

Personalized Medicine and cancer treatment

Background

A lot has been said during the past several years about how precision medicine and, more concretely, how genetic testing is going to disrupt the way diseases like cancer are treated.

But this is only partially happening due to the huge amount of manual work still required. Once sequenced, a cancer tumor can have thousands of genetic mutations. Currently this interpretation of genetic mutations is being done manually. 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.

The project is to distinguish the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers). And to develop a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations.

For example, the text description for CBL gene's variant w802 is " Abstract Background Non-small cell lung cancer (NSCLC) is a heterogeneous group of disorders with a number of genetic and proteomic alterations. c-CBL is an E3 ubiquitin ligase and adaptor molecule important in normal homeostasis and cancer. We determined the genetic variations of c-CBL, relationship to receptor tyrosine kinases (EGFR and MET), and functionality in NSCLC. ...". The pathologist classified the variant as class 2 according to the text. It should be related to tumor growth. Our task is to automate the classification process.

The data set has four variables which are genes, variation, class and text. The genes are cancer related genes which means they are overexpressed or low expressed in cancer patients. Variation are the same types of cancer related genes with minor difference of gene structure. For example, CBL is lung cancer related genes. It has at least four types such as W802, Q249E, N454D and L399V. Variation W802, Q249E, N454D and L399V are all CBL genes but just has some minor difference in their gene structure.

The minor structure changes of gene, or so called variation (mutations), will contribute to tumor growth or have no effect on the tumor. The variable 'Class' in the data set is the classification of the gene's variation according to whether it helps the tumor growth or not. Knowing the gene variants belong to which class can help the doctor to find the best suitable treatment for different cancer patients. It is called personalized medicine when Patients with same type of disease but are treated with different medicine due their genetic testing.

Data wrangling

There are six records 'Text' variable are null and I delete the six rows.

Explanatory data analysis

I. Gene distribution overview

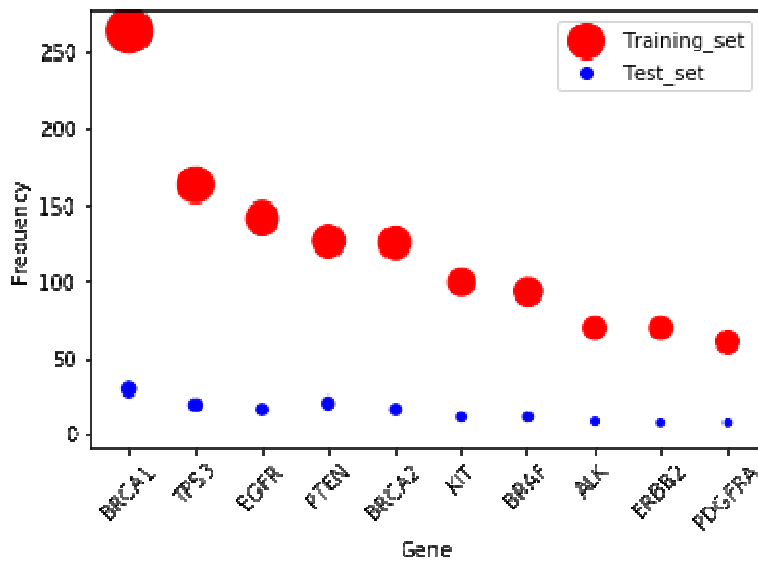
Top 10 occurred genes

Genes with maximal occurrences in training data Gene BRCA1 264 TP53 163 EGFR 141 PTEN 126 BRCA2 125 KIT 99 BRAF 93 ERBB2 69 ALK 69 PDGFRA 60 Name: ID, dtype: int64

Genes with maximal occurrences in test data Gene F8 134 CFTR 57 F9 54 G6PD 46 GBA 39 PAH 38 AR 38 CASR 37 ARSA 30 BRCA1 29 Name: ID, dtype: int64

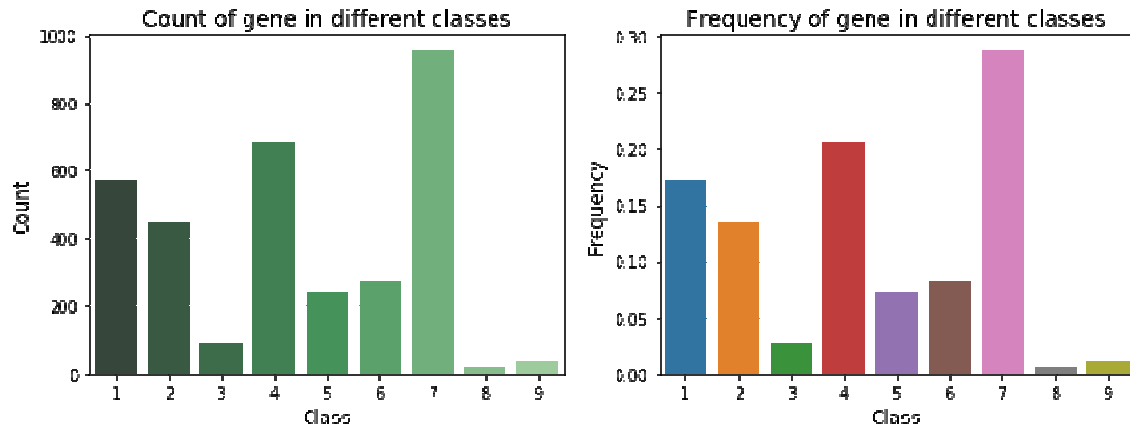
The top ten genes in training data are different from that of the test data. The only common top ten gene is BRCA1. It shows the gene distributions in the training and test data are not the same.

Top 10 genes of training set scatter plot



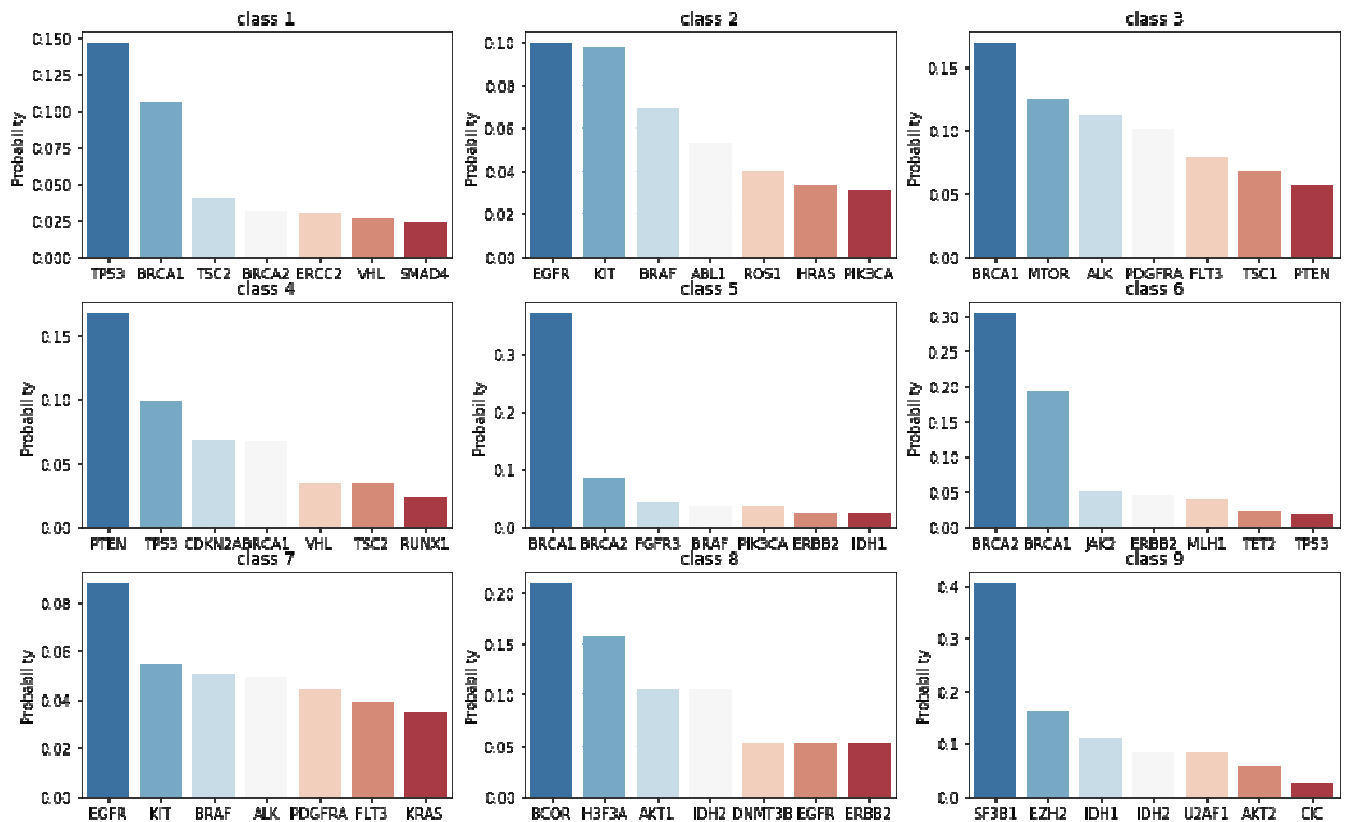
II. Gene distribution in classes

Gene frequency in nine mutation classes



Gene's frequency peaks at class 7. Lowest gene occurrence class are 8 and 9. Middle high class are class 1, 2 and 4. Middle low class are class 3, 5 and 6.

Top seven genes distribution in different classes



EGFR rank first in class 2 and 7.

BRCA1 ranks first in class 3, 5 and ranks second in class 6.

BRCA2 ranks first in class 6 and ranks second in 5.

And these three genes are among the top ten occurred genes in training data.

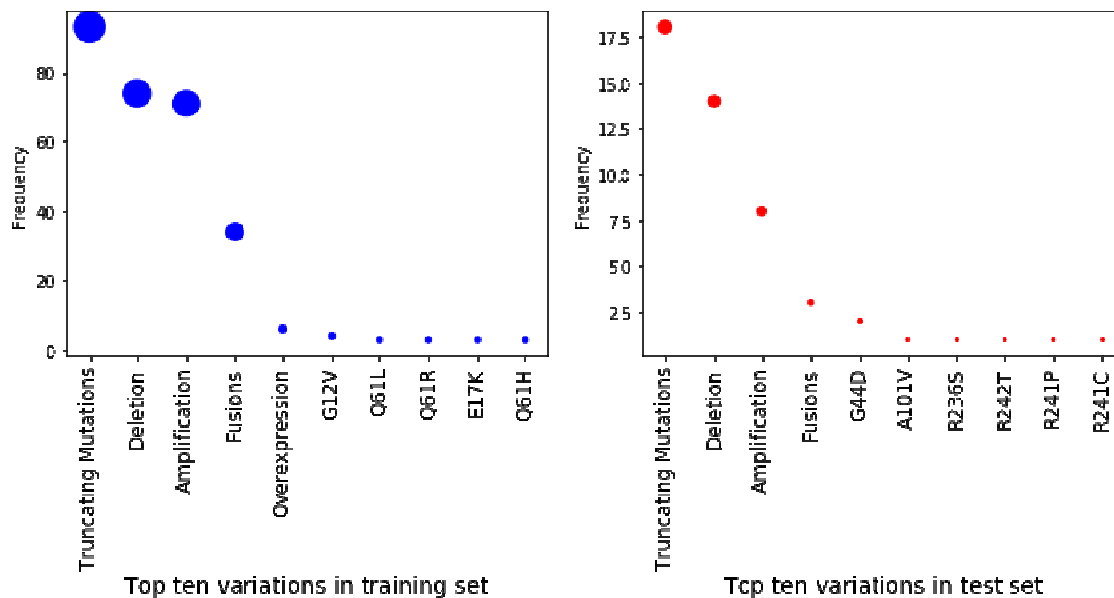
III. Variations overview

Top 10 variations

Variations with maximal occurrences Variation Truncating Mutations 93
Deletion 74 Amplification 71 Fusions 34 Overexpression 6 G12V 4 E17K 3
T58I 3 Q61L 3 Q61R 3 Name: Variation, dtype: int64

Variations with maximal occurrences Variation Truncating Mutations 18
Deletion 14 Amplification 8 Fusions 3 G44D 2 H1464P 1 H12Q 1 H132P 1 H136R
1 H137L 1 Name: Variation, dtype: int64

Top 10 variations' scatter plot



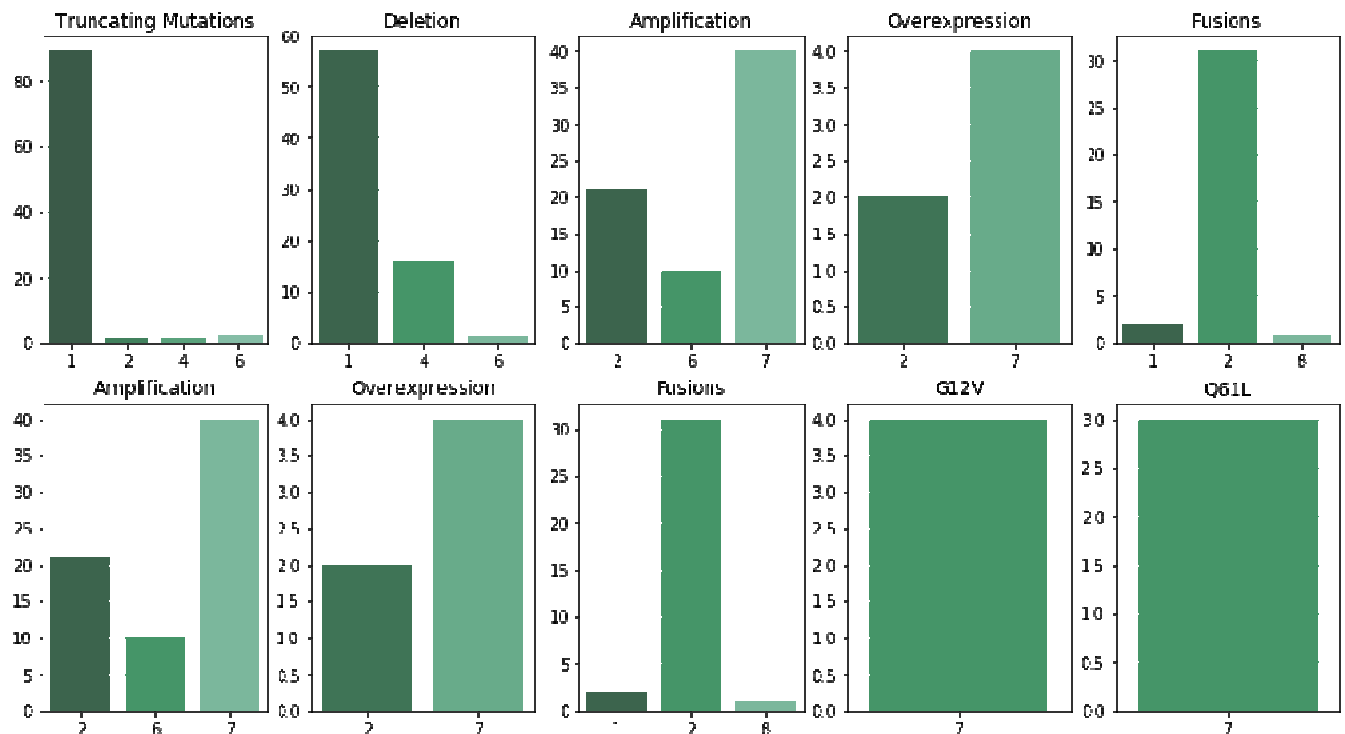
Top four variations are same in train and test data set. They are Truncating mutations, Deletion, Amplification and Fusions.

But the counts of top four variations in training data are much higher than that in test data.

It shows the variation distributions in training data and test data are of much difference.

IV. Variation and class

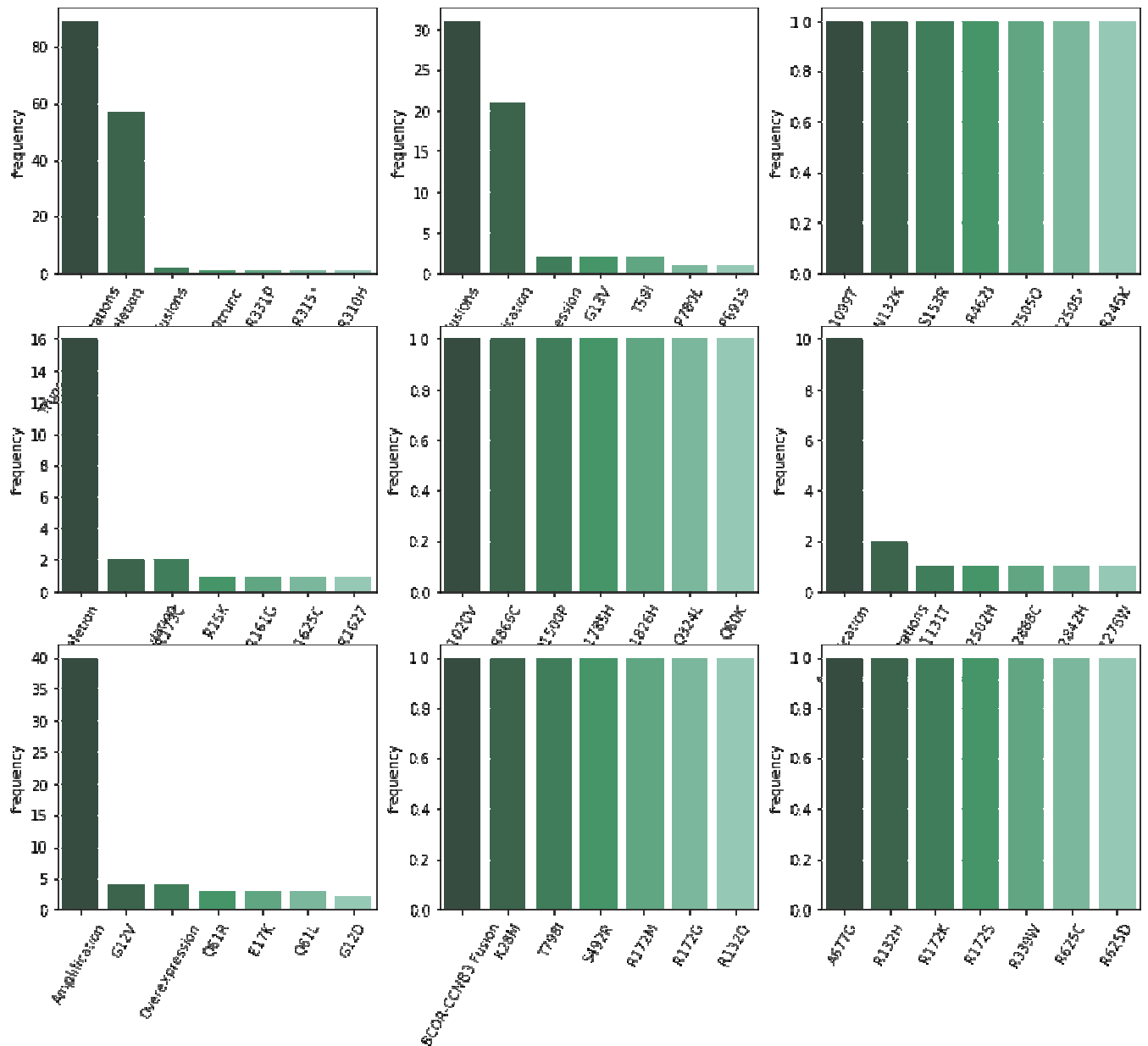
Top 10 variations' class distribution



Top one and top two variations are Truncating mutations and Deletion. They are both located in Class 1, 4 and 6. Truncating mutations is also in class 2.

Top three and top four variations are Amplification and Fusions. They are both located in Class 2 and 7. And Amplification is also in class 6.

Variation distribution in different classes



Class 3, 5, 8, 9 the maximum gene variation is 1.

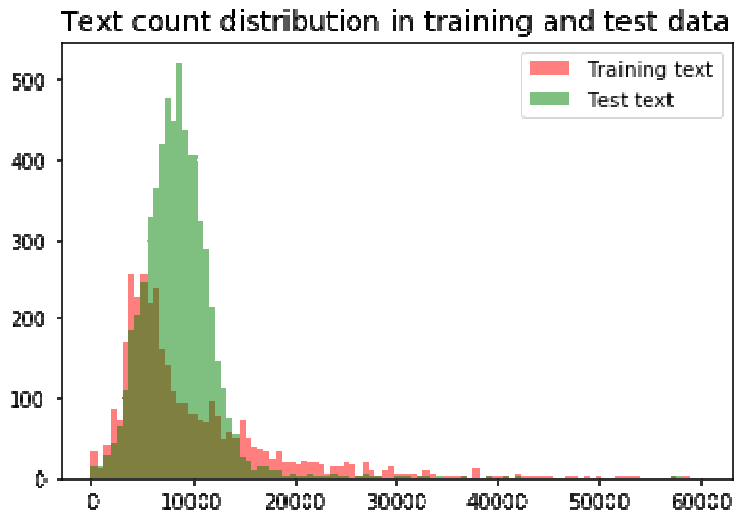
Truncating mutations ranks first in class 1 and second in class 6 (class location 1, 2, 4, 6).

Deletion ranks first in class 4 and second in class 1 (class location 1, 4, 6).

Amplification ranks first in class 6 and 7. And it ranks second in class 2 (class location 2, 6, 7).

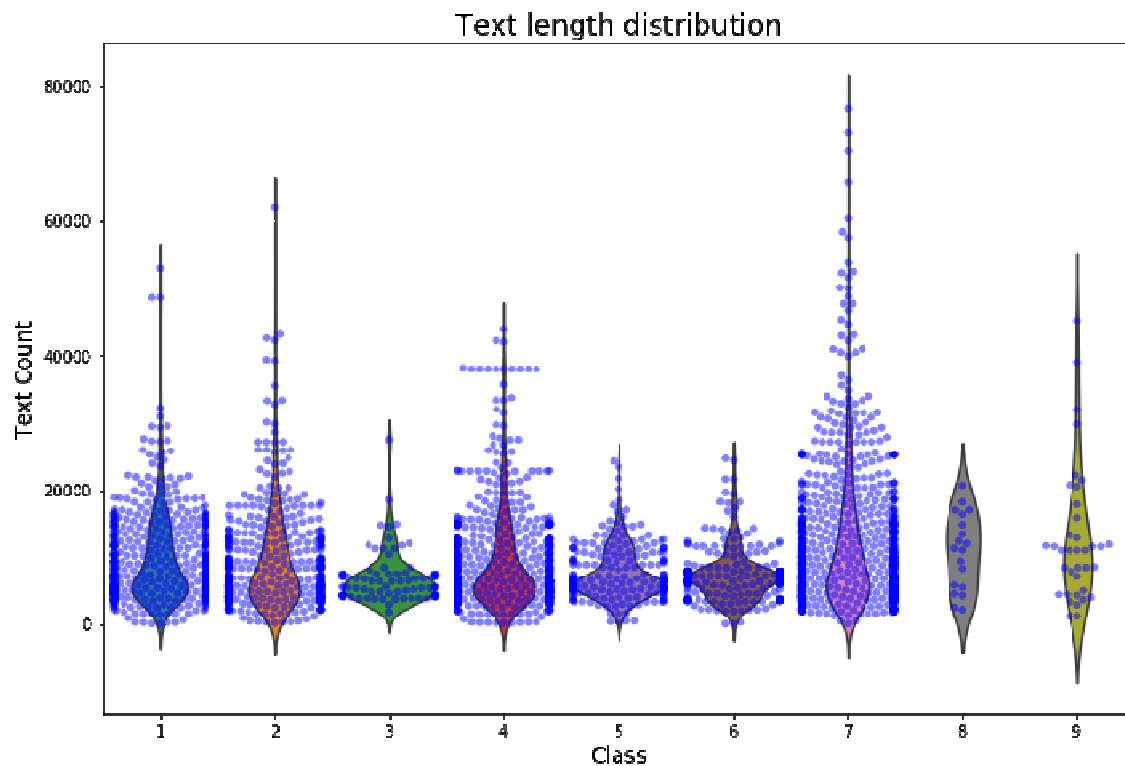
Fusions ranks first in class 2 (class location 2, 7).

V. Text length overview



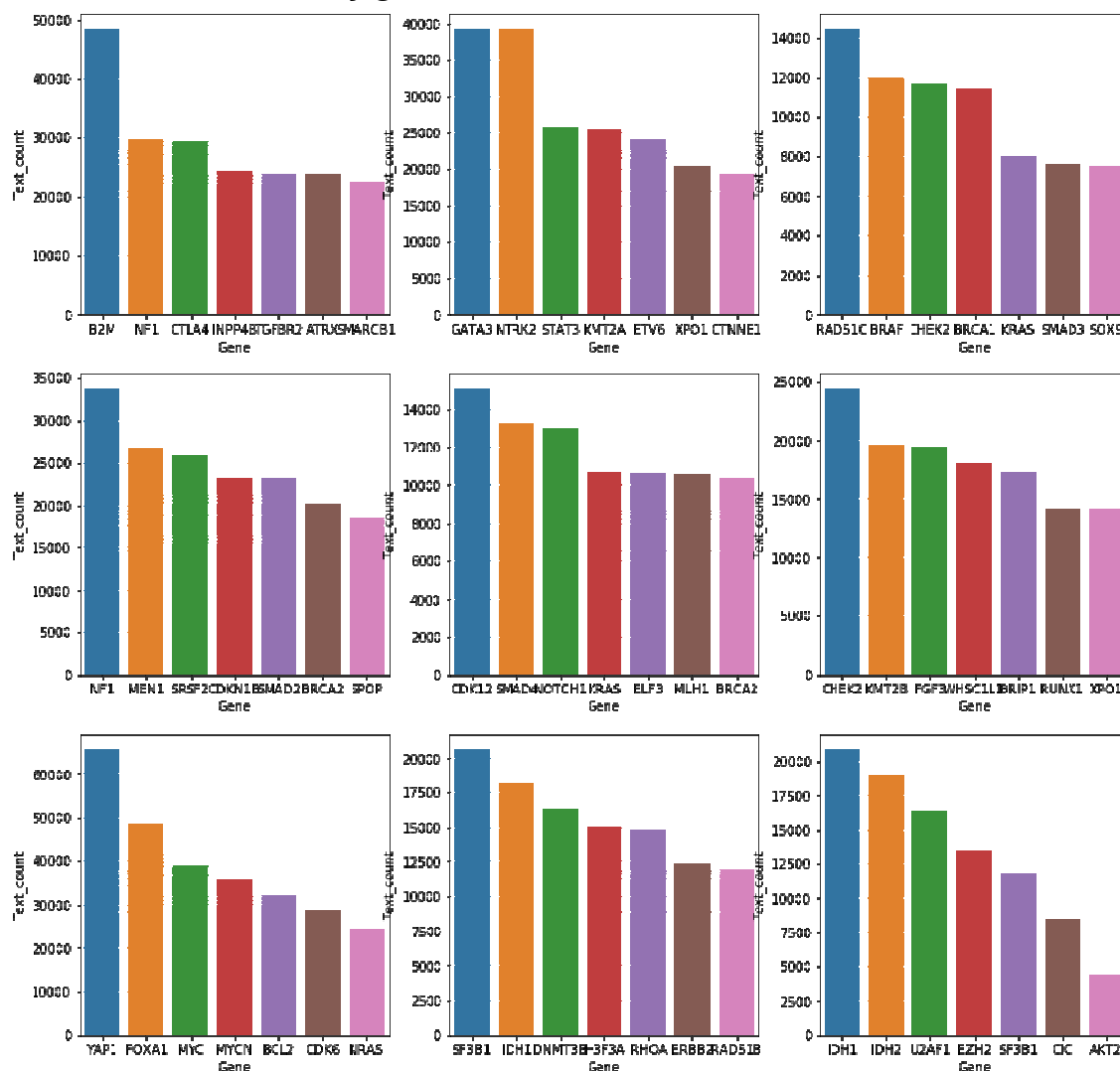
The most frequent text count in training set is about 5000 while that in test set is about 10000. The text amount distribution in both training and test data set are close to normal distribution.

VI. Text length distribution by class



Text count in Class 7 ranges from 1 to nearly 80000. It is the largest range among the nine classes. Class 8 and 9 have the smallest ranges. The text count distribution is similar to the gene occurrence distribution in different classes.

VII. Mean text count by genes in different classes



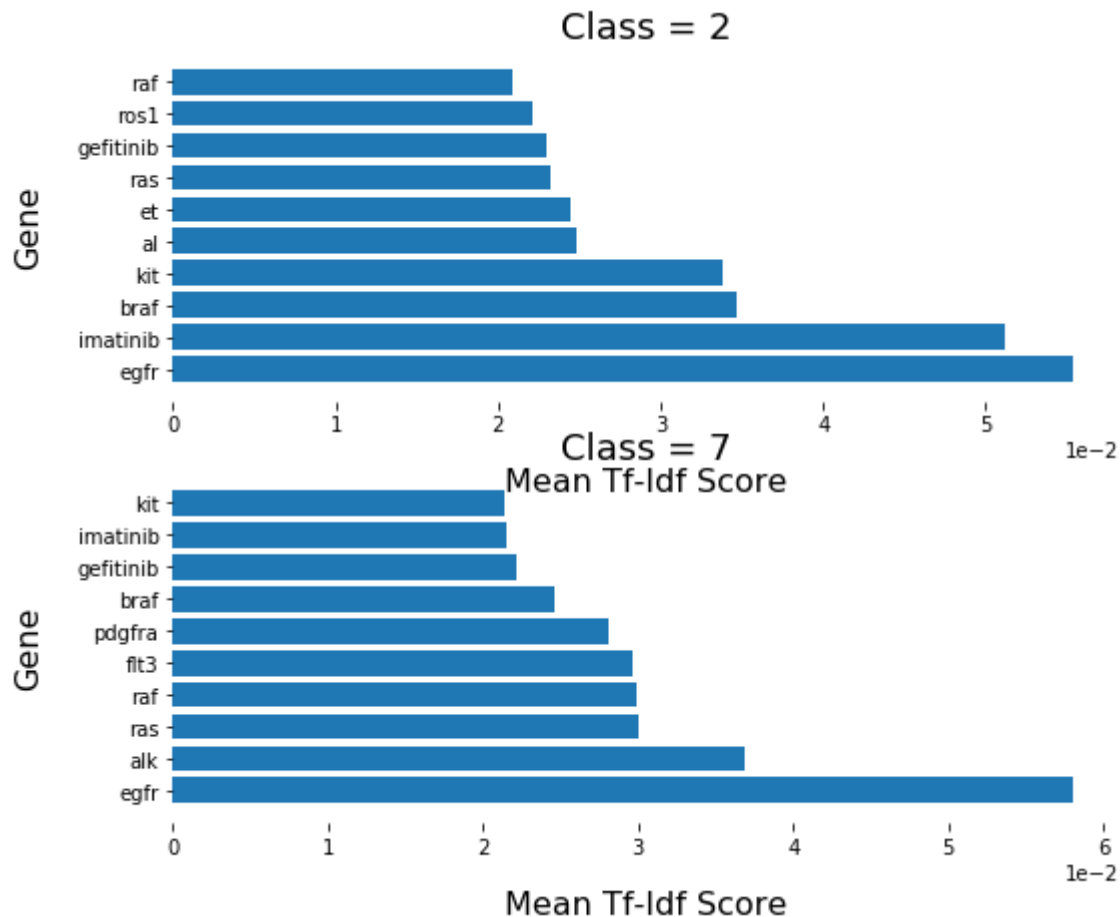
This plot shows the top seven mean text count for genes in different classes. BRCA1 and BRCA2 are among the top ten occurred genes. And they are also in the top seven text count genes in some classes.

VIII. Top 10 TF-IDF features in class 2 and 7

Gene distribution analysis shows top one gene EGFR ranks first in class 2 and 7. Variation distribution analysis shows Amplification and Fusions (top 2 variations) mainly located in class 2 and 7. What's more, Amplification ranks first in class 7 and Fusions ranks first in class 2.

The above fact give us an impression that class 2 and class 7 are highly similar in both gene occurrence and variation distribution. What are about their top 10 text features?

Tf-idf is known as one good technique to use for text transformation and get good features out of text for training our machine learning model. Let's see the top 10 text features in class 2 and class 7.



There are seven common text features in the two classes. There are 70% similarity for the top 10 text features between class 2 and class 7! But the common text features' value are different in the two classes.

For example, word 'egfr' contributes as a strong indicator for both classes 2 & 7 and its value does not change much between them - it adds only a little higher score for class 7. While word 'braf' TF-IDF weights is 3.5 in class 2 and that of class 7 is about 2.5. It means 'braf' contributes more to class 2 than to 7.

Base model building without text processing

In the basic model we used variable 'Gene' and 'Variation' as predictors. There are nearly three thousands categories of variation and we select the variations which count more or equal to 5. With the logistic regression classifier we get test accuracy is 0.93.

Test accuracy: 0.928571428571

Variation		ID
2626	Truncating Mutations	92
473	Deletion	74
149	Amplification	70
807	Fusions	34
1724	Overexpression	6
844	G12V	4

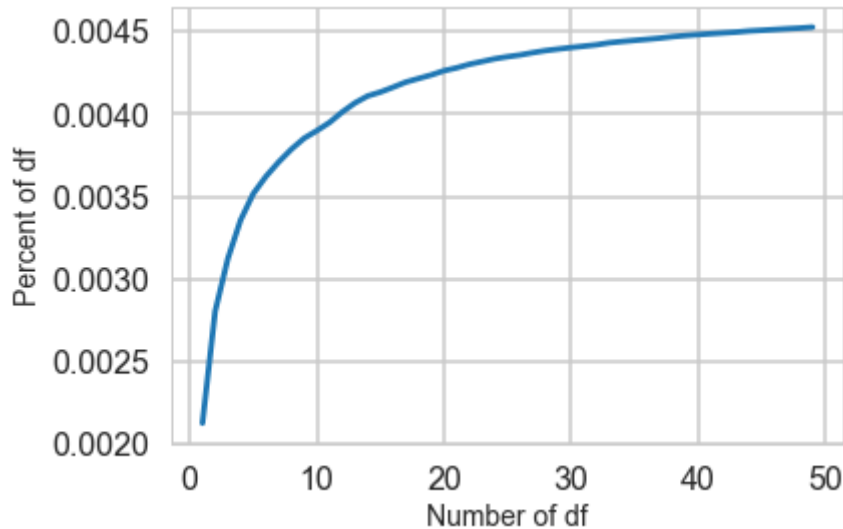
Text processing and Model building

Bag of words and TF-IDF

In information retrieval, TF-IDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling.

Term frequency: the number of times a term occurs in a document is called its term frequency. Inverse document frequency: in simple terms, it's a measure of the rareness of a term. Conceptually, we start by measuring document frequency.

Getting word vector



Elbow method to estimate min_df is about 4 or 5.

```
features length of counter vectorizer: 46288  
features length of TF_IDF vectorizer: 131056
```

Model building

Naive Bays without tuning

Counter vector:

```
Cross validate log loss 13.53  
Cross validate accuracy 0.60
```

TF-IDF vector:

```
Cross validate log loss 1.74  
Cross validate accuracy 0.52
```

The accuracy of Naive Bays with Counter vector is 0.60 and higher than that of model with TF-IDF. But the log loss of TF-IDF is 1.74 and far less than that of model with Counter vector. It means TF-IDF has less false classification. As for the accuracy we can tune the hyperparameter to increase it.

TF-IDF and Naive Bays tuning

```
[('tfidf', TfidfVectorizer(analyzer='word', binary=False,  
decode_error='strict', dtype=<class 'numpy.int64'>, encoding='utf-8',  
input='content', lowercase=True, max_df=0.25, max_features=None,  
min_df=4, ngram_range=(1, 2), norm='l2', preprocessor=None,  
smooth_idf=True, stop_words='english', strip_accents=None,  
sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None,  
use_idf=True, vocabulary=None)), ('clf',  
OneVsRestClassifier(estimator=MultinomialNB(alpha=0.001,  
class_prior=None, fit_prior=True), n_jobs=1))]
```

```
Features length of TF-IDF vectorizer: 600802
Cross validate log loss 6.51
Cross validate accuracy 0.62
```

Using the tuned hyper parameters, Naive bays with TF-IDF model accuracy in validate set increased from 0.52 to 0.62. Great increase! The tuned Naive Bays with TF-IDF has higher accuracy and lower log loss than the model with Counter vector. So in the following model building we will choose TF-IDF as the way we get word vector.

Confusion matrix of Naive bays with TF-IDF model in test data set

Test accuracy: 58.8127853881

Test log loss: 7.09

Confusion matrix:

```
[[107 1  2 32  24 3 12 1 0]
 [ 5 81  2 4   5 0 56 0 0]
 [ 2  0 21 4   5 0 2  0 0]
 [ 51 9 10 121 20 3 9  0 2]
 [ 19 2 4  4   38 7 7  0 0]
 [ 18 2 1  0   6 56 10 0 0]
 [ 3 60 15 2  15 1 204 0 1]
 [ 0 1  0 0  1  1 1  2 2]
 [ 2 0  0 0  1  0 1  0 14]]
```

Test accuracy is 0.58 and is lower than that of cross validate accuracy 0.62. And the log loss of test set is 7.09 and higher than that of cross validate set 6.51. Log Loss quantifies the accuracy of a classifier by penalizing false classifications. Minimizing the Log Loss is basically equivalent to maximizing the accuracy of the classifier.

Confusion matrix shows class 7 is over predicted. The majority over predicted class is class 7. In the top 10 TF-IDF features analysis we can see class 7 and class 2 have 70% similarity in the top 10 features. It may be one of the reasons that there are 57 class 2 mutations predicted as class 7.

Random Forest hyperparameter tuning

```
{'max_depth': 15, 'min_samples_leaf': 10, 'n_estimators': 1000}
features length of TF-IDF vectorizer: 46125
```

Cross validate log loss 1.20

Cross validate accuracy 0.63

Confusion matrix of Random Forest with TF-IDF model in test data set

Test accuracy:

0.595433789954

Test log loss:

1.23

Confusion Matrix:

```
[[ 92  1  0 51 11  0 27  0  0]
 [  8 42  0  9  1  0 93  0  0]
 [  3  0  2 15  1  0 13  0  0]
 [ 37  0  0 162  2  0 24  0  0]
 [ 22  1  1 15 18  4 20  0  0]
 [ 15  2  0 15  0 48 13  0  0]
 [  2  5  1  3  2  0 288  0  0]
 [  3  0  0  1  0  0  4  0  0]
 [  7  0  0  6  0  0  5  0  0]]
```

Test accuracy of Random Forest is 0.60 and is higher than that of Multinomial Naive Bays 0.58. And the log loss of test set is 1.23 and lower than that of Multinomial Naive Bays 7.09. Lower log loss means higher accuracy.

Confusion matrix shows class 1, 4 and 7 are over predicted. And there is zero observations which are predicted as class 8 and 9.

GBM: Tree based hyperparameters tuning

Final GBM model:

```
gbm_tuned_2 = GradientBoostingClassifier(learning_rate=0.01, n_estimators=600, max_depth=9,
                                         min_samples_split=1000, max_features='sqrt',
                                         min_samples_leaf=50, subsample=0.8, random_state=10)
```

Model Report

Test log loss 1.11

CV Score: Mean - 0.6195038 | Std - 0.01467631 | Min - 0.5990991 | Max - 0.6373874

Test Accuracy: 0.6301

Summary:

1. In this part we use count vector and tf-idf to get word vector. Stop words is 'English'.
2. Tf-idf has better performance than the count vector.
3. We tried Naive bays, Random forest and Gradient boost method to build the classifying model. The accuracy of test set is from 0.52 to 0.63. And GBM performs best.
4. The best one hyper parameter combination for GBM is gbm_tuned_2 with learning rate 0.01. Test accuracy is 0.63 and log loss is 1.11.

II. Bag of words and TF-IDF with new stop words

Here I want to collect various text features, artefacts, and global properties that I noticed during this initial exploration. This list will likely expand as the kernel grows.

Scientific terminology and stop words: Most scientific papers have a common style of language that will be reasonably homogeneous throughout the text files. Words like “result” or “discuss” will be frequent without necessarily containing any signal for our prediction goal. Therefore, below I define my own list of additional stop words.

Research field related stop words: My impression is that the list of stop words could be extended by including characteristic terms of the overall research field that are so ubiquitous that their high frequency may mask genuinely interesting terms. Words such as “mutation”, “cancer”, or “tumor” appear to be too general to have much distinguishing power here. The TF-IDF below seems to confirm this. It would be interesting to get some feedback from people with domain knowledge about which other terms could a-priori be removed from the text.

Paper notation quirks: Converting the paper text straight to ascii leads to a number of artefacts. None of those will have a big impact individually, but together they might reduce the accuracy of the analysis:

- Citation numbers (as used e.g. by Nature magazine) are attached to the corresponding word.
- Occasionally, there are what seems like webpage navigation commands like “SectionNext” embedded in the text.
- Author names and affiliations are occasionally included

```
newStopWords = ["fig", "figure", "et", "al", "table", "data", "analysis", "analyze", "study",  
                "method", "result", "conclusion", "author", "find", "found", "show",  
                "perform", "demonstrate", "evaluate", "discuss", "abstract", "background", "university"]
```

```
Features length of counter vectorizer: 46545  
Features length of TF_IDF vectorizer: 46301
```

Multinomial Naïve Bays with Counter vector

```
Model Report  
Test log loss 1.24  
CV score: Mean - 0.6082655 | Std - 0.01442368 | Min - 0.5968468 | Max  
- 0.6351351  
Test accuracy: 0.5918
```

Multinomial Naïve Bays with TF-IDF vector

```
Model Report  
Test log loss 1.23  
CV score: Mean - 0.6145831 | Std - 0.01220539 | Min - 0.5945946 | Max  
- 0.6283784  
Test accuracy: 0.5963
```

Random Forest with TF-IDF vector

Model Report

Test log loss 1.23

CV score: Mean - 0.6118925 | Std - 0.009162869 | Min - 0.5990991 | Max - 0.6261261

Test accuracy: 0.5918

GBM with TF-IDF vector

Model Report

Test log loss 1.10

CV score: Mean - 0.6226185 | Std - 0.02310286 | Min - 0.5918367 | Max - 0.6531532

Test accuracy: 0.6228

Summary

1. In this part we add some new stop words to 'English'. They are commonly used in publications.
2. We also tried Naive Bays, Random Forest and GBM. The test accuracy is from 0.59 to 0.62 and GBM has still the best performance.
3. Although new defined stop words decreased features of the TF-IDF from 131056 to 46301, the model performance is the same as models using 'English' as stop words.

III. Truncated SVD

Truncated SVD performs linear dimensionality reduction by means of truncated singular value decomposition (SVD). Contrary to PCA, this estimator does not center the data before computing the singular value decomposition. This means it can work with sparse matrices efficiently.

In particular, truncated SVD works on term count/tf-idf matrices as returned by the vectorizers in `sklearn.feature_extraction.text`. In that context, it is known as latent semantic analysis (LSA).

Start with simple Truncated SVD

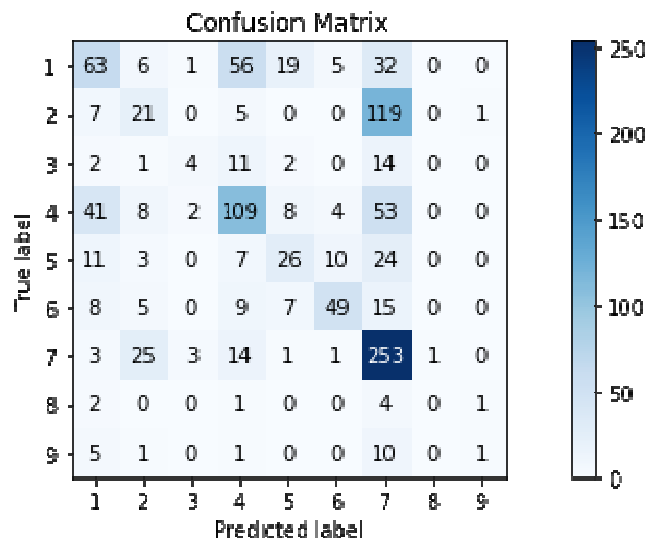
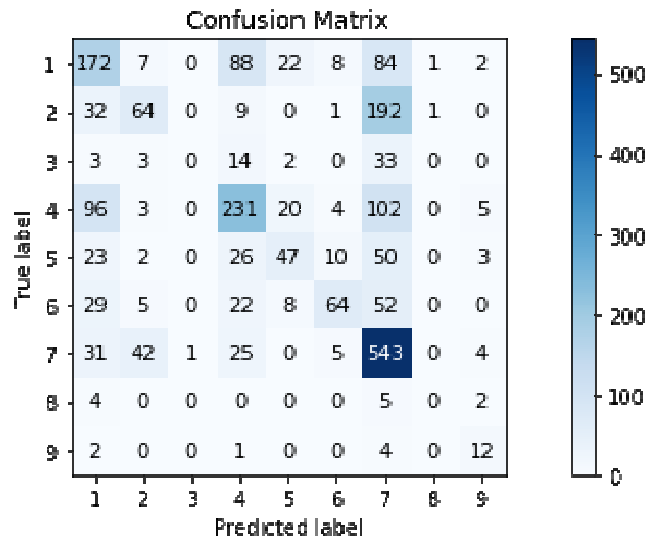
Logistic regression with Truncated SVD

Log loss of training set: 1.418979049362458

Accuracy of training set: 0.510130571814498

Log loss of test set: 1.538822587603099

Accuracy of test set: 0.480365296803653



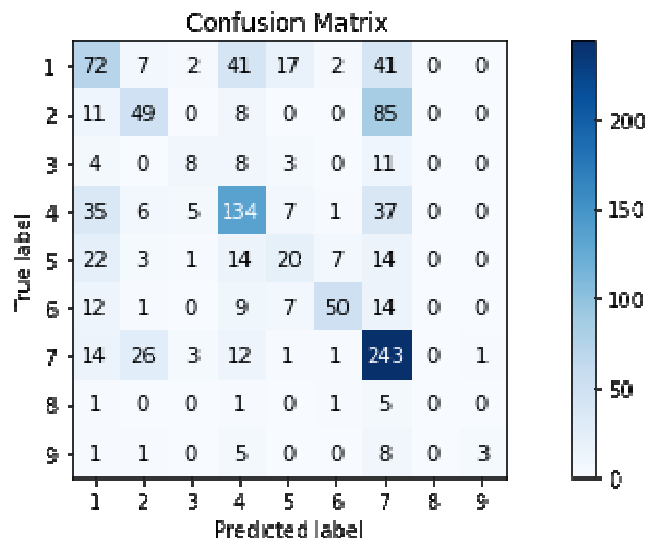
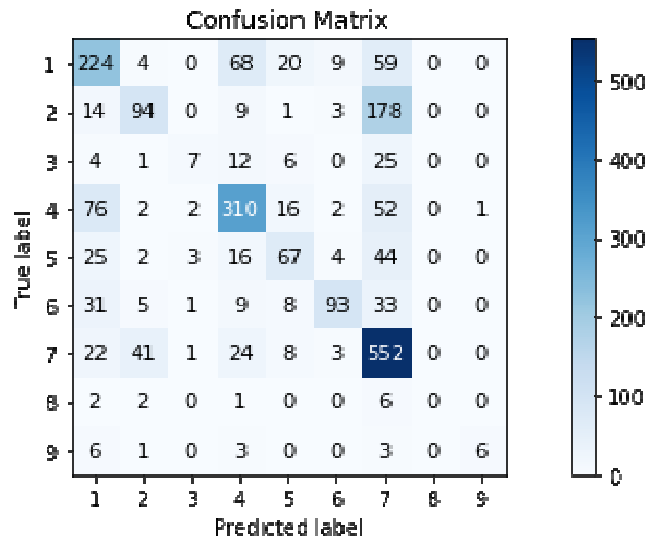
Random Forest with Truncated SVD

Log loss of training set: 1.1823431227614996

Accuracy of training set: 0.6091850517784781

Log loss of test set: 1.326395708210827

Accuracy of test set: 0.5287671232876713



Random Forest model is better than the logistic regression model both in log loss and model accuracy.

Truncated SVD and TF-IDF

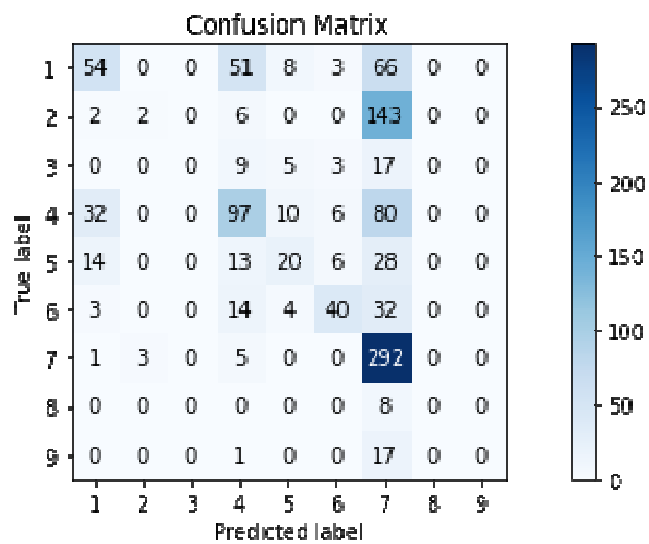
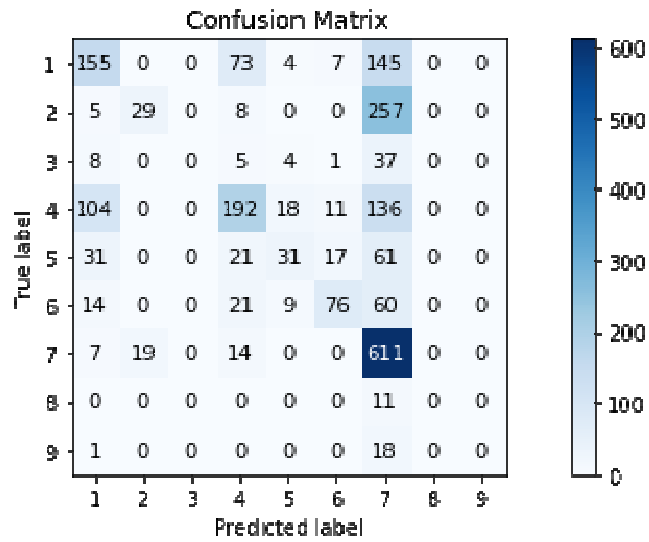
Logistic regression:

Log loss of training set: 1.396008210084452

Accuracy of training set: 0.4925709140027015

Log loss of test set: 1.515921573148094

Accuracy of test set: 0.4611872146118721



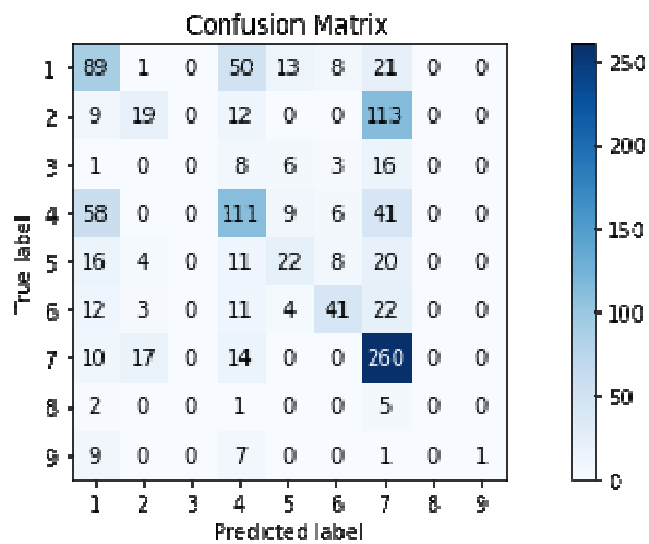
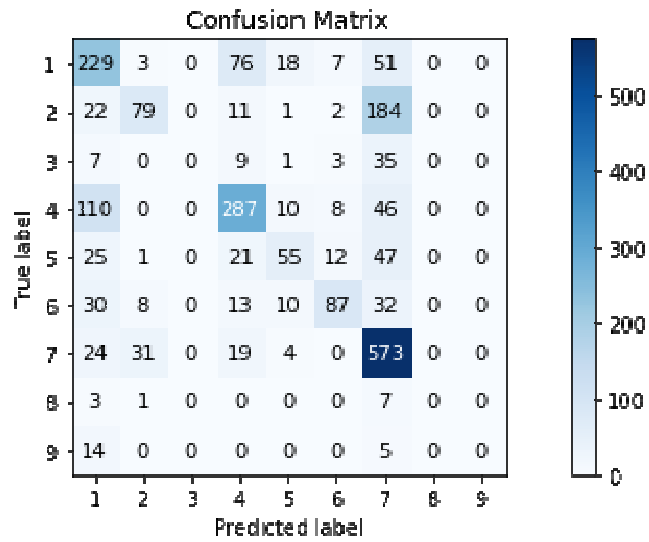
Random Forest:

Log loss of training set: 1.1506099595322337

Accuracy of training set: 0.589824403421882

Log loss of test set: 1.3364489293258033

Accuracy of test set: 0.4958904109589041



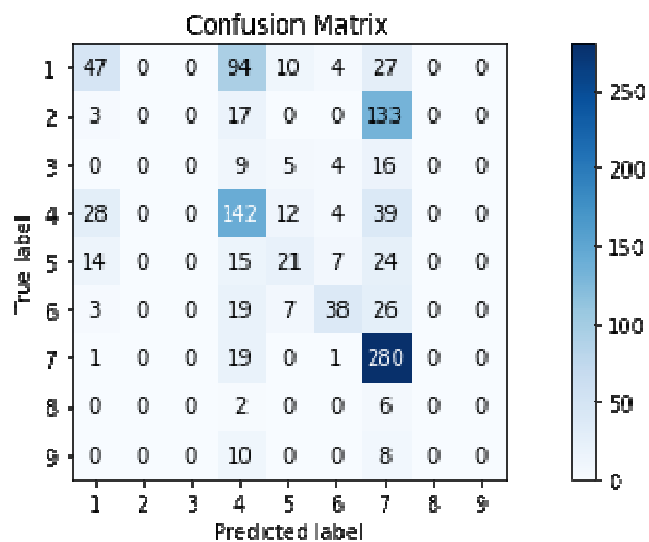
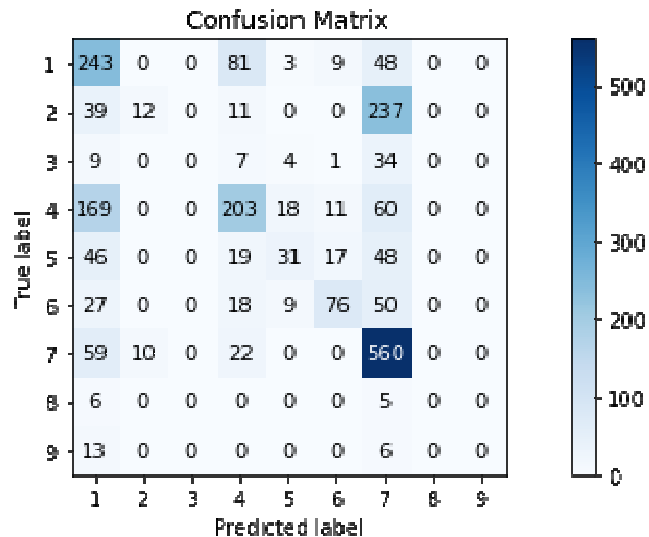
Support Vector Machine

Log loss of training set: 1.350459083584947

Accuracy of training set: 0.5065285907248986

Log loss of test set: 1.4314985327328322

Accuracy of test set: 0.4821917808219178



Summary:

In this part Truncated SVD with count vectorizer performs better than that with TF-IDF. But Truncated SVD with count vectorizer didn't perform better than the model without Truncated SVD.

IV. Word2vec

This time, let's try the popular word2vec to get features.

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

In many cases traditional text classification can be difficult to scale, because as the order of the taxonomy count increases, the amount of training required increases as well. Moreover, with taxonomy counts in the thousands or tens of thousands, it can become increasingly expensive to gather a sufficient volume of labeled text examples for each taxonomic class.

One solution to this problem is to move to Word2Vec for the processing of your unstructured text data. Word2Vec (W2V) is an algorithm that takes every word in your vocabulary—that is, the text you are classifying—and turns it into a unique vector that can be added, subtracted, and manipulated in other ways just like a vector in space.

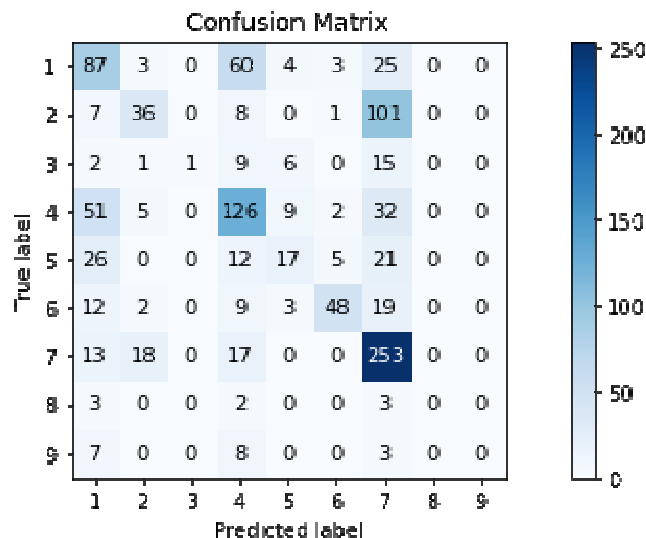
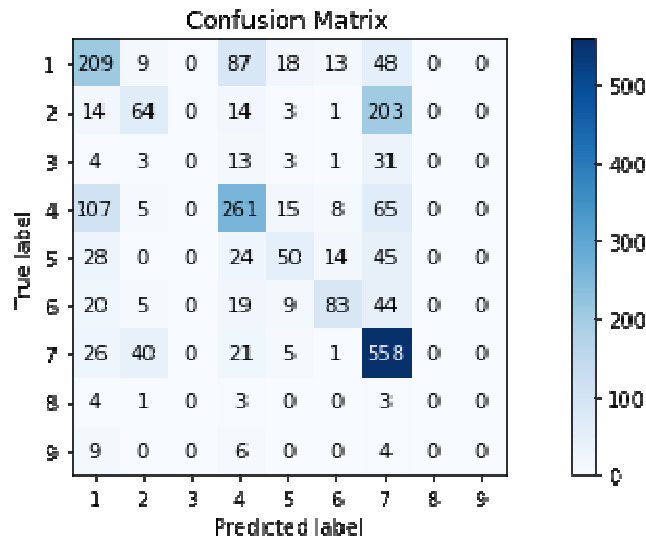
Logistic regression:

Log loss of training set: 1.2508235971322528

Accuracy of training set: 0.5515533543448897

Log loss of test set: 1.3665064056669463

Accuracy of test set: 0.5187214611872146



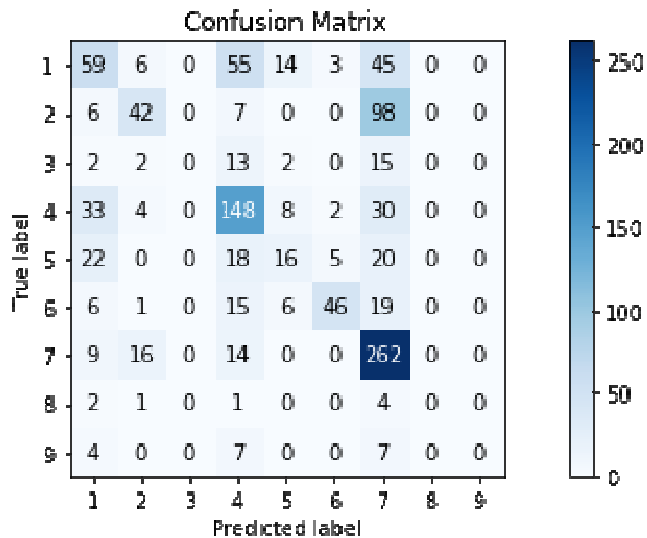
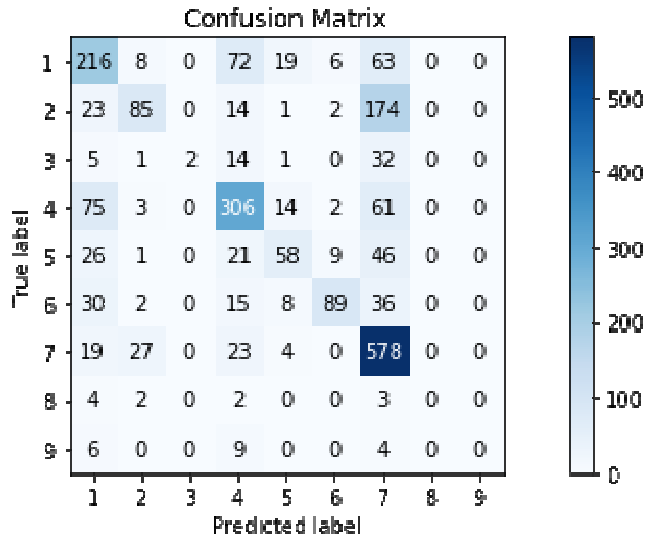
Random Forest:

Log loss of training set: 1.2043050604945094

Accuracy of training set: 0.6006303466906798

Log loss of test set: 1.3665889524368535

Accuracy of test set: 0.523287671232876



As expected, we get better results than Truncated SVD with count vectorizer/TF-IDF. But the Random forest accuracy of word2vector is 0.52 while Random forest model with only TF-IDF in the first part is 0.54.

So the results are still not very good though. One way to explain this is that there is a lot of information loss from just getting the mean of all word vectors of the document. This is roughly analogous to taking the entire document, summarizing it into one word, and using that word to classify the entire text.

V. LSTM

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Let's try a quick and dirty LSTM in Keras to take into account the sequential nature of text

- We won't do any hyperparameter search

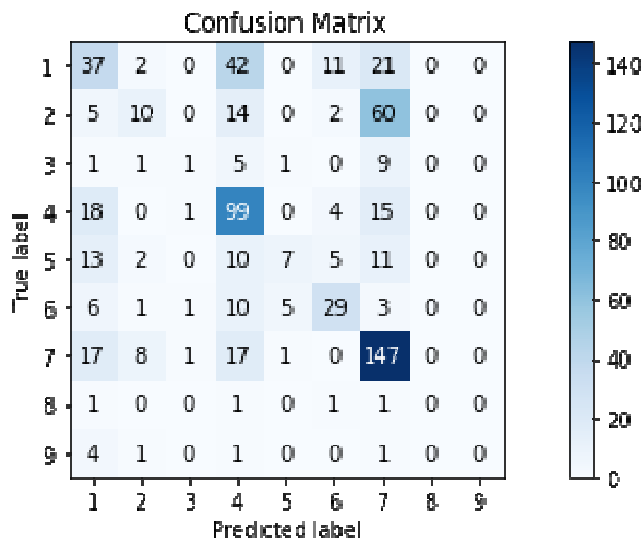
- We'll go with 15 epochs, and save the model with the best validation loss after an epoch

- Max sequence length is cut down to a measly 2000 (longest text has 77000+ words), to shorten training time

Keras Model:

Log loss: 1.3737221723955393

Accuracy: 0.49698795180722893



The dirty LSTM model's accuracy in test data set is 0.49. And the results of the quick LSTM are promising.

On the first try with no hyperparameter search, 6th epoch, max sequence length cut down to a measly 2000 (longest text has 77000+ words), we get the best log loss so far of around 1.37.

Further tuning of the LSTM will likely produce better results.

Summary

In this project we applied EDA techniques on the genes, variations and text variables for both training and test. It has been showed that the gene, variation and text in training set are in differently distributed as compared to the test set. And text variables are imbalanced in different classes.

If we use down sampling method to balance the text data in each class we would lose too much data for the count of the lowest class is too few. And advanced techniques like SMOTE were out of the scope of the project. So we keep the original sample.

First we only used variable "gene" and "variation" as predictors to classify the classes. And we got a random forest model with accuracy high to 0.93. For the rest part of project we only "text" as predictor for classification. Different word vectorization techniques like Counter vector, TFIDF, Truncated SVD, word2vec and LSTM are applied. GBM with TFIDF got highest model accuracy about 0.63.

It is 30 percent lower than that of basic model using 'gene' and 'variation'. As we mentioned in the first paragraph the text variable's distribution are various in training and test data set. While the basic model used part of training set to build the model and got the accuracy from validate set (another part of the training set). The gene and variation distribution are very similar in training and validate set. It will be the main reason why basic model has much higher accuracy than the text model.

Looking closer at the statistics we calculated above, "training_text" actually has duplicates, and the duplicates have different classes. This is part of the challenge. A lot of papers are studies of 2 or more genes. It is our future job to use the other fields to figure out which parts of the text are relevant for the particular Gene and Variation.

Future job

There is still a long way for the automatic classification of cancer gene/variation classification. The following steps will be helpful for it:

1. Trying to find better way to get word vector.
2. Normalizing text data. For example, deleting the duplicate parts, citing only the abstract part of the paper.