shape : ",train_text_df.shape, test_text_df.sha
```

```
Train and Test variants shape :  (3321, 4) (5668, 3)
Train and Test text shape :  (3321, 2) (5668, 2)
```

```
In [66]: train_variants_df.head()

Out[66]:    ID    Gene             Variation  Class
         0   0  FAM58A  Truncating Mutations      1
         1   1     CBL                 W802*      2
         2   2     CBL                 Q249E      2
         3   3     CBL                 N454D      3
         4   4     CBL                 L399V      4

In [67]: train_text_df["Text_num_words"] = train_text_df["Text"].apply(lambda x: le
```

The type of data frame and data type of each field are as follows:

```
In [68]: print(type(train_text_df))
         print(type(train_variants_df))
         print(train_text_df.dtypes)
         print(train_variants_df.dtypes)
         print(train_text_df.shape)
         print(train_variants_df.shape)

<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
ID                 int64
Text              object
Text_num_words     int64
dtype: object
ID            int64
Gene         object
Variation    object
Class         int64
dtype: object
(3321, 3)
(3321, 4)
```

What does the data look like:

```
In [69]: train_text_df.head()

Out[69]:    ID                                               Text  Text_num_words
         0   0  Cyclin-dependent kinases (CDKs) regulate a var...            6089
         1   1   Abstract Background  Non-small cell lung canc...            5756
         2   2   Abstract Background  Non-small cell lung canc...            5756
         3   3  Recent evidence has demonstrated that acquired...            5572
         4   4  Oncogenic mutations in the monomeric Casitas B...            6202

In [70]: pd.isnull(train_variants_df).any()
```

```
Out[70]: ID           False
         Gene         False
         Variation    False
         Class        False
         dtype: bool
```

```
In [71]: print(train_text_df["Text_num_words"].max())
```

```
76782
```

```
In [72]: print(train_text_df["Text_num_words"].min())
```

```
1
```

Investigate why some text fields have a value of 1 and solve any problems.

```
In [73]: train_text_df[train_text_df['Text_num_words'] <50]
```

```
Out[73]:         ID  Text  Text_num_words
         1109  1109  null               1
         1277  1277  null               1
         1407  1407  null               1
         1639  1639  null               1
         2755  2755  null               1
```

The problem is some samples have 'null' in the text field. These will be removed and the index/ID reset.

```
In [74]: excluded =train_text_df[train_text_df['Text_num_words'] <50].index
         train_text_df=train_text_df[train_text_df.Text_num_words > 50]
         train_variants_df.drop(excluded, inplace=True)
```

```
In [75]: #Reset index
         train_text_df = train_text_df.reset_index(drop=True)
         train_variants_df = train_variants_df.reset_index(drop=True)

         train_text_df['ID'] = train_text_df.index
         train_variants_df['ID'] = train_variants_df.index
```

The mean and standard deviation are:

```
In [76]: print(train_text_df["Text_num_words"].mean())
```

```
9565.512062726177
```

```
In [77]: print(train_text_df["Text_num_words"].std())
```

```
7846.338304540312
```

The additional biological data from the David website.

```
In [89]: david_df.head()
```

```
Out[89]:       ID                                          Gene Name       Species
         0   2632          1,4-alpha-glucan branching enzyme 1(GBE1)  Homo sapiens
         1   1718            24-dehydrocholesterol reductase(DHCR24)  Homo sapiens
         2   9060  3'-phosphoadenosine 5'-phosphosulfate synthase...  Homo sapiens
         3   3158  3-hydroxy-3-methylglutaryl-CoA synthase 2(HMGCS2)  Homo sapiens
         4  26275           3-hydroxyisobutyryl-CoA hydrolase(HIBCH)  Homo sapiens

           BBID BIOCARTA                              COG_ONTOLOGY  \
         0  NaN      NaN  Carbohydrate transport and metabolism,
         1  NaN      NaN                                       NaN
         2  NaN      NaN                                       NaN
         3  NaN      NaN                                       NaN
         4  NaN      NaN                     Lipid metabolism,

                                                 KEGG_PATHWAY  \
         0  hsa00500:Starch and sucrose metabolism,hsa0110...
         1  hsa00100:Steroid biosynthesis,hsa01100:Metabol...
         2  hsa00230:Purine metabolism,hsa00450:Selenocomp...
         3  hsa00072:Synthesis and degradation of ketone b...
         4  hsa00280:Valine, leucine and isoleucine degrad...

                                      PIR_SEQ_FEATURE  \
         0                                        NaN
         1                                        NaN
         2                                        NaN
         3  active site: Cys,domain: transit peptide,
         4                                        NaN

                                             REACTOME_PATHWAY  \
         0  R-HSA-3322077:R-HSA-3322077,R-HSA-3878781:R-HS...
         1  R-HSA-6807047:R-HSA-6807047,R-HSA-6807062:R-HS...
         2  R-HSA-174362:R-HSA-174362,R-HSA-2408550:R-HSA-...
         3  R-HSA-1989781:R-HSA-1989781,R-HSA-77111:R-HSA-...
         4                        R-HSA-70895:R-HSA-70895,

                                              SP_COMMENT_TYPE  \
         0  catalytic activity,disease,function,online inf...
         1  cofactor,disease,function,online information,p...
         2  alternative products,catalytic activity,diseas...
         3  catalytic activity,disease,function,pathway,si...
         4  alternative products,catalytic activity,diseas...
```

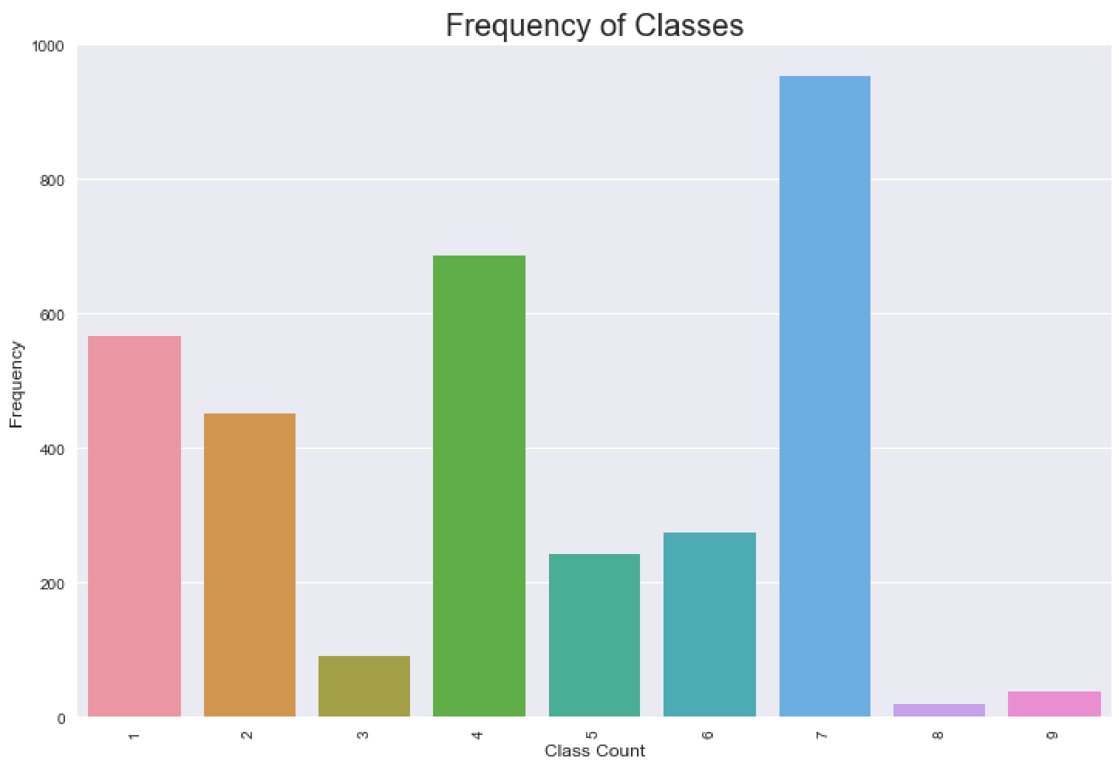
```
                                          UP_KEYWORDS  \
0  3D-structure,Acetylation,Complete proteome,Dis...
1  Alternative splicing,Cholesterol biosynthesis,...
2  3D-structure,Acetylation,Alternative splicing,...
3  3D-structure,Acetylation,Alternative splicing,...
4  3D-structure,Acetylation,Alternative splicing,...

                                       UP_SEQ_FEATURE
0  chain:1,4-alpha-glucan-branching enzyme,modifi...
1  chain:24-dehydrocholesterol reductase,domain:F...
2  active site:Phosphoserine intermediate,chain:B...
3  chain:Hydroxymethylglutaryl-CoA synthase, mito...
4  binding site:Substrate,binding site:Substrate;...
```

2.2.2 Exploratory Visualization

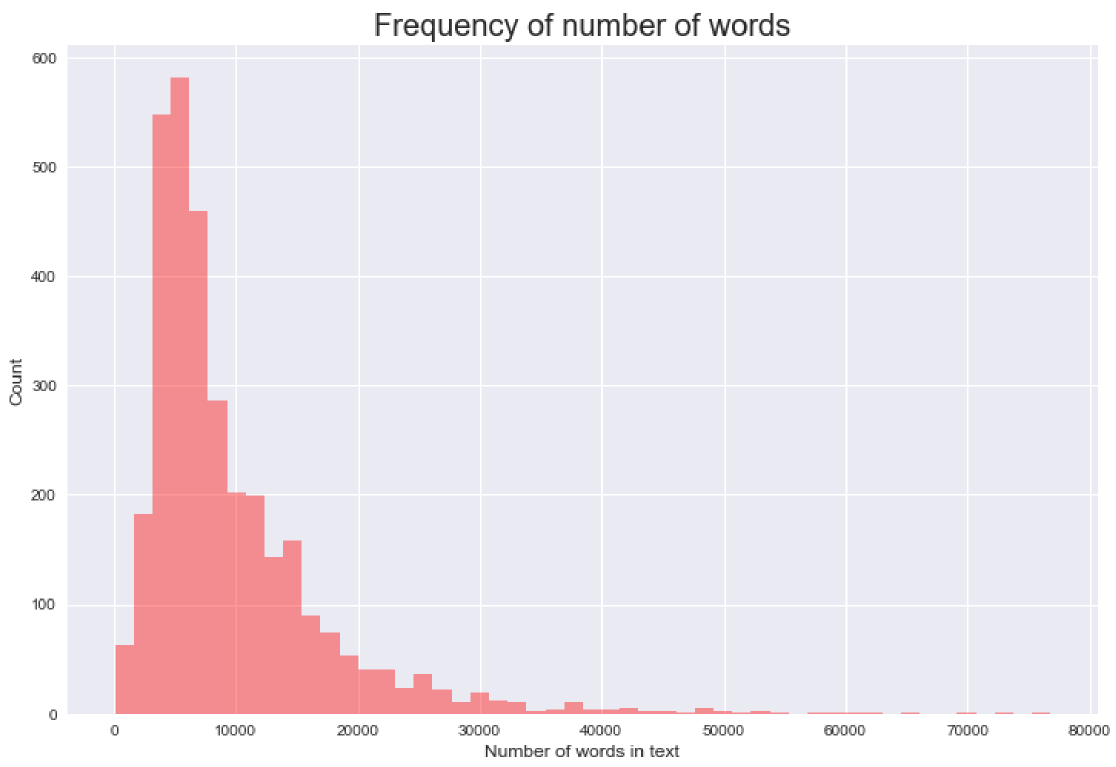
Below demonstrates: - The count of occurrence of each class. - A distribution of the number of words in each text field. - A box plot showing the median, quartiles by class.

These were done to determine if the word count could be used to predict class.

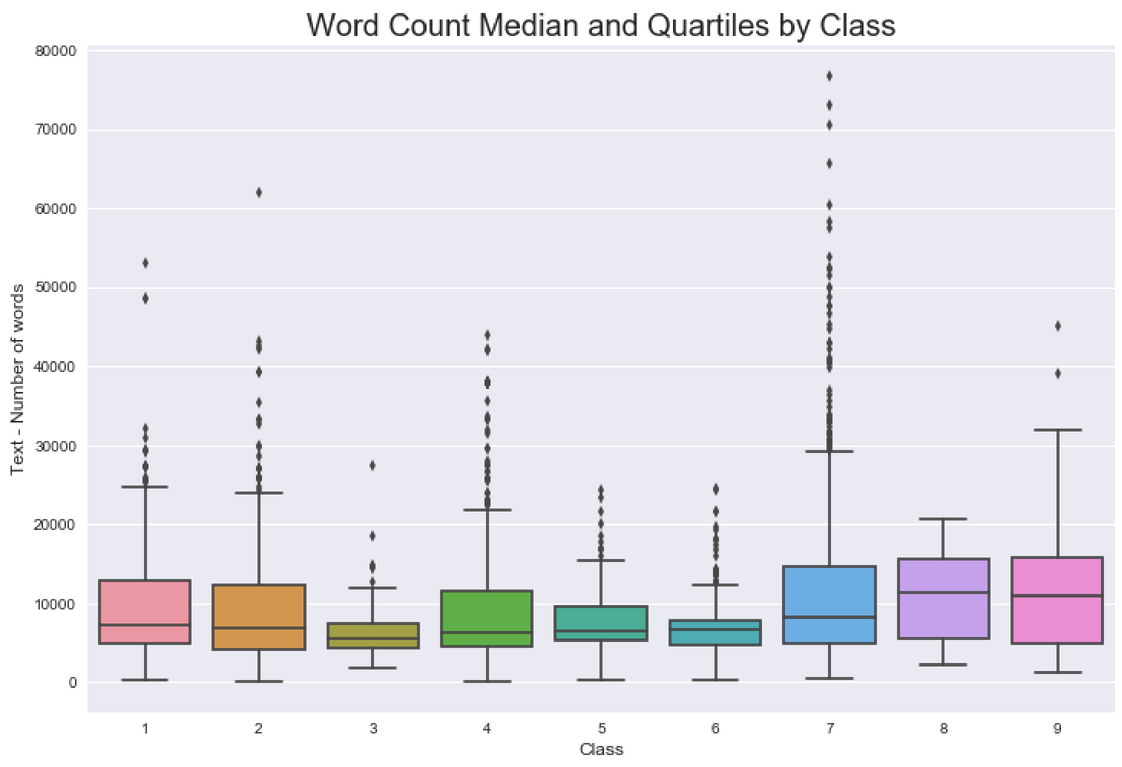
```
In [78]: plt.figure(figsize=(12,8))
         #sns.countplot(x="Class", data=train_variants_df, palette="Blues_d")
         sns.countplot(x="Class", data=train_variants_df)
         plt.ylabel('Frequency', fontsize=12)
         plt.xlabel('Class Count', fontsize=12)
         plt.xticks(rotation='vertical')
         plt.title("Frequency of Classes", fontsize=20)
         plt.show()
```



```
In [79]: plt.figure(figsize=(12, 8))
         sns.distplot(train_text_df.Text_num_words.values, bins=50, kde=False, colo
         plt.xlabel('Number of words in text', fontsize=12)
         plt.ylabel('Count', fontsize=12)
         plt.title("Frequency of number of words", fontsize=20)
         plt.show()
```



```
In [80]: train_df = pd.merge(train_variants_df, train_text_df, on='ID')

         plt.figure(figsize=(12,8))
         sns.boxplot(x='Class', y='Text_num_words', data=train_df)
         plt.xlabel('Class', fontsize=12)
         plt.ylabel('Text - Number of words', fontsize=12)
         plt.title("Word Count Median and Quartiles by Class", fontsize=20)
         plt.show()
```



2.2.3 Algorithms and Techniques

I adapted the code from Sklearn 'Classification of text documents using sparse features'(8) to my data and removed algorithms that did not give log loss and added other algorithms.

- The following algorithms were tested:
  - test AdaBoost
  - test kNN
  - test MLPClassifier
  - test SGDClassifier L2 penalty
  - test SGDClassifier L1 penalty
  - test SGDClassifier Elastic-Net penalty
  - test MultinomialNB
  - test BernoulliNB
  - test Keras (tested separately)

Also the Sklearn CountVecotizer(10) and TfidfVectorizer(12) were tested individually. These were tested to see if the count or sparsity of words in the text could be used to predict the category of Class.

2.2.4 Benchmark

The model will predict one of nine categories. So the benchmark I will use is to create a model that has a predictive accuracy better than chance. Chance based on a random number between

1 - 9 would give an accuracy of about 11.1%. The model will have its accuracy compared to the benchmark accuracy of 11.1% to determine if it does better than chance.

2.3 III. Methodology

2.3.1 Data Preprocessing

As mentioned above samples with a 'Text' value of 'null' were removed.

Sklearn CountVecotizer(10) was used to preprocess the data. This counted the number of words in the text by categories. Also Sklearn TfidfVectorizer(12) was used to find specific rare / sparse word in the entire corpus of word and then these very sparse word may be useful in predicting the Class category.

Typical text preprocessing like punctuation removal, number removal were not done because in the text there are gene names or mutation positions that may have number or punctuation that may be specific to a class. All letters were change to lower case.

The data dumped from the biological websites preprocessed in the same way.

2.3.2 Implementation

- Implementation Workflow Outline:
  - Data Wrangling - Down load given files and create X_train, y_train, X_test and y_test variables.
  - StratifiedShuffleSplit X_train and X_train to get X_train, y_train, X_valid and y_valid.
  - Find gene function and pathway information to add to corpus.
    - use biopython to get info from ncbi function
    - use david.com to get info function and pathway
  - TfidfVectorizer to vectorize the text corpus and look for words specific to one of the nine categories:
    - test without additional info
    - test with additional info
  - test different machine learning algorithm:
    - test AdaBoost
    - test kNN
    - test MLPClassifier
    - test SGDClassifier L2 penalty
    - test SGDClassifier L1 penalty
    - test SGDClassifier Elastic-Net penalty
    - test MultinomialNB
    - test BernoulliNB
    - test Keras

Complications that occurred in the above process were many. First the distribution of the 9 Classes were not evenly distributed. This was especially true for classes 8 and 9. This could be a problem when I split the training data into training and validation sets. If most of classed 8 and 9 just happen to be in the validation set these two classed would not be well represented for training the model. Two overcome this problem Sklearn StratifiedShuffleSplit(12) used to split the data. This ensured each category were well represented in the training set.

Another complication was creating TfidVectorized data and keeping each separate so different combination could be tested.

The National Center for Biotechnology Information(NCBI) additional data was one additional corpus, ‘gene summary’. The data from ”Database for Annotation Visualization and Integrated Discovery“(DAVID) biological website had function and pathway information form different website resulting in nine new fields:

- ‘BBID’
- ‘BIOCARTA’
- ‘COG_ONTOLOGY’
- ‘KEGG_PATHWAY’
- ‘PIR_SEQ_FEATURE’
- ‘REACTOME_PATHWAY’
- ‘SP_COMMENT_TYPE’
- ‘UP_KEYWORDS’
- ‘UP_SEQ_FEATURE’

Each was used to create a new corpus. The way I evaluated each field for sparse word was stack each TfidVector with the original ‘Text’, ‘Variant’ and ‘Gene’ corpuses. Each was kept separate so different combinations could be tested.

2.3.3 Refinement

First I tried the default setting of the Adaboost algorithm with TfidfVectorizer using only the ‘Text’ field. Then I modified the ‘Classification of text documents using sparse features’(6) form sklearn to this data using TfidfVectorizer only on the ’Text’ field . I removed algorithms that could not calculate log loss because the Kaggle competition evaluates on log loss. For the refinement of my process I used accuracy first and log loss.

CountVecotizer was also evaluated. The visualization below show the results for the different algorithms tested using TfidfVectorizer.

To improve the model I tried varying the input fields in the algorithms. I also tried adding the ‘Gene’ ‘Variation’ fields.. I tried treated each as a separate corpus and creating a TfidfVectorizer for each. Later stacking these to the ‘Text TfidfVectorizer’ for evaluation.

In an attempt to improve the model I augmented the data with outside data. As mentioned above I used NCBI data and David data. This contained information about gene function and pathways totaling 10 new fields. Again all new fields were treated as a separate corpus. Separate TfidfVectorizer for each were created. Later these new TfidfVectorizer where stacked with the ‘Text’, ‘Gene’ and ‘Variation’ TfidfVectorizer and evaluated. Also, many subset combinations of all the TfidfVectorizers were tested.

This code was challenging to add the new data fields to the original data frame then rerun the indexing and then create a TfidfVectorizer for each separately. This was done to allow subsets of the many possible TfidfVectorizer to be stacked and tested.

2.4 IV. Results

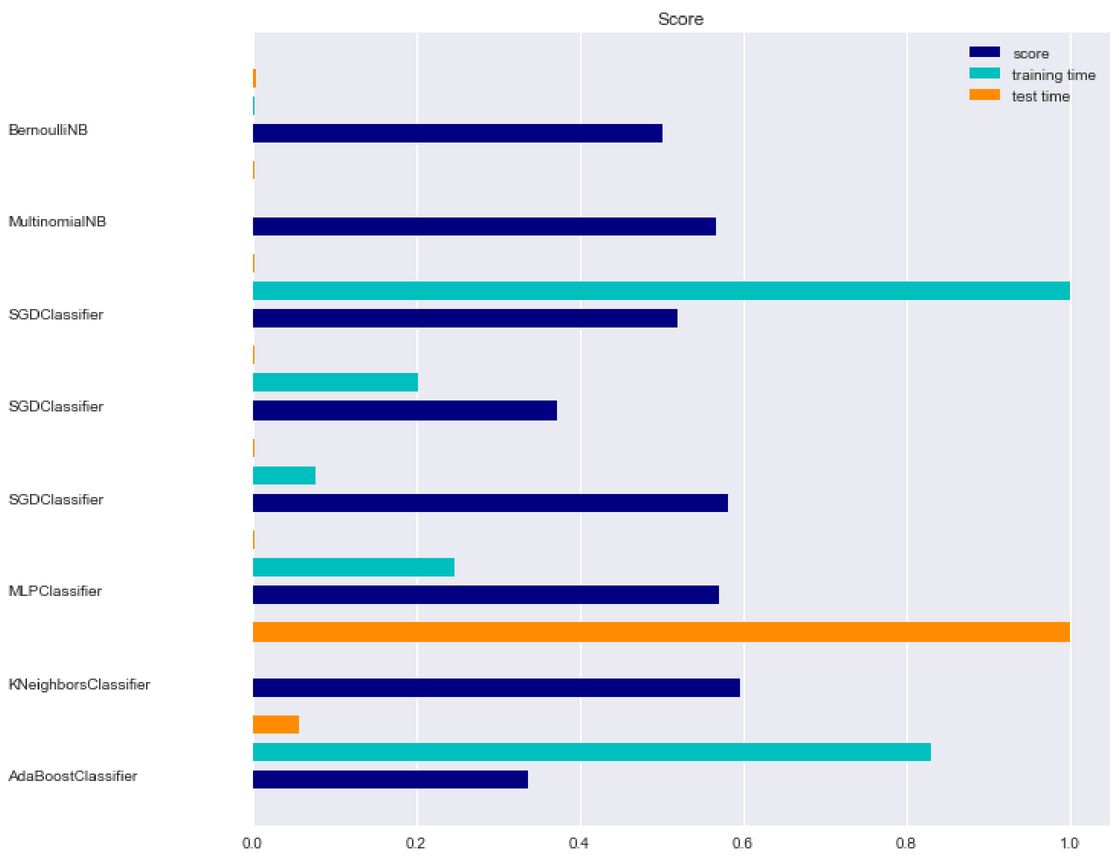
2.4.1 Model Evaluation and Validation

This model was created by using logical tools available and attempting to create ideas to improve the results. I first tried TfidfVectorizer and adaboost on the ‘Text’ field getting a result of 38% accuracy. Then I modified Sklearn ‘Classification of text documents using sparse features’ to this data to evaluate many models at one time. From this I saw that SGDClassifier(penelty='l2') preformed the best with an accuracy of 58%. Again this was done only on the 'Text' field. The results are shown below:

'Text' corpus only

```
In [91]: extract_sparse(X_train_pre=X_train_pre,X_test_pre = X_test_pre, y_train_pr

Extracting features from the training data using a sparse vectorizer
done in 182.240224s at 1.044MB/s
n_samples: 2988, n_features: 16000

Extracting features from the test data using the same vectorizer
done in 14.635011s at 1.496MB/s
n_samples: 333, n_features: 16000

================================================================================
AdaBoostCladdifier
________________________________________________________________________________
Training:
train time: 169.935s
2.04434067131 log_loss(y_test, pred_proba)
test time:  0.820s
accuracy:   0.336

================================================================================
kNN
________________________________________________________________________________
Training:
train time: 0.062s
nan log_loss(y_test, pred_proba)
test time:  14.245s
accuracy:   0.598

================================================================================
MLPClassifier
________________________________________________________________________________
Training:
train time: 50.649s
3.02465664469 log_loss(y_test, pred_proba)
test time:  0.026s
accuracy:   0.571

================================================================================
L2 penalty
________________________________________________________________________________
Training:
train time: 15.819s
```

```
1.32405300507 log_loss(y_test, pred_proba)
test time:  0.027s
accuracy:   0.583


================================================================================
L1 penalty
________________________________________________________________________________
Training:
train time: 41.408s
1.75642866298 log_loss(y_test, pred_proba)
test time:  0.027s
accuracy:   0.372


================================================================================
Elastic-Net penalty
________________________________________________________________________________
Training:
train time: 204.442s
1.51850317494 log_loss(y_test, pred_proba)
test time:  0.027s
accuracy:   0.520


================================================================================
Naive Bayes
________________________________________________________________________________
Training:
train time: 0.153s
2.75576567958 log_loss(y_test, pred_proba)
test time:  0.026s
accuracy:   0.568



________________________________________________________________________________
Training:
train time: 0.328s
nan log_loss(y_test, pred_proba)
test time:  0.063s
accuracy:   0.502


================================================================================
```



2.4.2 Model Evaluation and Validation cont

After varying many input features of many models I found the best results had the combinations of: - TfidfVectorizer(max_features= 16000, strip_accents=unicode,lowercase =True, - analyzer=word, token_pattern=rw, ngram_range=(1, 3), use_idf=True,
- smooth_idf=True, sublinear_tf=True, stop_words = english)

and - SGDClassifier(alpha=.00005, n_iter=75, penalty=penalty)

After modifying the input variables as shown here there the accuracy increased to 66%.

After adding the 'Gene' and 'Variation' fields as there own corpus to create two new TfidfVectorizer and stacking them all together resulted in the model having and 69%. This result is below:

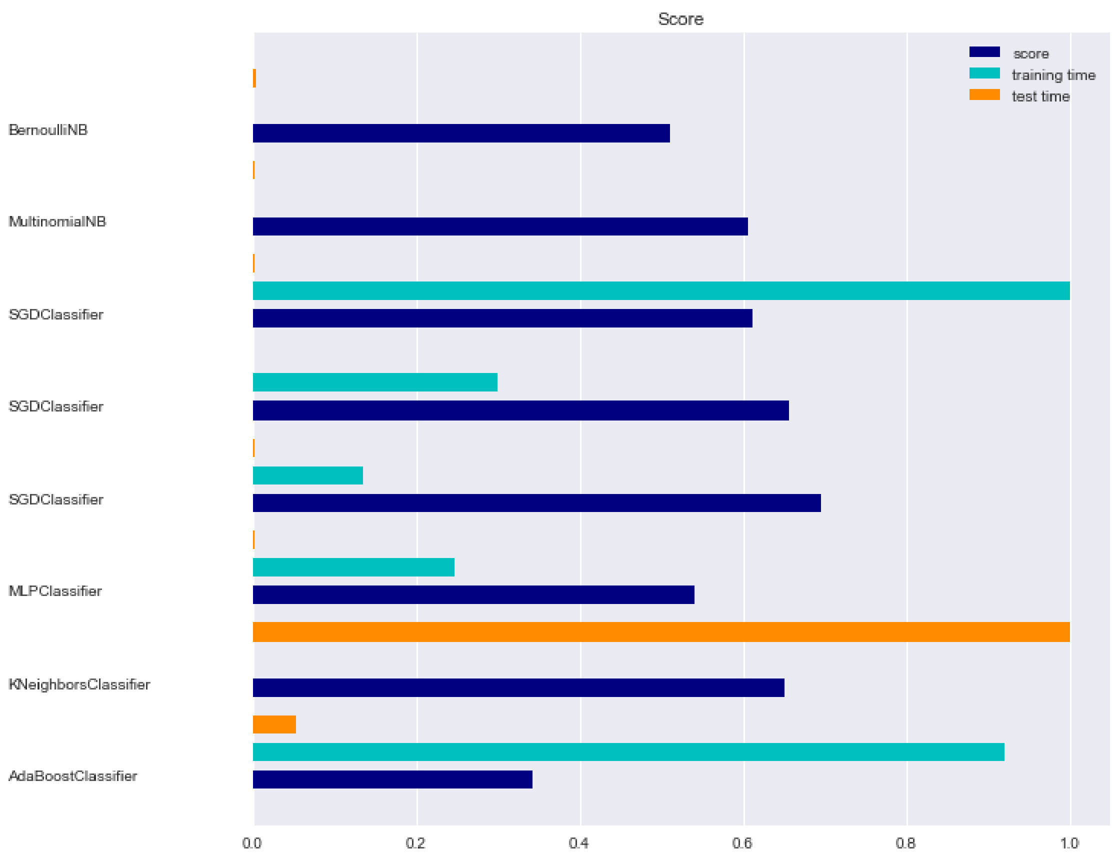
'Text', 'Gene' and 'Variant' corpuses stacked (after refinement)

```
In [97]: extract_sparse_modified(X_train_pre=X_train_pre,X_test_pre = X_test_pre, y
```

```
Extracting features from the training data using a sparse vectorizer
done in 205.747725s at 0.924MB/s
n_samples: 2988, n_features: 19392

Extracting features from the test data using the same vectorizer
```

```
done in 15.601299s at 1.403MB/s
n_samples: 333, n_features: 19392

================================================================================
AdaBoostCladdifier
________________________________________________________________________________
Training:
train time: 178.070s
2.05263520846 log_loss(y_test, pred_proba)
test time:  0.877s
accuracy:   0.342

================================================================================
kNN
________________________________________________________________________________
Training:
train time: 0.067s
nan log_loss(y_test, pred_proba)
test time:  16.476s
accuracy:   0.652

================================================================================
MLPClassifier
________________________________________________________________________________
Training:
train time: 47.825s
nan log_loss(y_test, pred_proba)
test time:  0.042s
accuracy:   0.541

================================================================================
L2 penalty
________________________________________________________________________________
Training:
train time: 26.042s
0.878044562408 log_loss(y_test, pred_proba)
test time:  0.031s
accuracy:   0.697


================================================================================
L1 penalty
________________________________________________________________________________
Training:
train time: 57.976s
1.07760070114 log_loss(y_test, pred_proba)
test time:  0.026s
accuracy:   0.658
```

```
================================================================================
Elastic-Net penalty
________________________________________________________________________________
Training:
train time: 193.489s
1.23431066094 log_loss(y_test, pred_proba)
test time:  0.034s
accuracy:   0.613


================================================================================
Naive Bayes
________________________________________________________________________________
Training:
train time: 0.167s
2.85889206824 log_loss(y_test, pred_proba)
test time:  0.030s
accuracy:   0.607


________________________________________________________________________________
Training:
train time: 0.285s
nan log_loss(y_test, pred_proba)
test time:  0.062s
accuracy:   0.511


================================================================================
```



2.4.3 Model Evaluation and Validation cont 2

Random state was used for all algorithms that had it as an input variable. Random state was used for all algorithms that had it as an input variable. I noticed it consistently decreased the accuracy for most algorithms by about 5% so I removed it.

The biological dump data was treated as individual corpus each with it own TfidfVector. These were all evaluated as a group of stacked TfidfVector and one at a time concatenated to the main text corpus. To my surprise and disappointment none increase accuracy and sometimes decreased accuracy.

Sensitivity Analysis Additionally, to test the robustness of the model I removed the main 'text' , 'gene' and 'variant' corpuses. Then I added the nine corpuses from the external David date I downloaded. It did surprisingly well but in no case did it improve accuracy. These results are below:

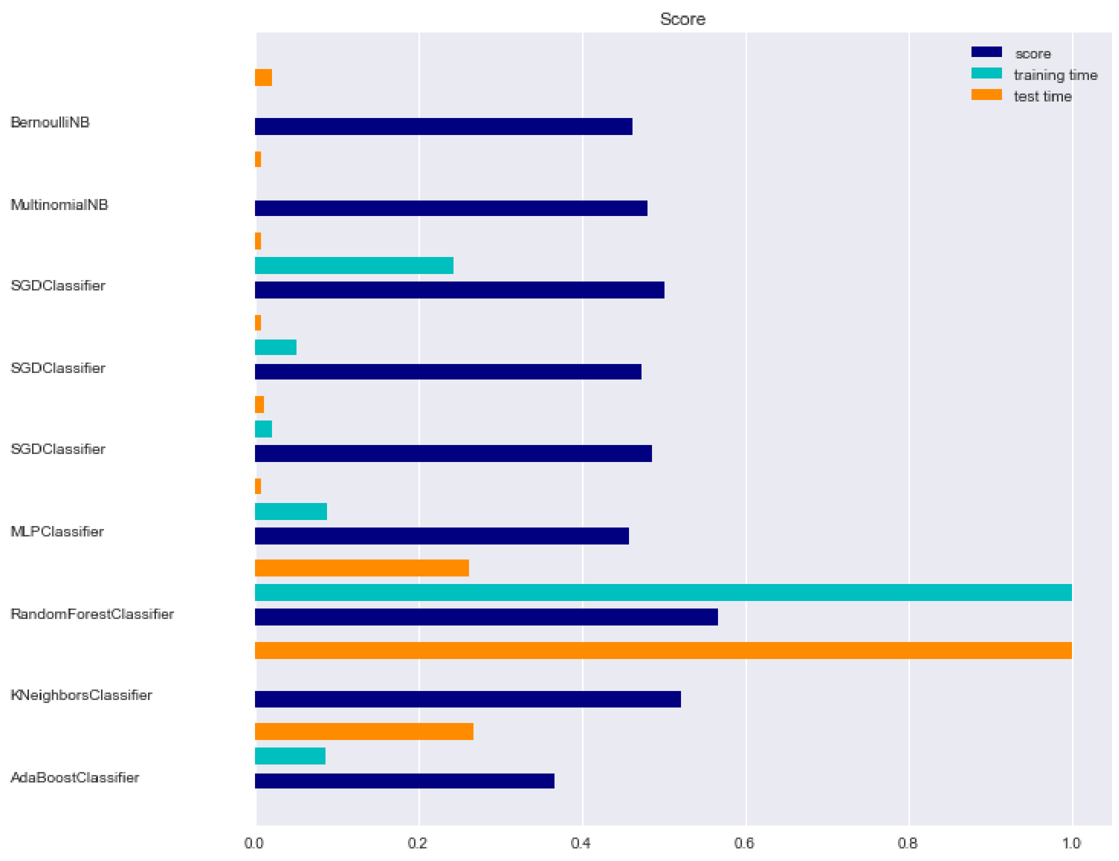
9 David corpuses no original 'text', 'gene' and 'variaton' corpuses

```
In [45]: extract_sparse_modified_david(X_train_pre=X_train_pre,X_test_pre = X_test_
```

```
Extracting features from the training data using a sparse vectorizer
done in 3.162624s at 84.480MB/s
n_samples: 2988, n_features: 20437
```

```
Extracting features from the test data using the same vectorizer
done in 0.292384s at 104.219MB/s
n_samples: 333, n_features: 20437

================================================================================
AdaBoostCladdifier
________________________________________________________________________________
Training:
train time: 7.730s
2.05703051928 log_loss(y_test, pred_proba)
test time:  0.158s
accuracy:   0.366

================================================================================
kNN
________________________________________________________________________________
Training:
train time: 0.008s
nan log_loss(y_test, pred_proba)
test time:  0.591s
accuracy:   0.523

================================================================================
Random forest
________________________________________________________________________________
Training:
train time: 90.161s
nan log_loss(y_test, pred_proba)
test time:  0.155s
accuracy:   0.568

================================================================================
MLPClassifier
________________________________________________________________________________
Training:
train time: 8.022s
1.69776740475 log_loss(y_test, pred_proba)
test time:  0.005s
accuracy:   0.459

================================================================================
L2 penalty
________________________________________________________________________________
Training:
train time: 1.948s
1.24571157496 log_loss(y_test, pred_proba)
test time:  0.007s
accuracy:   0.486
```

```
================================================================================
L1 penalty
________________________________________________________________________________
Training:
train time: 4.607s
1.33489541732 log_loss(y_test, pred_proba)
test time:  0.005s
accuracy:   0.474

================================================================================
Elastic-Net penalty
________________________________________________________________________________
Training:
train time: 21.947s
1.4363733116 log_loss(y_test, pred_proba)
test time:  0.005s
accuracy:   0.502

================================================================================
Naive Bayes
________________________________________________________________________________
Training:
train time: 0.017s
12.873441152 log_loss(y_test, pred_proba)
test time:  0.005s
accuracy:   0.480


________________________________________________________________________________
Training:
train time: 0.025s
nan log_loss(y_test, pred_proba)
test time:  0.013s
accuracy:   0.462

================================================================================
```



Keras This keras model had a low validation accuracy.

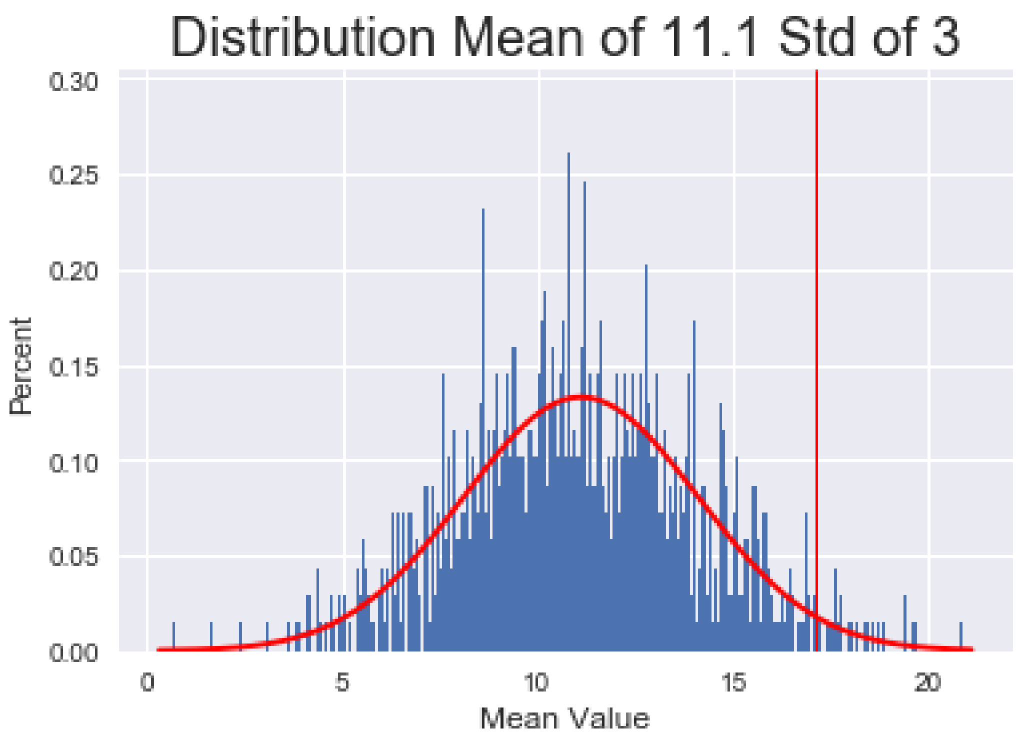
```
In [60]: estimator.fit(sentence_vectors[0:3321], dummy_y, validation_split=0.10)

Train on 2988 samples, validate on 333 samples
Epoch 1/5
2988/2988 [==============================] - 1s - loss: 1.6417 - acc: 0.4227 - val_
Epoch 2/5
2988/2988 [==============================] - 0s - loss: 0.9943 - acc: 0.6396 - val_
Epoch 3/5
2988/2988 [==============================] - 0s - loss: 0.7564 - acc: 0.7115 - val_
Epoch 4/5
2988/2988 [==============================] - 0s - loss: 0.6400 - acc: 0.7577 - val_
Epoch 5/5
2988/2988 [==============================] - 0s - loss: 0.5630 - acc: 0.7855 - val_


Out[60]: <keras.callbacks.History at 0x1b85a4f60>
```

2.4.4 Justification

This model used TfidVectorizer followed by SGDClassifier with and 'l2' penalty. This model used sparse words to identify the class from groups of text. Each group of text had some words that were specific to a class. This allows the model to predict which of the nine classes. This model had an accuracy better than other models tested and better than chance.

The final model did work better than the benchmark of random classification prediction. Since there are nine Classes in this model that would be a 1 in 9 chance or 11.1% chance of being accurate. If we tested the 3321 training samples 1000 times a distribution would be created with a mean of 11.1%. If the standard deviation is, lets say, 3 a 95% confidence limit would be greater than 2 standard deviations away from the mean. This would be greater than 17.1% accuracy. The final model here had an accuracy of 68% that is clearly above the cutoff of 17.1%. This model did better than chance.

```
In [61]: mu, sigma = 11.1, 3 # mean and standard deviation
         s = np.random.normal(mu, sigma, 3321)

         import matplotlib.pyplot as plt
         count, bins, ignored = plt.hist(s, 1000, normed=True)
         plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
                         np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
                   linewidth=2, color='r')
         plt.axvline(linewidth=1, color='r', x = 17.1)
         plt.title("Distribution Mean of 11.1 Std of 3", fontsize=20)
         plt.xlabel('Mean Value', fontsize=12)
         plt.ylabel('Percent')

         plt.show()
```

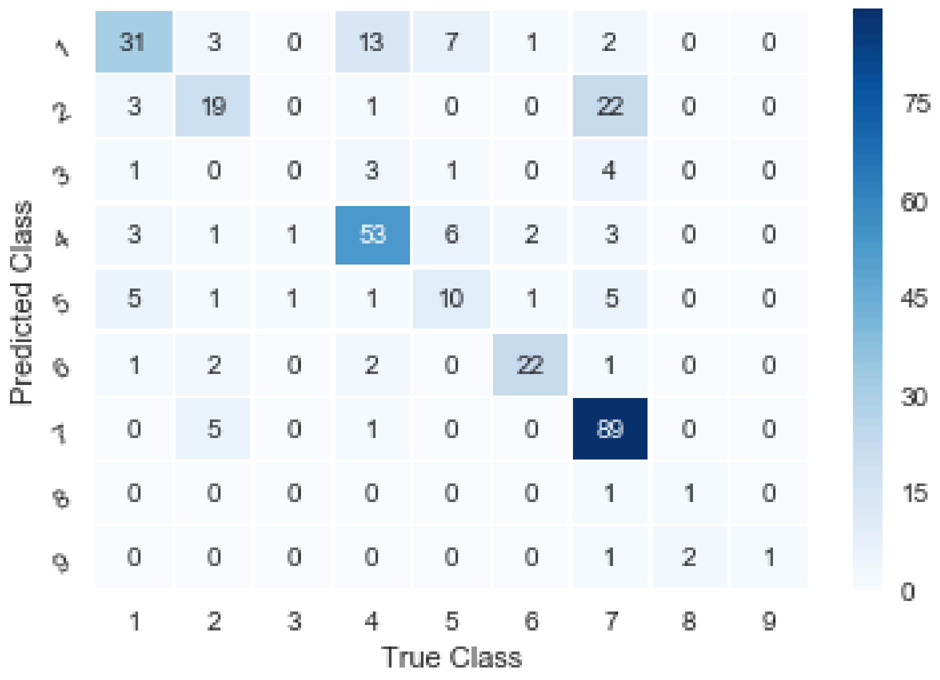


2.5 V. Conclusion

2.5.1 Free-Form Visualization - Possible Improvement Idea

```
In [64]: extract_sparse_modified_plot(X_train_pre=X_train_pre,X_test_pre = X_test_p
```

```
================================================================================
L2 penalty
________________________________________________________________________________
Training:
top 10 keywords per class:
1: v804g w349c y835f h1686r foxa1 mutations ercc2 truncating mutations trunca...
2: d1203n s33f mpl r267p ros1 k292i hras abl1 epas1 fusions
3: p47s t131s i122v k125r wildtype e839k s1651f c628y r462i g1706a
4: f808l cdkn2b stk11 keap1 lats1 r2659k a598t cdkn2a nf1 pten
5: p25l a8s r592h r1608s i491m p848l h870r h297n t123a s362l
6: h94y dusp4 h115n kmt2b s921r v1804d g42r v84l d1818g brca2
7: myd88 y139d akt2 r15s rac1 l493p nfe2l2 ctnnb1 jak1 fusion
8: dnmt3b7 k179m r132q k700r e40n k590r idh2 h3f3a s492r bcor
9: ctcf r100a g44s idh1 r339w k181m u2af1 c135y ezh2 sf3b1

Confusion matrix, without normalization
```



	1	2	3	4	5	6	7	8	9
1	31	3	0	13	7	1	2	0	0
2	3	19	0	1	0	0	22	0	0
3	1	0	0	3	1	0	4	0	0
4	3	1	1	53	6	2	3	0	0
5	5	1	1	1	10	1	5	0	0
6	1	2	0	2	0	22	1	0	0
7	0	5	0	1	0	0	89	0	0
8	0	0	0	0	0	0	1	1	0
9	0	0	0	0	0	0	1	2	1

==

2.5.2 Free-Form Visualization - Possible Improvement Idea cont

A confusion matrix could be used to try to improve the model. The confusion matrix would ideally have all the true value match the test/predicted values. Here is a confusion matrix that is conditionally formatted to make a cell background color darker blue the greater the value of that cell. Ideally there should be diagonal(from top left to bottom right) dark blue line of cells representing the correct matches. The blue squares not within the diagonal line represent mismatches. I was thinking these mismatched categories may share sparse words causing a incorrect classification. So for the darkest blue cells out side the diagonal blue line removing the overlapping sparse words may improve accuracy. This can be done by creating a list for both categorizing and removing the over lapping words. The numpy function 'np.intersect1d' can create a list of the overlapping words. The words in this list can be added to the stop word list. This would then make the prediction rely on none overlapping words.

Also as an alternative to doing this manually a function could be written to search the confusion matrix for the highest mismatched value, find and remove overlapping words as mentioned above,retest model and if the accuracy improve permanently add overlapping words to stop words. If the accuracy does not improve then remove the overlapping words from stop words. Then go the next highest mismatched cell and repeat the process until accuracy does not improve.

2.5.3 Reflection

This project and Kaggle competition is a small step in personalized medicine. As mentioned above the goal is to create a model to predict if a mutation in a tumor is responsible for tumor growth. First the effect of the mutation on the encoding protein must be determined:

- (1) Likely Loss-of-function
- (2) Likely Gain-of-function
- (3) Neutral
- (4) Loss-of-function
- (5) Likely Neutral
- (6) Inconclusive
- (7) Gain-of-function
- (8) Likely Switch-of-function
- (9) Switch-of-function

That is what this challenge is about. Later challenges will be done to determine oncogenicity:

- 'Likely Oncogenic'
- 'Oncogenic'
- 'Likely Neutral'
- 'Inconclusive'

And finally the oncogenicity can be used to determine the best course of treatment for each individual patient.

Here the model predicts one of the nine categories above. Many algorithms were tested and compared:

- AdaBoost
- kNN
- MLPClassifier
- SGDClassifier L2 penalty
- SGDClassifier L1 penalty
- SGDClassifier Elastic-Net penalty
- MultinomialNB
- BernoulliNB
- Keras

All english stop words were removed from the data. Word CountVectroization and TfidfVectorization were used to prepare the data for one of the above algorithms. After much trials I found TfidfVectorizer with SGDClassifier with an L2 penalty had the best accuracy. Log Loss is the metric for the Kaggle competition evaluation. Here I focused on accuracy and then log loss.

I tried to augment the data by getting information about a gene function and pathway. I used National Center for Biotechnology Information(NCBI) to get 'Gene Function' information. The was a clean data dump from the NCBI data base to my computer and saved into a variable. I used biopython and imported Entrez from the 'Bio' python library.

I used the biology website, "Database for Annotation Visualization and Integrated Discovery"(DAVID) site(9) to do a search for Gene function and Gene Pathway. This was a little more challenging. I needed to supply a list of gene name. I created a list of unique gene and input these to the David website. I checked the fields that I wanted 'function' and 'pathway' information and it processes (and crashed a few times) my request and downloaded a csv file with infomation, from many websites, about function and pathway for each of the genes.

I interseting problem I had was the NCBI used gene symbol and the data from kaggle had gene name. Lucky The david website had a conversion function that save the day.

The gene summary data from NCBI over lapped the give text data so it was excluded.

The david data had different fields each one with information from a different biological website or category. Each field was treated as its own corpus. The fields for each gene are:

'BBID' 'BIOCARTA' 'COG_ONTOLOGY' 'KEGG_PATHWAY' 'PIR_SEQ_FEATURE' 'REACTOME_PATHWAY' 'SP_COMMENT_TYPE' 'UP_KEYWORDS' 'UP_SEQ_FEATURE'

These were stacked with the original 'text', 'gene' and 'variant' corpuses. 12 corpuses in all. To my surprise and disappointment the accuracy did not improve and some cases effected the log loss negatively. This was very disappointing considering the time I put in until I realized I could use the david data as an 'alternate data set' to test the robustness of the model, above. I removed the main 'text', 'gene' and 'variant' corpuses and replace it with the 9 fields above. Surprisingly, It did almost well as the text, variation and gene data. This was with only web based information about each 'gene' used . If I had additional web based information about the 'variant' field this would have probably had better accuracy.

This was a great project about a topic I have a passion about. It used science, programing and machine learning to make people healthier. This would be a awesome and rewarding path to take and so would robotics or self driving car and now self driving plane!! Only time will tell which of those paths I will take but the first path of machine learning / deep learning / predictive algorithms is clear.

2.5.4 Improvement

As described above a confusion matrix can be used to identify high areas of mis categorization between two classes and remove overlapping words and adding them to the stop words. This would hopefully allow other non overlapping sparse words to correctly classify sample.

Also, getting more information about the 'variant' field from biological web sites. Information about the location of the mutation in the gene such as if the mutation in the part of the gene that 'turns on'(regulation) the encoding protein production or is it a structural location. If it is in an area of regulation the mutation may be more likely to be Class 1 - 'Likely Loss-of-function' or class 2 - 'Likely Gain-of-function'. If the mutation is in a structural part of the protein this could change the shape of the protein and this may be more likely class 9 -'Switch-of-function'.

Also improving the algorithm by combining TfidfVectorization and CountVectorization at the same time may improve accuracy. Also, maybe taking each letter and the numbers of the 'Variant' field and one hot encoding each. For example the 'Variant' of "M541L" the first letter is the original amino acid of the protein, the number is the location of the mutation and the last letter is the new amino acid in the protein caused be the mutation. The location may be informative as mentioned above. Also, some amino acids are substituted with very similar amino acid this many be more likely a Class 5 -'Likely Neutral mutation. One hot encoding the first letter, number and the last letter many give good predictive information.

Lastly, incorporate pipeline and gridsearchCV to try different parameters(13) may improve accuracy.