Redefining Cancer Treatment

Udacity Machine Learning Nano-degree Capstone

Anthony Chiu

September 2017

1 Introduction

1.1 Project Overview

The genetic testing is crucial us to understanding more about cancer, while the research has been slow due to significant amount of manual work, Therefore the Memorial Sloan Kettering Cancer Center (MSKCC) proposed this competition on Kaggle as one of the item of the NIPS 2017 Competition Track.

The classes of the genetic mutations were defined by the American College of Medical Genetics and Genomics and the Association for Molecular Pathology[1]. The classes defines the different fundamental causes of a cancer and are applicable to different type of cancers.

1.2 Problem Statement

There are many genetic mutations inside a cancer tumor, but only some mutations (drivers) contributing to the growth of the tumor. Currently, those drivers are captured manually. It is a really time-consuming task as the clinical pathologist need to classify every single genetic mutation based on evidence from the text-based clinical literature.

In this project, the text input data is first preprocessed by Natural Language Processing methods (NLP) and re-sampled with Overampling method. Finally, a machine learning model is trained to classify the mutation effect (9 classes) of the genes based on given evidence from the text-based clinical literature. Therefore, It is a natural language processing and classification task.

1.3 Metrics

The performance of the model is evaluated by Multi-Class Log Loss, the lower the best.

$$loss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} ln(p_{i,j})$$

Where N is the number of samples, M is number of classes, p is the prediction, and y is the label. This matrix considers not only the accuracy but also how confident the model is. In multi-class classification, the output of a model is the probability distribution of the input being each of the N classes $p_j \in [0, 1]$. A model predicts a lower probability for a correct class will have higher loss.

Unlike the Accuracy, Multi-Class Loss requires the model being confident about the prediction. For example, a binary prediction : [0.49 0.51] is acceptable for the Accuracy but not the Multi-Class Log Loss. It is important in medical industry, which has low fault tolerance.

[1] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4544753/

2 Analysis

2.1 Data Exploration and Visualization

2.1.1 Summary

The dataset contains 3 features as shown below:

Feature	Data Type	Description
Clinical Evidence	Text	The article used to classify the genetic mutations.
Variation	Text / Enum	The amino acid change for this mutations.
Gene	Text / Enum	The gene where the genetic mutation is located.

Table 1: The description of 3 different features of the dataset.

The dataset is expert-annotated and published by MSKCC. Those features are included because those are the evidence used by the experts. There are 3321 training samples and 5668 testing samples for tuning the model. The label of this dataset is the mutation effect (9 classes) of the genes.

2.1.2 Labels Distribution

The label of this dataset is the mutation effect (9 classes) of the genes. The class distribution is not

Figure 1: The frequency distribution of classes

uniform, There are only a few samples belonging to classes 3, 8, and 9. Therefore, training and testing set must be ensured containing all classes.

2.1.3 Explore Clinical Evidence

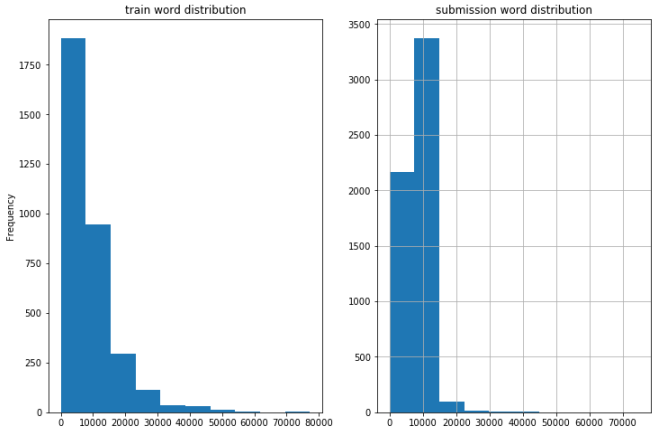

Figure 2: The box plots of word count of the Clinical Evidence

Minimum	1
Maximum	77202
1st quartile	5700
Medium	8040
3rd quartile	10598
Mean	8905.540550
Standard deviation	5597.929867

Table 2: Statistics of the Clinical Evidence

The text count distribution is highly skewed right ($mean > median$). Therefore it is not easy to get the key concepts of a text. Word vectorization/embedding can be useful in this case.

2.1.4 Explore Genes and Variations

Feature	Unique Count	Total Count
Variation	8609	8989
Gene	1507	8989

Table 3: Genes and variations overlapping situation

From the above finding, the one-hot encoding is not applicable for the variation feature, as there is not many overlapping in variation. It is also better to use word vectorization to preprocessing these features. Also, they are usually words not sentences.

3 Algorithms and Techniques

3.1 Term Frequency–Inverse Document Frequency (tf-idf)

The tf-idf is one way to vectorize a document. It converts a list of words to a list of real numbers. With tf-idf, a document is represented by the most frequent words within it (Term Frequency), but the word should not be too common across different documents (Inverse Document Frequency). Those Clinical Evidences should contains many overlapping words across documents, therefore tf-idf should be helpful.

3.1.1 Term Frequency

The Term Frequency $tf(t, d)$ represent how common a term t in a given document d. The most simplest definition would be the term count within a docuemnt.

3.1.2 Inverse Document Frequency

The Inverse Document Frequency $idf(t)$ measures how much information a term t provides. The more common across different documents the term, the less information it carries. Below is one of the implementations of $idf(t)$:

$$idf(t) = log\frac{1 + n_d}{1 + df(d, t)} + 1$$

Where n_d is the total number of documents, $df(d, t)$ is the number of documents containing the term t.

3.1.3 Combine

Combining two values, the tf-idf represents how important a term is within the document and across all documents.

$$tfidf(t, d) = tf(t, d) \times idf(t)$$

3.1.4 Dimension Reduction

The word vector will be truncated to smaller dimension by TruncatedSVD.

3.2 Stratified K-fold Cross Validation

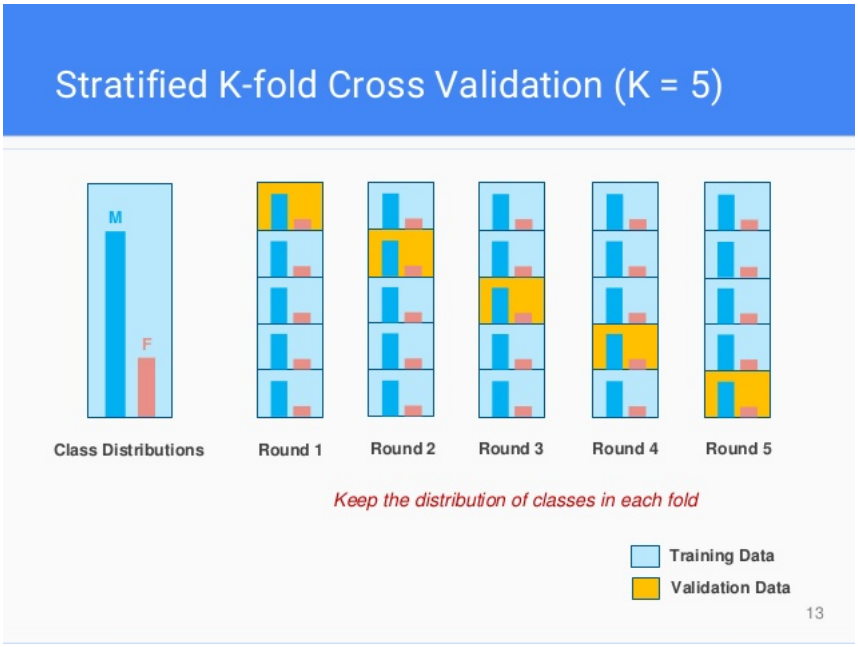

Figure 3: The box plots of word count of the Clinical Evidence

The Stratified K-fold preserves the percentage of samples for each class, which means each fold can represent the whole dataset. It is useful for the data in this project, as the dataset is highly non-uniform. During the cross validation, the whole dataset is separated into the training set and the validation set (the testing set is hosted by Kaggle) and both of them preserves the percentage of samples for each class.

3.3 Gradient Boosting

3.3.1 Adaboost

Boosting is one of the ensemble learning methods. It trains a list of weak learner $h_m$in sequence to produce a ensemble learner H.

$$H(x) = sign(\sum_{m}^{M} c_m * h_m(x))$$

In Adaboost, the output is the weighted average of the results from all n weak learners. Those weak learners are trained in sequence, the mismatched samples from h_{m-1} will be given higher weight for h_m. As a result, each learner will be specialized in learning a part of the problem.

3.3.2 Gradient Boosting

In Kaggle competitions most of the models are either Deep Learning models or Gradient Boosting models. Since the training dataset is small, Deep Learning is not applicable and the Gradient Boosting is chosen here. Gradient Boosting is a generic boosting method. It can work with different loss function. Suppose there are M functions, each function F_m is trained to correct the error of the previous one F_{m-1} by the improvement function $h_m(x)$ with a factor γ_m.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Therefore:

$$F_M = F_{M-1} + \gamma_M h_M$$
$$F = (F_{M-2} + \gamma_{M-1} h_{M-1}) + \gamma_M h_M$$

$$\dots$$
$$F(x) = \sum_{m}^{M} \gamma_m h_m(x)$$

Which is the same as the Adaboost above, but obtained in a different way.

Initialize $F_0(x)$ to a constant C.
for *m in [1,M]* **do**
In each iteration $m-1$, the next base learner h_m and weight γ_m are induced to minimize the loss in this iteration L_{m-1}:
*****
$L_{m-1} = L(y, F_m(x)) = L(y, F_{m-1}(x) + \gamma h(x))$
$h_m, \gamma_m = argmin_{h,\gamma}(L_{m-1})$
$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$
*****
The *argmin* is a optimization problem solved by Gradient descent in Gradient boosting to find out the h_m and γ_m and then F_m can be calculated.
end

Algorithm 1: Gradient Boosting

For example, suppose $F_0(x) = 0, y = 1, L(y, p) = y - p$:

$$h_1, \gamma_1 = argmin_{h,\gamma}(L(y, F_0(x) + \gamma h(x)))$$

$$h_1, \gamma_1 = argmin_{h,\gamma}(L(y, \gamma h(x)))$$

$$h_1, \gamma_1 = argmin_{h,\gamma}(1 - \gamma h(x))$$

Again this is a optimization problem (but no need Gradient descent), the $h_1(x)$ and γ_1 can be 1 and 1 respectively, therefore:

$$F_1(x) = F_0(x) + \gamma_1 h_1(x)$$

$$F_1(x) = 0 + 1 \times 1$$

$$F_1(x) = 1 = y$$

Parameters to be selected (LightGBM):
1. max depth: int, default = -1 – Maximum tree depth for base learners, -1 means no limit.
2. num leaves: int, default = 31 – Maximum tree leaves for base learners.
3. n estimators: int, default = 10 – Number of boosted trees to fit.
4. learning rate: float, default = 0.1 – Boosting learning rate.

3.4 Benchmark

The result of this project is benchmarking the Kernels in the Kaggle leader board. The current best reasonable result (lowest loss) from the Kaggle leader board: 0.42453

4 Methodology

4.1 Data Preprocessing

4.1.1 Text Cleaning

There are 3 steps for cleaning text:

Step	Description
Lower case	All characters are converted to lower case.
Alphabets and digits only	Remove any non-alphabets and non-digits characters
Remove Stop words	Remove all English stop words, such as 'the'.

Table 4: The description of text cleaning steps applied.

4.1.2 Text Vectorization

All documents are converted to same dimension vectors for training by tf-idf. After that, the resultant vectors are truncated by TruncatedSVD to lower dimension.

```
print('Raw:', text_column[0][:60])
print('Clean:', clean_text_column[0][:60])
print('tf-idf vector:', text_tfid[0].shape)
print('Truncated tf-idf vector:', truncated_text_tfid[0].shape)
```

```
Raw: Cyclin-dependent kinases (CDKs) regulate a variety of fundam
Clean: cyclin dependent kinases cdks regulate variety fundamental c
tf-idf vector: (1, 166033)
Truncated tf-idf vector: (50,)
```

Figure 4: The example output of the whole text prepropossing pipeline

4.2 Implementation

4.2.1 Overview

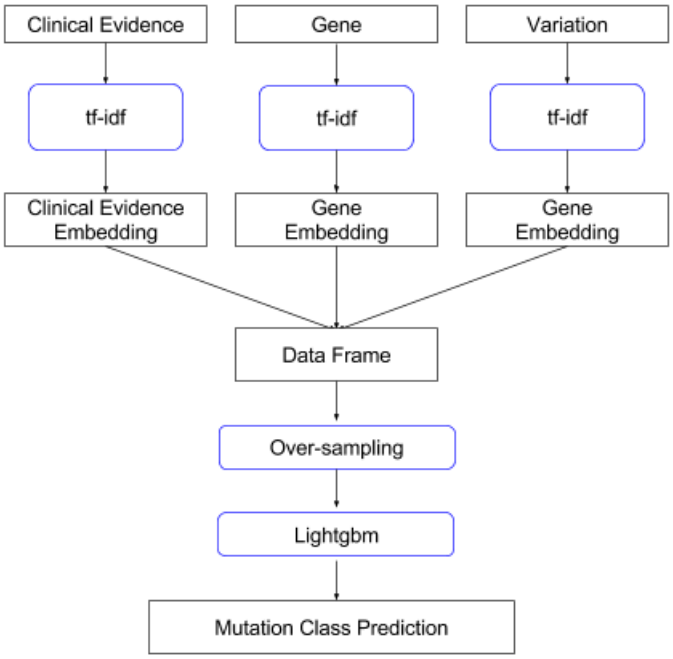

Figure 5: The architecture of the implementation

Step	Package	Description
Text Cleaning	nltk	Discussed in section 4.1
Text Vectorization	sklearn	Discussed in section 4.1
Dimensionality Reduction	sklearn	Discussed in section 4.1
Oversampling	imblearn	Handling imbalanced class distribution
Classifier	lightgbm	LightGBM is used as the classifier
Evaluation	sklearn and seaborn	Plotting graphs for model evaluation

Table 5: The description of the overall architecture.

4.2.2 Oversampling

After preprocessing those three text features, they are grouped to one single pandas dataframe. The dataframe is then be re-sampled by selected Oversampling methods (discussed in section 4.3.1). It is mainly used to handle the issue of imbalanced class distribution.

4.2.3 Classifier

As discussed in section 3.3, Gradient Boosting is used. The Microsoft LightGBM package is selected becuase it is fast and less memory demanding. It supports parallel processing in both CPU and GPU. It is also very easy to be installed in Window system via pip.

4.2.4 Evaluation

Each model is evaluated by average loss of the 5-fold Stratified K-fold Cross Validation. Again it is for preserving the percentage of samples for each class in each fold.

Beside, the following 2 plots are used to evaluate a model:
1) Training Loss and Validation Loss - evaluating over-fitting problem
2) Confusion Matrix - evaluate accuracy of the model in each class

4.3 Refinement

4.3.1 Sampling methods

Dealing with imbalanced dataset, oversampling is a way too improve model performance. In this section the result of using 3 different sampling methods is discussed. All model settings are the same and discussed in the section 4.2. Default values with large number of estimators for the LightGBM is used for a quick check.

4.3.2 LightGBM

After confirming the sampling methods, The the LightGBM is tuned.
The following values is fixed:

1. number of estimators: keep it large and rely on early stopping.
2. learning rate: keep it small and rely on early stopping

The following values is tuned by 5-fold GridSearch:

1. max depth: 4 to 20 with step 4
2. num leaves: 4 to 20 with step 4
3. boosting type: Gradient Boosting Decision Tree, Dropouts meet Multiple Additive Regression Trees, and Gradient-based One-Side Sampling.

5 Results

5.1 Refinement

5.1.1 Oversampling

Sampling method	Average 5-folds validation loss
Without Sampling	0.9191
Random Sampling	0.2071
SMOTE	0.7230
ADASYN	0.7189

Table 6: Oversampling methods comparison results

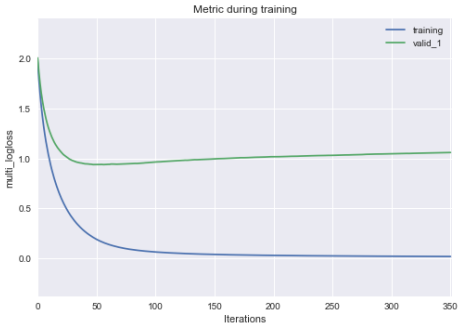

Figure 6: Without sampling

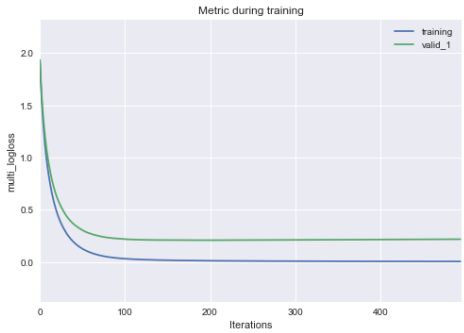

Figure 7: Random Sampling

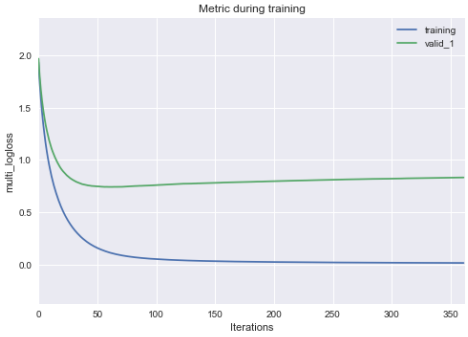

Figure 8: SMOTE Sampling

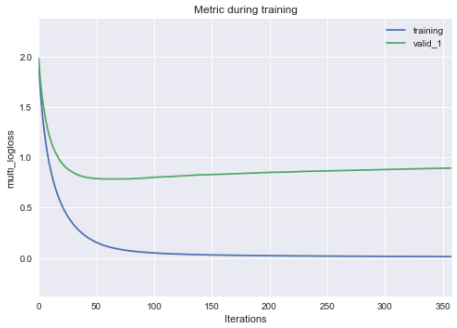

Figure 9: ADASYN Sampling

The result suggests:
1. Oversampling is helpful to improve performance.
2. Random Sampling avoid overfitting.
3. Random Sampling is the most effective way.
Therefore Random Sampling is selected for the next step (model tuning).

5.1.2 Classifier

The best parameters obtained for the LightGBM:

1. Gradient Boosting Decision Tree
2. 12 maximum depth
3. 20 number of leaves

The above parameters achieve average 5-fold validation loss: 0.1967, which is better than the quick check above.

5.2 Model Evaluation and Validation

5.2.1 Final Pipeline

TfidfVectorizer

The three input features are first processed by TfidfVectorizer.

The TfidfVectorizer for Clinical Evidence:

```
TfidfVectorizer(analyzer="word", ngram_range=(1, 2),
    tokenizer=nltk.word)
```

The TfidfVectorizer for Genes and Variations:

```
TfidfVectorizer(analyzer="char", ngram_range=(1, 10))
```

Word-level 1-2 grams tf-idf is used for the Clinical Evidence, while character-level 1-10 grams tf-idf is used for the Genes and Variations. Since both Genes and Variations are usually a word, therefore char-level vectorizer is applicable.

TruncatedSVD

The word vectors produced from TfidfVectorizer is then truncated to lower dimension.

The TruncatedSVD for Clinical Evidence:

```
TruncatedSVD(n_components = 100, n_iter=30)
```

The TruncatedSVD for Genes and Variations:

```
TruncatedSVD(n_components = 50, n_iter=30)
```

More components from the Clinical Evidence feature are kept because they are more important than the Genes or the Variation features.

RandomOverSampler

```
X_random_resampled, y_random_resampled = ros.fit_sample(tf_idf_X, training_labels)
```

LightGBM

```
tf_idf_clf = lgb.LGBMClassifier(
    boosting_type='dart',
    learning_rate= 0.01,
    objective = 'multiclass',
    n_estimators= 10000, # will be stopped by early_stopping_rounds
    num_class = 9,
    max_depth = 8,
    num_leaves = 17
)
tf_idf_clf.fit(X_train, y_train,
                    verbose = False,
                    early_stopping_rounds = 300,
                    eval_set= [(X_train, y_train), (X_test, y_test)])
```

Setting a high number of estimators and low learning rate to train the model slowly and early stopping rounds is used for terminating training as the validation loss keep unchanged. Using the Dropouts meet Multiple Additive Regression Trees (Dart) instead of the Gradient boosting decision trees (GBDT) as the boosting method because the Dart produces higher accuracy.

Fold	Validation Loss
1	0.1978
2	0.1919
3	0.1977
4	0.1857
5	0.2105

Table 7: 5-fold cross validation result

5.2.2 Model Performance

The average validation loss is 0.196, and the standard deviation is 0.008206, which indicates the model is stable.

5.3 Justification

After each fold a Kaggle submission is generated to maximize the performance in the public leader board.

Fold	Kaggle Loss
1	0.67952
2	0.65902
3	0.72539
4	0.55727
5	0.67410

Table 8: 5-fold Kaggle result

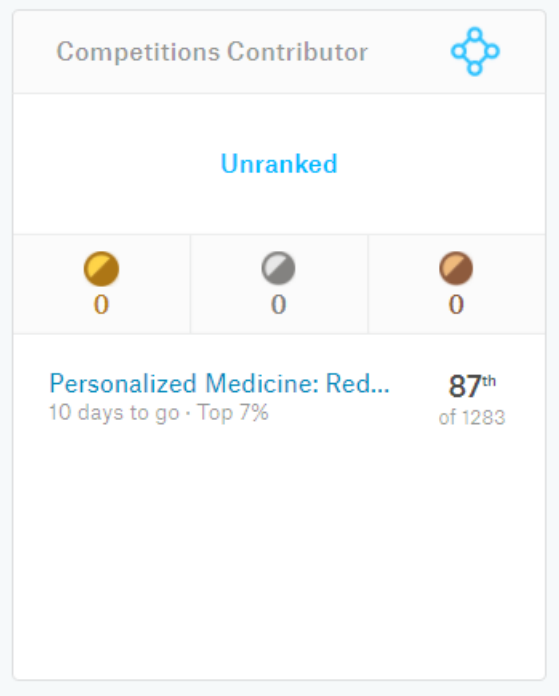

Figure 10: The Kaggle result

The best Kaggle loss of my model is 0.55727, which is worse than other top players in this competition. But it is better than majority of the entries, I think it is encouraging.

6 Conclusion

6.1 Free-Form Visualization

The following figure show the confusion matrix and the accuracy plot of the final model.

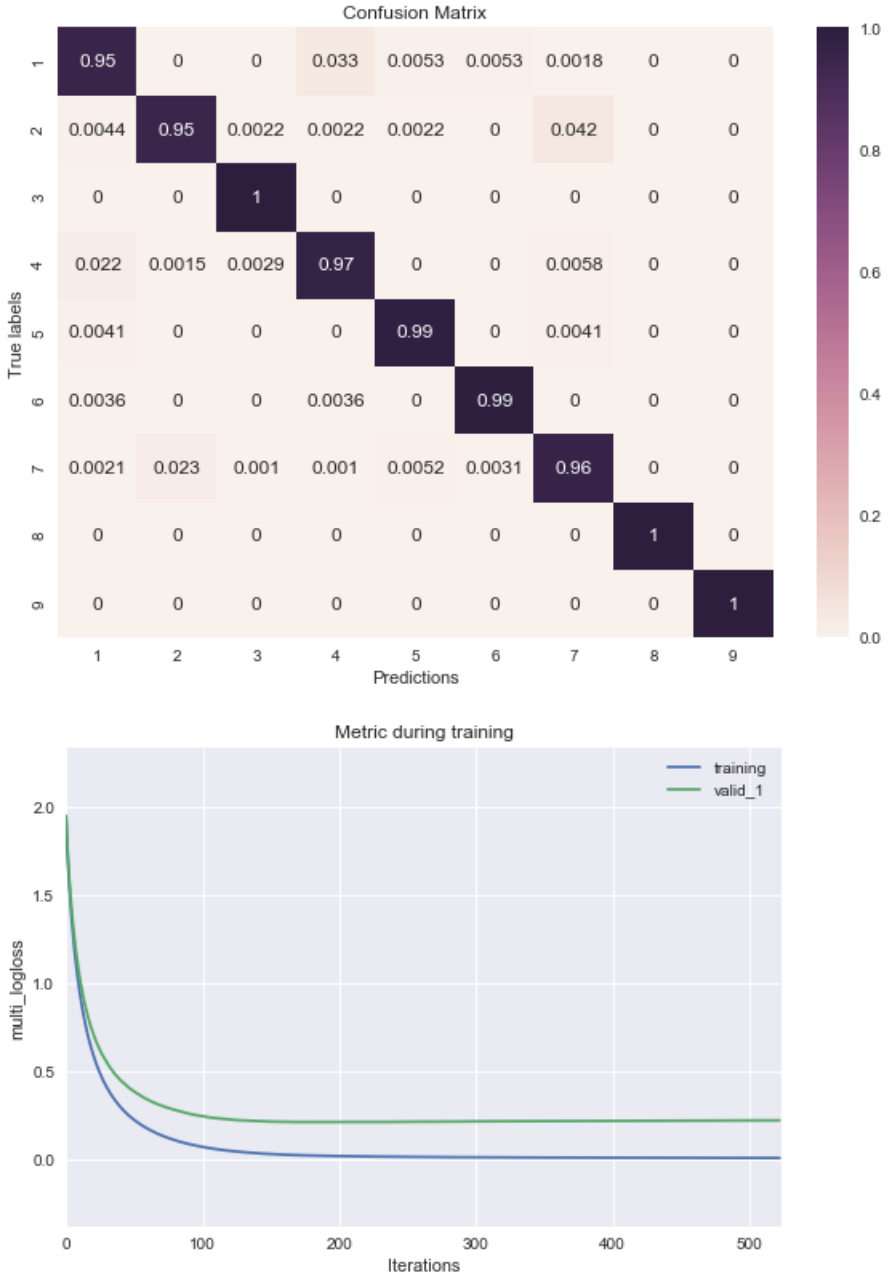

Figure 11: The confusion matrix of the model

Recall the class distribution, class 8, class 9, and class 3 have the least distribution. While the confusion matrix shows the model's predictions completely correct on those classes. Also, the overall accuracy per class is larger than 0.95, which is acceptable. The accuracy plot suggests the model is not overfitted.

6.2 Reflection

6.2.1 Summary

Text Vectorization

In-order to apply machine learning methods, the input must be real number vectors, At the beginning of this project, I have explored different Natural Language Processing methods.
1. I have explored different Natural Language Processing methods to encode the text features, such as Bag of words, n-grads, tf-idf and Doc2Vec.
2. I plotted the resultant word vectors and see if any of these methods can produce clear separation of classes.
3. I used a simple Decision Tree model to see which word vectors produce the best performance.

Oversampling

The data frame is then re-sampled randomly with replacement to make a even class distribution.

Classification Model

The vectors is then being used for training the classification model.

Is the model good enough?

The final model in this project can be a replacement for the existing manual work:
1. The final model prediction accuracy is higher than 0.95 for each class.
2. It takes less than a minute to predict over 5000 samples, which must be faster than human.

6.2.2 Interesting part: Imbalanced Class Distribution

As shown in above, the class distribution of this dataset is totally imbalanced. Looking at the initail result of the K-fold cross validation, there are great variation from fold to fold. I think the real world data should be very similar. Thanks to my first reviewer pointing out the oversampling methods, I am able to obtain a better model.

6.2.3 Difficult part: Small Training Set

The training set size is small (only 3321 samples), while the submission set size is 5668.

6.3 Improvement

Other than tfidf, a pre-trained word embedding model, such as Word2Vec should produce better result. There is a pre-trained Word2vec model by Google trained on roughly 100 billion words from a Google News dataset.