Project Overview

In the past several years, researchers have been trying new ways to treat cancer with precision medicine and genetic testing customized to the specific patient. However, analyzing genetic information is a very time-consuming task. A cancer tumor can have thousands of different mutations, and the clinical pathologist must manually review every single genetic mutation by reading long pages of clinical literature. Therefore, Kaggle.com has collaborated with Memorial Sloan Kettering Cancer Center (MSKCC) to create this competition to find a machine learning algorithm that can automatically classify genetic variations. [1]

Problem Statement

The goal of the project is to have a working python script that will read the data provided and result in a multi class log loss between the predicted and the observed target, as requested by Kaggle. The tasks are as followed:

1. Download and explore data
2. Do NLP analysis on the clinical evidence
3. Find underlying relations between genes and their variations
4. Feature Engineering
5. Reduce dimensions of both categorical and text data
6. Train a classifier to learn then predict the multi class log loss
7. Parameter tuning
8. Compare with other Kaggle competition kernels as benchmarks

Metrics

Kaggle has specifically requested for a multi class log loss as the evaluation metric since this is a multiclass classification tasks. Log loss penalizes estimators with false classification. The smaller the log loss value, the higher the accuracy of the estimator.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log p_{ij}$$

[2]

N: Number of samples, M: Number of labels, yij: T/F for whether j is correct label for instance i, pij: Probability of assign label j to instance i.

In addition to log loss, this model will be evaluated on the performance of maximizing True Positives (Accuracy, guessing the correct class) while minimizing False Positives (Misclassification).

Accuracy: True Positive + True Negative / Total

Misclassification: False Positive + False Negative / Total

Data Exploration

Datasets provided by Kaggle:

3321 training examples:

```
Data columns (total 5 columns):
ID           3321 non-null int64
Gene         3321 non-null object
Variation    3321 non-null object
Class        3321 non-null int64
Text         3321 non-null object
```

5668 testing examples:

```
Data columns (total 4 columns):
ID           5668 non-null int64
Gene         5668 non-null object
Variation    5668 non-null object
Text         5668 non-null object
```

Proportion of the unique cases:

```
Out of 3321 training cases, there are:
264 unique Genes
2996 unique Variations
1921 unique Texts

Out of 5668 testing cases, there are:
1397 unique Genes
5628 unique Variations
5611 unique Texts
```

Example of the first 10 observations:

	ID	Gene	Variation	Class	Text
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...
5	5	CBL	V391I	4	Oncogenic mutations in the monomeric Casitas B...
6	6	CBL	V430M	5	Oncogenic mutations in the monomeric Casitas B...
7	7	CBL	Deletion	1	CBL is a negative regulator of activated recep...
8	8	CBL	Y371H	4	Abstract Juvenile myelomonocytic leukemia (JM...
9	9	CBL	C384R	4	Abstract Juvenile myelomonocytic leukemia (JM...

**Repeating text maybe due to one document containing two variations of the same gene

Exploratory Visualization

Fig 1. Plot shows the distribution of each mutation class's and occurrences in the training data. Most of the cases result in class 7, 4, and 1, while there are few cases of class 3, 9 and 8.

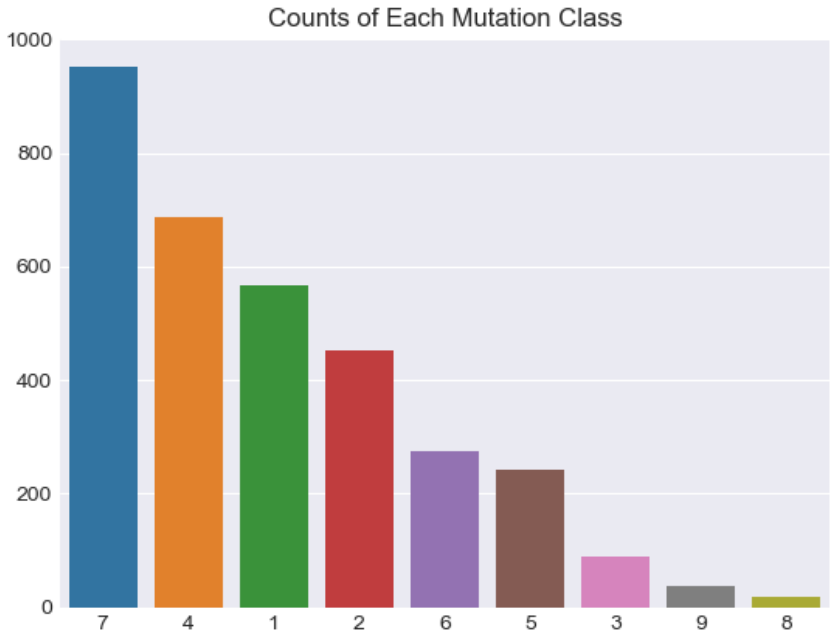
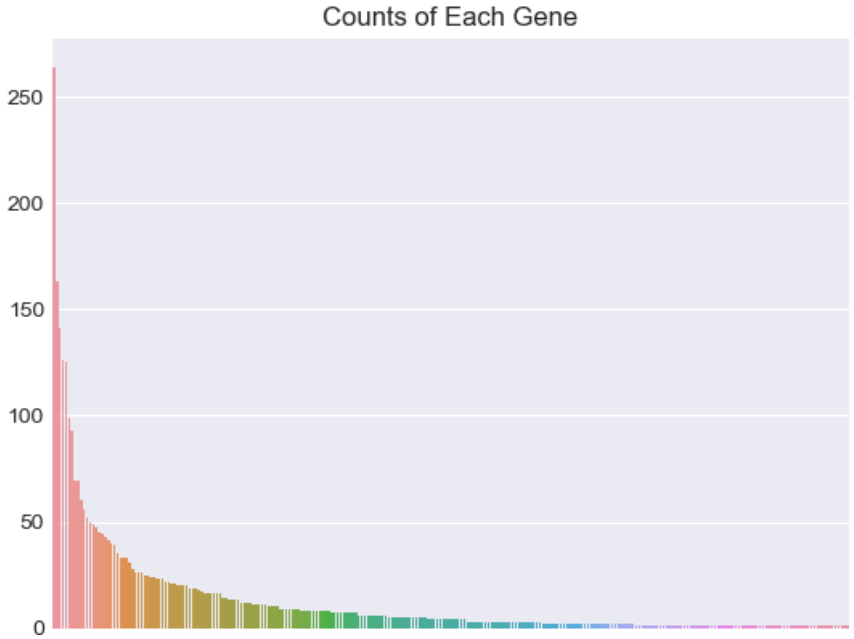


Fig 2. Plot shows the occurrences of the genes, BRCA1 is the most common gene, out of the 264 unique genes.



BRCA1	264
TP53	163
EGFR	141
PTEN	126
BRCA2	125
KIT	99
BRAF	93
ERBB2	69
ALK	69
PDGFRA	60
PIK3CA	56
CDKN2A	52
FGFR2	50
FLT3	49
TSC2	47

Fig 3. Shows first and last 15 of variations sorted. Most occurs 1-2 times only.

Truncating Mutations	93	S502T	1
Deletion	74	G719D	1
Amplification	71	C24Y	1
Fusions	34	Y791F	1
Overexpression	6	D84H	1
G12V	4	C125S	1
T58I	3	P577S	1
Q61H	3	I744_K745delinsKIPVAI	1
Q61R	3	V561A	1
E17K	3	V1398D	1
Q61L	3	S170R	1
R841K	2	N2113S	1
G35R	2	H701P	1
T73I	2	E281K	1
Q22K	2	R1699Q	1

Note: In a kernel [3], provided by Ekaterina Avdeeva, she has proved that putting 'Gene' and 'Variation' together will create 8989 unique cases with max counts of 1 each, making each combination unique.

Fig 4. Shows unique combinations of Gene and Variation proof

In [4]:
```
dfSum.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8989 entries, 0 to 5667
Data columns (total 5 columns):
Class                3321 non-null float64
Gene                 8989 non-null object
ID                   8989 non-null int64
Variation            8989 non-null object
Gene_And_Variation   8989 non-null object
dtypes: float64(1), int64(1), object(3)
memory usage: 421.4+ KB
```

There are 3321 values for 'Class' (training set)

and 8989 values for all other features (training + test sets)

Fig 5. Plots shows the distribution of the document's number of words. Most have around 10000 words, but there are a few with more than 60000 words.

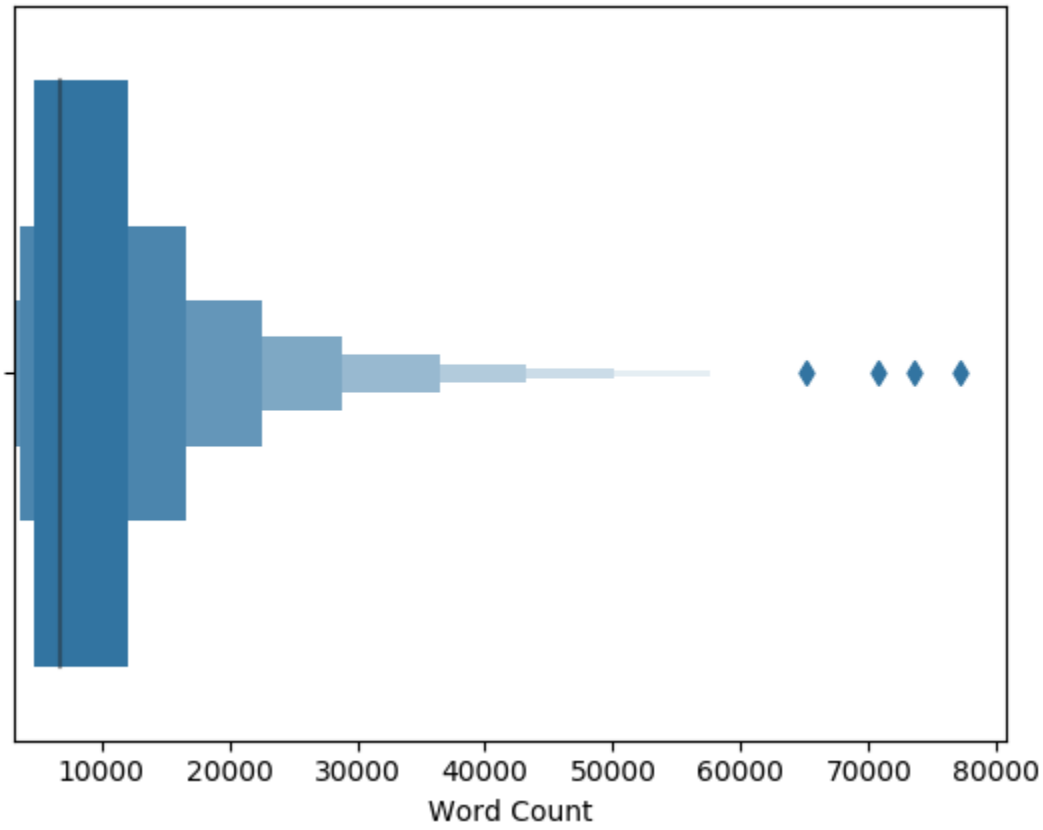
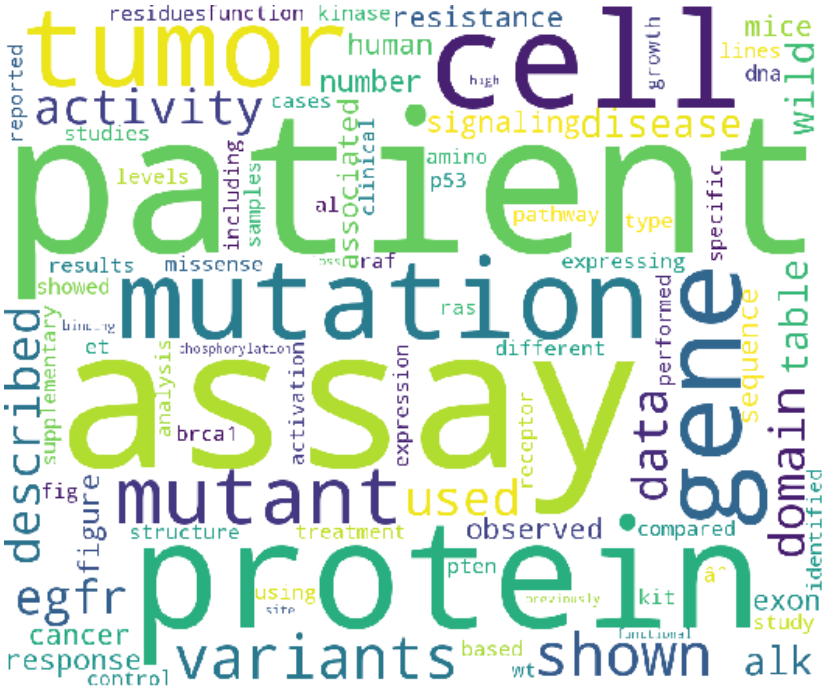


Fig 6. Shows Word Cloud of the most occurring but useful words found in the training set



Algorithms and Techniques

Text data:

- TF-IDF (Term Frequency, Inverse document frequency): This transform the documents into a term document matrix with terms and their term importance to the document for each observation.

- Truncated SVD (Singular Value Decomposition): This is also called LSA (Latent semantic analysis) when applied to term document matrices. This reduces the text by grouping each word into components based on explained variance.

Categorical data:

- Get Dummies: Turns categorical data into columns with binary dummy data.
- MCA (Multiple Correspondence Analysis): This is a dimensionality reduction for categorical variables. This finds the underlying relationship between categorical variables by grouping them into factors.

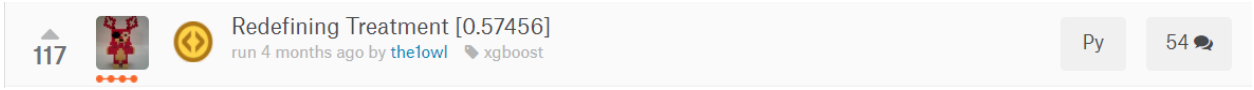
Classifier:

- XGBoost is a Kaggle competition winning supervised learner that is an extension to Gradient Boosting Machine, which is an ensemble tree method.

Benchmark

The bench mark for performance that this project will be placed against is other top kernels in this Kaggle competition. The log loss score will be used.

Fig 7. Log loss result for kernel: 1.0627

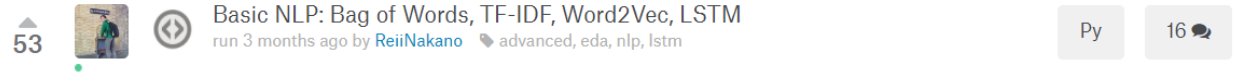



```
Stopping. Best iteration:
[425]   train-mlogloss:0.446482 valid-mlogloss:1.06274

1.06274188304
```

[4]

Fig 8. Log loss result for kernel with xgb: 0.9653



Public LB Score Log Reg: 1.032000

Public LB Score XGB: 0.96536

Public LB Score SVC: 0.97059

[5]

Data Preprocessing

Text data transformation:

- TF-IDF (Term Frequency * Inverse Document Frequency):

$$\text{tf-idf}(t) = \text{tf}(t, d) \times \text{idf}(t)$$

[6]

  - Get term frequency, remove stop words, then index each sentence's special words' occurrences by alphabetical order

Fig 9. Shows example of indexing two sample sentences' special words and its frequency.

activating	0
activity	1
cdk10	2
cdks	3
cellular	4
cyclin	5
dependent	6
fundamental	7
identified	8
kinase	9
kinases	10
orphan	11
processes	12
regulate	13
revealed	14
stands	15
variety	16

```
[[0 0 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1]
 [1 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0]]
```

"Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes"

activating	0
activity	1
cdk10	2
cdks	3
cellular	4
cyclin	5
dependent	6
fundamental	7
identified	8
kinase	9
kinases	10
orphan	11
processes	12
regulate	13
revealed	14
stands	15
variety	16

"CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed"

  - Get Inverse document frequency: log (total number of documents / 1 + number of document the term appeared in) of each term

$$\text{idf}(t) = \log \frac{|D|}{1 + |\{d : t \in d\}|}$$

[6]

Fig 10: Shows calculated IDF value of each word on the list by index from Fig 9.

```
print(tfidf.idf_)
```

```
[ 1.40546511  1.40546511  1.40546511  1.          1.40546511  1.
  1.40546511  1.40546511  1.40546511  1.40546511  1.40546511  1.40546511
  1.40546511  1.40546511  1.40546511  1.40546511  1.40546511]
```

- Get Product of Term Frequency * Inverse Document Frequency

Fig 11. Shows the product of TF matrix (2 samples x 17 features), and IDF matrix (1 sample x 17 features).

```
print(matrix.todense())
```

```
[[ 0.          0.          0.          0.25136004  0.35327777  0.25136004
   0.35327777  0.35327777  0.          0.          0.35327777  0.
   0.35327777  0.35327777  0.          0.          0.35327777]
 [ 0.33310232  0.33310232  0.33310232  0.23700504  0.          0.23700504
   0.          0.          0.33310232  0.33310232  0.          0.33310232
   0.          0.          0.33310232  0.33310232  0.        ]]
```

- Latent Semantic Analysis on the TF-IDF matrix with SVD (Singular Value Decomposition)

$$X_k = U_k \Sigma_k V_k^T$$ [7]

The decomposition of Matrix X (TF-IDF Matrix) to dimension K is made by the product of these 3 matrices:

U: X * X-transposed: into 2x2 matrix (2 document and 17 words' TF IDF scores summed into 2 vectors)

∑: 2 x 2 Matrix of Singular Values (2 singular values corresponding to 2 documents)

V^T: X-transposed * X: matrix 2x17 matrix of dot product of the documents

Fig 12. Shows the product of U, Sigma and V-transposed matrixes

```
U, Sigma, VT = randomized_svd(matrix, n_components = 2, random_state=None)
```

```
U =  [[ 0.70710678  0.70710678]
      [ 0.70710678 -0.70710678]]

Sigma = [ 1.05789753  0.93853759]

VT =  [[ 0.22264813  0.22264813  0.22264813  0.32642695  0.23613355  0.32642695
         0.23613355  0.23613355  0.22264813  0.22264813  0.23613355  0.22264813
         0.23613355  0.23613355  0.22264813  0.22264813  0.23613355]
       [-0.25096374 -0.25096374 -0.25096374  0.01081525  0.2661642   0.01081525
         0.2661642   0.2661642  -0.25096374 -0.25096374  0.2661642  -0.25096374
         0.2661642   0.2661642  -0.25096374 -0.25096374  0.2661642 ]]
```

=

```
print(decomposed)
```

```
[[ 0.74804652  0.66364629]
 [ 0.74804652 -0.66364629]]
```

Figure 9 – 12 shows the transformation of the two sample sentences from text into numbers that represents them in the number of "concepts" chosen, but in the project, each clinical literature will behave as single sentence as the example.

- In Fig 9, each sentence is transformed into vector space by their special words occurrence in the position of the sentence, then each word is listed in the "corpus" alphabetically. The sentence is represented by the index of the special words it contains from the corpus.
- In Fig 10, the IDF value of index 3 (CDK) and 5 (Cyclin) have a value of 1 because of their appearance in both sentences, making it comparatively more common/less special than the rest of the words in the corpus.
- In Fig 11, the TF matrix multiplies the IDF vector. Each sentence is now represented by the special words' and their IDF value instead of just occurrences.
- In Fig 12, the TF-IDF matrix is then reduced to a selected dimension of 2 by SVD. In the final 2x2 matrix, each document and it's TF-IDF information, is now represented by the "likeness" to each "concept". Sentence two has a negative "likeness" to the second component.

Categorical Data Transformation:

- Dummy Variable (binary encoding)

Fig 13. Shows the first 5 columns Gene and Variation column turned into dummy variables, which now has 3321 rows (documents, rows) and 3260 features (categories, columns).

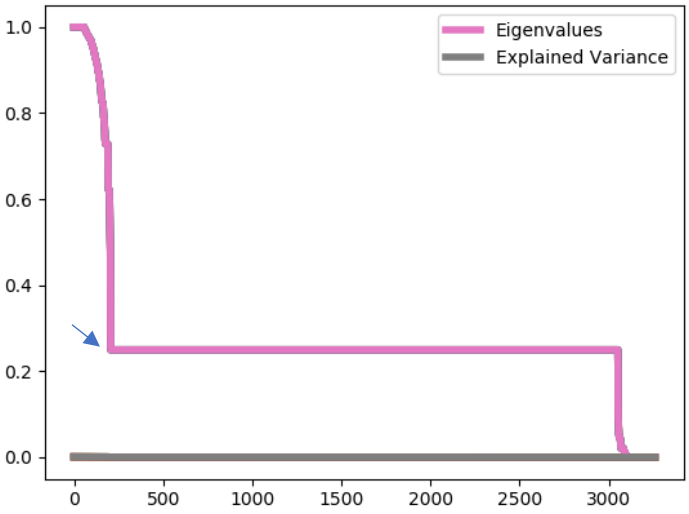
	Gene_ABL1	Gene_ACVR1	Gene_AGO2	Gene_AKT1	Gene_AKT2	Gene_AKT3	Gene_ALK	Gene_APC	Gene_AR	Gene_ARAF	...	Variation_Y
0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	0	0	0	0	0	0	0	...	0

5 rows × 3260 columns

- MCA (Multiple Correspondence Analysis)
  - MCA is also a reduction technique that uses SVD, but is for categorical variables. But instead of depending on the covariance matrix like the continuous variables techniques, this depends on the chi-square distance matrix, which calculates the frequency of features occurring together. [10]
  - Inertia is the measure for variation when used in correspondence analysis. The inertia is high when the expected frequency is far from the average frequency.

- Benzecri and Greenacre are mathematicians who both believed that this estimation is too optimistic, and gave their own corrections.
- Each feature will have its own Eigenvalues are assigned to then are listed from biggest to smallest, depending on the largest distance by count (Inertia) between categories.

Fig 14. Shows the 3260 eigenvalues representing each category, along with their explained variance. The eigenvalues taper off at the 206th mark.



	Eigenvalues	Expl_Var
200	0.594075	0.000661
201	0.545530	0.000607
202	0.528223	0.000588
203	0.503153	0.000560
204	0.476915	0.000530
205	0.296265	0.000330
206	0.249847	0.000278
207	0.249847	0.000278
208	0.249847	0.000278
209	0.249847	0.000278
210	0.249847	0.000278

- Most of the variance can be explained with around 200 categories. Therefore, 207 will be chosen as the number of dimensionality reduction.
- Concatenate MCA factors with SVD components

Fig 15. Shows the first 5 entries with their MCA factor scores and SVD components. Both categorical and text data are now represented by concepts that shows their likeness, either positive or negative and in all ranges.

mca_factor_7	mca_factor_8	mca_factor_9	...	svd_component_90	svd_component_91	svd_component_92
-8.916000e-14	-2.046930e-14	1.928368e-14	...	-0.007426	0.006297	-0.006874
-6.389868e-14	-2.410942e-14	-1.072718e-14	...	0.025561	0.005552	0.004093
-6.194446e-14	-1.834018e-14	-1.055463e-15	...	0.025561	0.005552	0.004093
-6.705280e-14	-2.229254e-14	1.584202e-15	...	0.016365	-0.003176	0.003422
-6.222154e-14	-2.663374e-14	-3.603054e-15	...	-0.030305	-0.000440	-0.002712

Implementation

- Split the preprocessed features and target into 90% for training, and 10% for validation,

- XGBoost [11]:
  - Ensemble boosting: Adding a new model that learns from the errors of the previous model until no improvements can be made, then combine the models for an ensemble prediction.
  - XGBoost is an extension to GBM (Gradient Boosting), which is an extension of the original boosting methods, such as AdaBoost (adaptive boosting).
    - AdaBoost combines weak learner's weighted predictions together, adding one at a time, depending on the accuracy.
    - Gradient Boosting adds gradient descent to a set of parameters for tuning at each new decision tree, with the objective to minimize a loss function, which for this case is log loss.
      - Gradient Descent[12] is the technique used for minimizing log loss in a gradual way until local minima (optimal parameters) are found. Formula:

$$m_{n+1} = m_n - \alpha \frac{\partial}{\partial m_n} LF(m_n)$$

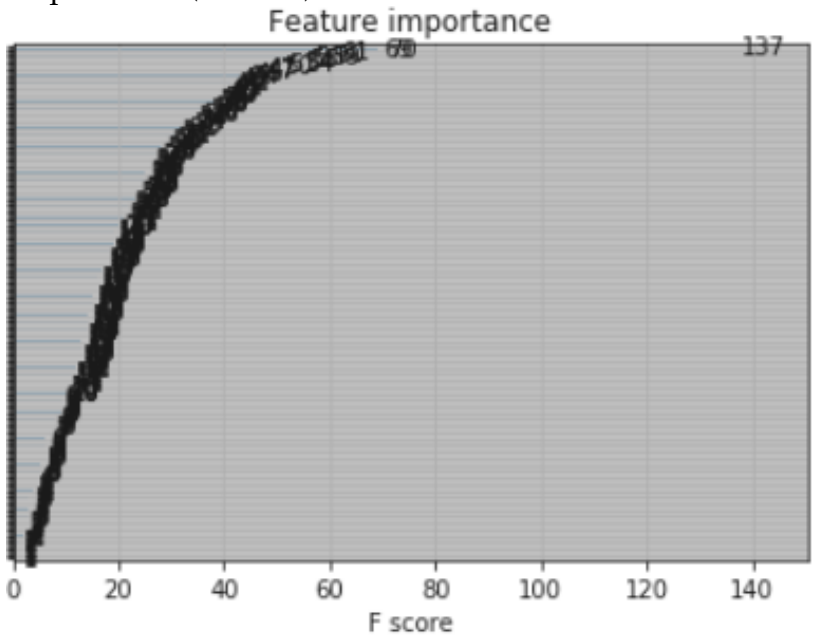
      - Th new tree added (Mn+1) equals the previous tree (Mn) minus learning rate (α) multiplied by the partial derivative ($\frac{\partial}{\partial m_n}$) or "change" in the loss function (LF) of the previous tree (Mn).
      - The sum of the predictions of each tree must follow gradient descent to reach local minima, but it is up to the parameters such as number of trees, depth of trees, and learning rate to decide the optimal complexity.

  - XGBoost is an extension to GBM which has features that allows for a much faster performance by optimizing usage of computer hardware. It also has helpful features such as:
    - Cross-validation to find local minima
    - Objectives from binary, multiclass, regression, to ranking
    - Visualization of features
- Initial results with default XGBoost parameters.

```
In [46]: clf = XGBClassifier()
         clf.fit(X_train, y_train)
         pred = clf.predict_proba(X_test)
         log_loss(y_test, pred)

Out[46]: 0.89310749137337819
```

Fig 16. Shows the importance (F-score) of all features.



Refinement

Additional features:

- Text:
  - Add total length of each document (did not have a positive impact)
  - Add total number of words for each document (did not have a positive impact)
- Categories
  - Add category whether variation contained the words "Fusion", "Overexpression" and others.
  - Add cosine scores of each entry to the factors

Fig 17. Shows new features added, and resulting log loss improvement of 0.8931 to 0.8928 with default XGBoost parameters. The kernel [3] from Fig 4. Also provided sample code and insight on finding additional features of the following:

	Is_amplification	Is_deletion	Is_fusion	Is_insertion	Is_overexpression	Is_silencing	None stated	mca_cosines_0	mca_cosines_1	mca_cosines_2
0	0	0	0	0	0	0	1	0.000041	1.066242e-29	9.269235e-29
1	0	0	0	0	0	0	1	0.000038	5.812447e-30	4.759638e-29
2	0	0	0	0	0	0	1	0.000038	6.047211e-30	5.077815e-29
3	0	0	0	0	0	0	1	0.000038	6.204391e-30	4.438468e-29
4	0	0	0	0	0	0	1	0.000038	4.481824e-30	4.943337e-29

5 rows × 57 columns

```
log_loss(y_test, pred)
```

```
0.89287172057092545
```

Parameter tuning:

- TFIDF:
  - Max_df: Maximum percentage of documents a word must appear in to be selected, 0.9 means a word must appear in no less than 90% of the documents
  - Min_df: Minimum number of documents a word must appear in to be selected, 2 means word must appear in at least 2 documents
  - Ngram_range: If choose to use n_grams, this will decide 2 or 3-word n-grams as (1,2) or (1,3)
- SVD:
  - N_components: Number of components to reduce text data
- MCA:
  - N_factors: Number factors to reduce or increase categorical data
  - Greenacre: Enable/disable corrections for MCA factor scores
  - Benzecri: Enable/disable corrections for MCA factor scores
- XGBoost:
  - N_estimators: The size of the tree to fit to data
  - Learning_rate: Shrinks the update of the new weights, the lower the number the more the more conservative to prevent overfitting
  - Max_depth: The maximum depth of a tree, which controls complexity and overfitting
  - Min_child_weight: The minimum sum of needed weight in a child to further partition, controls overfitting
  - Gamma: Minimum loss requirement to make a split
  - Subsample: Fraction of observations to be random samples

- **Fig 18.** Shows that the mean validation log loss was at its lowest at the 55th iteration, using XGBoost.cv's early_stopping_rounds.

```
[54]    train-mlogloss:0.190477+0.000691101     test-mlogloss:0.935905+0.0130819
[55]    train-mlogloss:0.185419+0.000804643     test-mlogloss:0.935904+0.013413
[56]    train-mlogloss:0.180212+0.00109947      test-mlogloss:0.936382+0.0137583
```

  - The default learning rate is 0.1, min_child_weight is 1, and max_depth is 6
  - The 55th tree provides is the optimal number of trees for this size of features, with default parameters.
  - Set N_estimators at 55 with default everything else
  - Grid Search parameters in the following order: [8]
    - {'max_depth': 5, 'min_child_weight': 8}
      -0.8958056642788611)
    - {'gamma': 0.0},
      -0.8954813796257004)
    - {'colsample_bytree': 0.6, 'subsample': 0.9}
      -0.8937879198606699)

- {'reg_alpha': 0.0001}
  -0.8907529784963681)
- {'learning_rate': 0.13}
  -0.8907529784963681)

***XG Boost Parameter tuning led to log loss improvements of 0.8928 to 0.8907

Model Evaluation

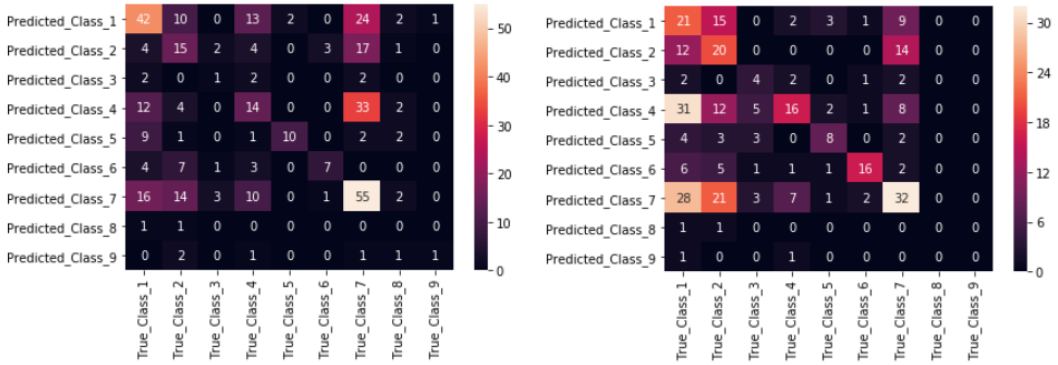
The model will be evaluated with Kaggle provided stage 1 solutions as unseen data. However, only 368 were provided out of 5668 cases. It had guessed with a 0.07% accuracy and 1.651 log loss.

```
print(result)
```

```
1.65148223058
```

For testing the model against noise, the text length and word count added originally intended as positive features, could be noise. There is not much sense the length of a document in helping an oncologist understand this genetic mutation better, as the model did not respond well at all.

Fig 18. Shows Confusion matrix of this model on the validation (top), and 368 unseen samples (bottom). The right shows a lot of more bright colors in cells that are incorrect, as there is a similar pattern in accuracy, but much high misclassification in the unseen data.



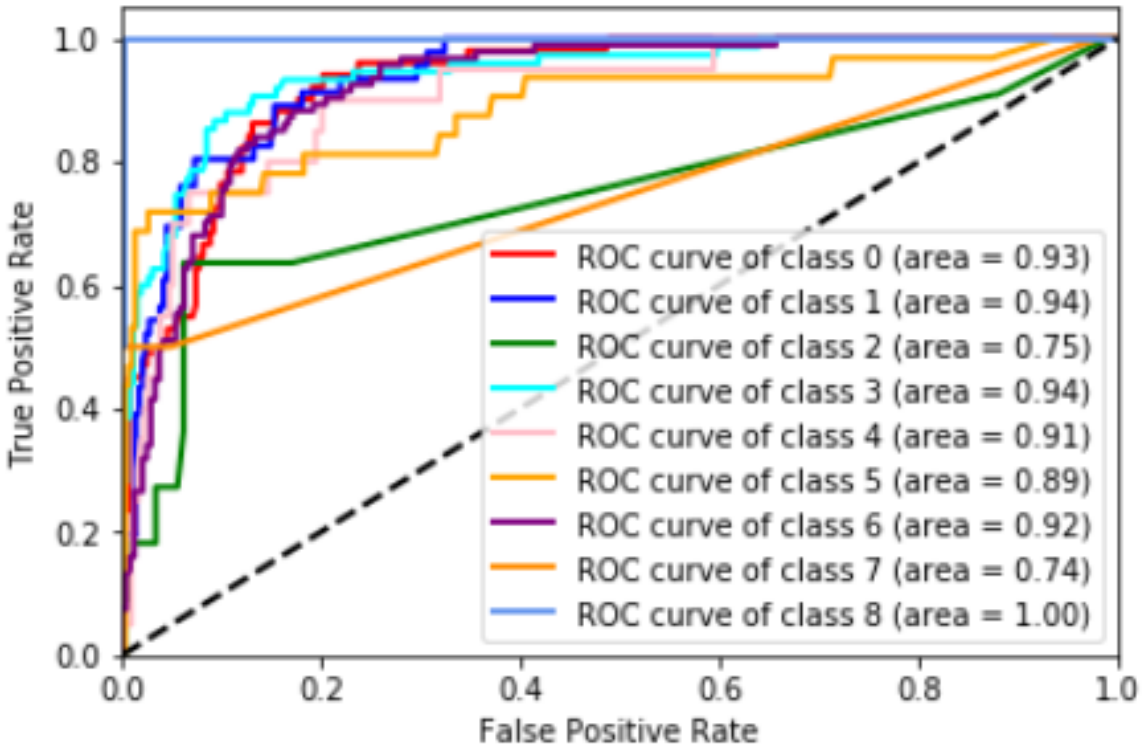
**XGBoost python package error, specifying "num_class" parameter does not work either way when XGBClassifier is used with another sci-kit learn package, in this case OneVsRestClassifier, so I will substitute the model with GradientBoostingClassifier with same parameters

```
TypeError: __init__() got an unexpected keyword argument 'num_class'
```

```
XGBoostError: b'value 0for Parameter num_class should be greater equal to 1'
```

Fig

Fig 19. Uses sample code[9] to show the ROC curve of each class against the rest with the validation data. The ideal area for each curve to be placed is the top left corner, where the TP rate is 1.0 (100% accuracy) and FP rate is 0.0 (0% misclassification). The AUC (Area under curve) is a good measure of how well a classifier is doing. The larger the AUC curve means the closer the ROC is to the ideal top left area, to maximize TP while minimizing FP. Class 2 and Class 7 are the most inaccurate predicted class, while class 0, 1, and 3 are the most accurate.



Justification

The benchmark set for this model was against two of the top kernels for this competition. I have surpassed the goal for this project, but the model could always be improved. I believe more intensive text analysis needs to be done for another significant jump in performance. But for this project, I am happy with the results. I would say this is a not a solution for gene mutation diagnosis, but a step towards advancement in using computer to aid precision medicine.

Conclusion

To further decrease log loss, more intricate text analysis must be performed. Currently, this model has the text transformation set at per word, instead of bi-grams or sentences. To closer emulate a cancer professional's way of understanding the clinical literature, sentences and its sentiments must become features. Scikit-learn's TfidfVectorizer allows for capturing of word phrases in any range, but can be become costly in time and computing. Additionally, the stop words taken out are essential in understanding the literature, as they provide meaning and direction to the keywords. Here is a small example top n_grams found in the 5-12 words range without taking out stop words, of the first 300 clinical documents. Not all the n_grams give meaningful information, but when combined they can provide much more insight than just single words. The combinations could be endless, but it could be worthwhile if it means helping cure cancer.

```
vec2 = TfidfVectorizer(ngram_range=(5,12), max_features=200
                       vec2.fit_transform(train_text['Text'][:300])
vec2.vocabulary_
```

['the epidermal growth factor receptor',

'progression free survival',

'egfr kinase inhibitors',

'to determine whether',

'non small cell lung cancer',

'inlineview popup table',

'in exon 19',

'in this study we',

'unknown clinical significance',

'egfr exon 20 insertion',

'small cell lung cancer',

Reflection

For my first Kaggle competition and my capstone project, I was given data in which each entry was a gene, variation, mutation class and the clinical literature responsible as ground truth. Then I preprocessed and reduced the categorical and text data separately into features, then combined them back together for an estimator to classifier the multilabel mutation class.

I thought the most difficult parts were just a learning experience. Most time was spent on revising coding, coming up with ideas that not every kernel does, and studying the algorithms. There were technical difficulties such as packages not compatible with windows 10 or had bugs for the python package. The final model did better than I expected, but I would not use it in any real-life situations.

Improvement

To further improve the computer's classification, more analysis must be done to clinical text. There are many more natural language processing techniques such as sentiment analysis, and position-tagging that can create more and better features. I do not think my model is a good benchmark, as the lowest log loss score is 0.14 in the public leaderboards. I will be submitting a kernel with the submission log loss file to see my result in the public leaderboards.

References

1. https://www.kaggle.com/c/msk-redefining-cancer-treatment
2. http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss
3. https://www.kaggle.com/eavdeeva/cancer-all-gene-variation-values-are-unique
4. https://www.kaggle.com/the1owl/redefining-treatment-0-57456
5. https://www.kaggle.com/reiinakano/basic-nlp-bag-of-words-tf-idf-word2vec-lstm
6. http://blog.christianperone.com/2011/10/machine-learning-text-feature-extraction-tf-idf-part-ii
7. https://nlp.stanford.edu/IR-book/pdf/18lsi.pdf
8. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python
9. http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py
10. http://www.academia.edu/926945/Correspondence_Analysis
11. https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/
12. http://ucanalytics.com/blogs/intuitive-machine-learning-gradient-descent-simplified/
13. https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/