CS249 Current Topics in Data Structures 2017 Final Report

Professor Wei Wang (weiwang@cs.ucla.edu)

Teaching Assistant Guangyu Zhou (guangyuzhou@cs.ucla.edu)

Project: Personalized Medicine for Redefining Cancer Treatment

Members: Yaxuan Zhu (704947072), Qi Qu (404888347)

Don Lee (505032750), Yiming Wang (804775584), Jiyuan Shen (504882830)

Abstract

Personalized medical treatment in cancer requires huge manual effort even with advanced genetic analysis technology. In this Kaggle contest, Personalized Medicine: Redefining Cancer Treatment, we aid to change this low-efficiency, time-consuming tedious work using machine learning models especially in text classification. In this report, we first presented our observation on the official dataset and then proposed a system consisting of feature selection and pattern classification modules. Based on word2vec and Xgboost, our model achieved promising classification results, ranking top 10% among all submission in this contest.

1. Introduction

Precision medicine has become popular in recent years and genetic testing is going to change to way cancer is treated. More specific, a cancer tumor may have thousands of genetic mutations, and the job of distinguishing the mutations that causing the tumor growth (drivers) from the neutral mutations(passengers) is challenging.

1.1 Problem statement

The interpretation of genetic mutations requires a huge amount of manual work. A clinical pathologist need to spend a lot of time manually review and classify every single genetic mutation based on text-based clinical literature. In order to make the process more efficient and effective, we need to develop an algorithm that automatically classifies genetic variations with some basic knowledge about this field. More specifically, we apply test mining and machine learning algorithms on training our model. For feature extraction, we apply both the bag of word model and TF-IDF model, and for classification, we apply neural networks, SVM, and xgboost. The experimental data mainly comes from the kaggle competition. It consists of the information about genes, variations, and the class label. we evaluate the result by calculating Multi Class Log Loss between the prediction probability and the ground truth labels.

1.2 Main Contribution

In this project, we compared effectiveness of the several word embedding methods (TF-IDF, BOW, Doc2vec with different settings) and classification models (KNN, SVM, Neural Network, Xgboost, etc.). We propose a model that ranks top 10% in the Kaggle competition (Note that our results are based on public score. We discuss why we think the public score is more convincing now in section 4.2). Our final model uses doc2vec to vectorize the clinical evidence in English and uses Xgboost as a classifier to achieve high accuracy multi-class classification. We also presented our observation in official dataset and visualize the statistical results. In the end, we try to propose a new way to reformulate this problem and report some of our trials in this direction.

This summary is organized as follows:
Section 2 talks about the related work of our project; Section 3 shows our statistical results of the dataset; Section 4 talks about our evaluation method; Section 5 talks about our feature selection method, including word embedding method we use and how to deal with the other two features; Section 6 talks about our classification model setting; Section 7 shows our experiment results and our discussion; In Section 8, we tried to propose a new way to formulate this task and we show our result and talks about our future work; Section 9 in our conclusion. Our source code can be find at github: https://github.com/AlvinZhuyx/Personalized-Medicine-Redefining-Cancer-Treatment and our pre-trained document vector can be find in the google drive https://drive.google.com/drive/folders/1703i996nsfiDldvK8_aTT1G2nX4i1Qnu?usp=sharing

2. Related Work

2.1 Basic Wording Embedding Method and Classification Method

2.1.1 Feature Extraction

(1) The Bag of Word model represents the documents by the frequencies of each word it contains. However, if a term appears very often across the corpus, it means it does not carry special information about a particular document. One drawback is that it only care about the frequencies of each word instead of their semantic similarities and do not care about the words' order. Two sentences might be semantically similar but do not use the same words. **(2) The TF-IDF model** followed the idea that a word that frequently occurs in one document but not so frequently in the corpus maybe important to represent this document. Therefore, it considers the Inverse Document Frequency (IDF) besides Term Frequency (TF). Here same as the above bag of word model, one drawback is that it focuses on the frequencies of each word while not considering semantic similarities and orders. To fix the above drawback, Mikolov er al. [6,7] proposed a novel word embedding method **(3) Word2Vec**. Word2Vec learns the vector representation for each word use a shallow neural network language model. In the model each word vector is trained to maximize the log probability of its neighboring words. Therefore, the embedded word vectors contain the context information and this enables us to measure the similarities between words. But when we think about measure similarities between documents, we need to refer to 4. Le and Mikolov er al [8] proposes a method called **(4) Doc2Vec**. It's quite similar to word2vec but trained a paragraph vector together with the word vectors to predict the words in the documents. This paragraph vector can then be used in classification. Another method proposed by Kusner er al. [9] focus calculating the distance between two documents, which is called Word Mover Distance. It bases on the word vectors and follows the idea of training a model to get the minimum cost for "travel" from one documents to another.

2.1.2 Classification Method

Here (1) Neural Networks require large amount of training data and they can result in overfitting. The WDM works well in clustering, but it also suffers from its complexity and sometimes it not works well with classification problem. Several other attempts have been made to fix this. Besides, due to its feedback structure, RNN can naturally make use of the sequence information is also widely used in NLP. However, it might be hard in training due to the vanishing gradients in time. LSTM fixes this problem by introducing the "forget" gate to the extent to which a value remains in memory. Considering (2) SVM and Xgboost, which is employed better for small-scale dataset without overfitting. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. Since for SVM, only support vectors but not all the samples are used to construct the separating hyperplane, that is the classifier. However, to be more scientifically, we need to further perform generalization tests (also for SVM not only with Gaussian but also with linear kernel).

2.2 Related Work in Biomedical

How to Train Good Word Embeddings for Biomedical NLP[1]

Although those feature extraction methods have been well studied in literature, how to properly apply them in biomedical domain still need more experiments. Chiu et al. make series of experiments using the word2vec method on life science literature. They adjust different hyper-parameters including the sub-sampling, minimum-count, context window size, etc. And they find that sentence-shuffled text corpora with proper size and parameter setting shows better result. And skip-grams generally shows better result than CBOW.

Personalized Medicine with RNN[10]

This kernel uses a recurrent neural network that uses GRU cells with a bidirectional layer and an attention context layer. The model uses the beginning of the text and the end of the text and then join both outputs along with a one hot encoded layer for the gene and another for the variation. The variation has been encoded using the first and the last letter. This general model is clarified as hierarchical attention network, explained in Hierarchical Attention Networks for Document Classification[5] The innovative points of this approach lie in that firstly it has hierarchical structure that mirrors the hierarchical structure of documents; secondly it has two levels of attention mechanisms applied at the word and sentence level, enabling it to attend differentially to more and less important content when constructing the document representation. This exploration is efficient in processing long-distance relations with memory. The biggest profit of adding attention is that we can then naturally explain the importance of every sentence and word for classification.

Cancer Hallmark Text Classification Using Convolutional Neural Networks[4]

In their published paper, Cancer Hallmark Text Classification Using Convolutional Neural Networks, Baker et al. proposed using a convolutional neural network method to tackle a similar problem, as opposed to the more generally used support vector machine method. The original problem in the paper is to classify text of abstracts of biomedical publication according to ten cancer hallmarks. Hallmarks of cancer are different biological properties that can lead to cancer. Each abstract can be classified to more than one hallmark, or none of the hallmark. So the original problem is a multi-label problem, whereas the problem of this project is a multi-class problem, ie. every input text is classified to be one type of result. Baker et al. tested their CNN model and obtained competitive results against SVM models. The tuned CNN model out-performed both SVM models presented in the paper. The pros of this approach are that it requires less manual engineering. General SVMs require a good amount of knowledge and engineering such as parsing to create useful features (mentioned above) from the texts to obtain good learning results. Some of the drawbacks of using CNN in this type of NLP problems include the large amount of computation and large number of optimization choices for parameters.

3. Data overview

The experimental data, both training and testing, comes from the online kaggle competition. Both train set and test set are stored separately in two files, one for genes, variations, and the class label and the other for the expert-annotated clinical literatures. In this section, we dive into the datasets to make clear of their structures, calculate some basic statistics, and draw some knowledge from the statistics. Our procedures in performing data overview mainly follow and deviate a little from that of https://www.kaggle.com/headsortails/personalised-medicine-eda-with-tidy-r/notebook.

3.1 dataset structure

The data comes in separate files even for the same set. There is one file for the data ID, the gene names, the variation type, and the cancer type (class label). In the other file table and in the corresponding columns are the data ID that matches the first file and the plain text of clinical annotation. The variants files are easy to import with read.csv() fuction. The text files are a little bit more complicated because in the data files the first line which indicates the name of each column is separated into columns as it should be, where as the following lines are not, ie. the lines after the first line are all in one column. For this reason, we take use of the tibble library to first read in each line as a long character object and then manually separate the data and name the features. After data importation, the next step to join the tables of the same set by the dataID. After joining the tables together will result in the final structure of our datasets:

Our datasets contain three features and one label. The following analysis in this section focuses on gene, variation, and label, leaving the text component in the later section 5 where we describe how we handle the text component and extract information from it using NLP-approaches.

3.2 basic statistics

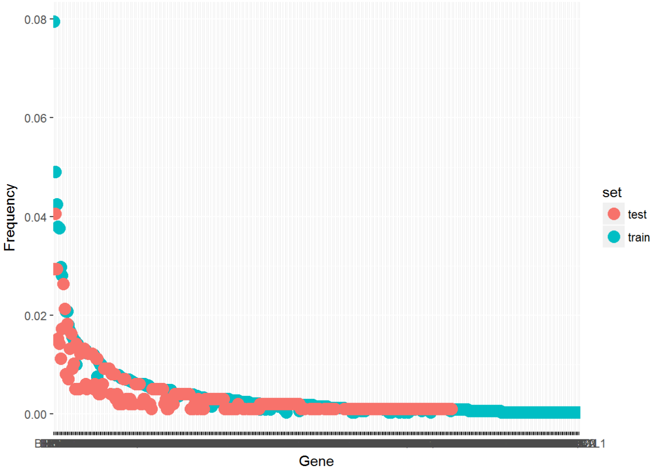
An initial glimpse at the train set shows that there are 3321 data entries, 264 different gene occurrence, 2996 types of variations, and classified into one of nine classes. Some genes are much are frequent than the others, and simple grouping and counting can prove this. The most occurring gene in the train set is "BRCA1" with a count of 264 occurrences, whereas many genes only appear once in the dataset. Further looking reveals that the top ten frequent genes take up roughly one third of the train set. Similarly for variation, a few of the frequent variations are much more frequent than the rest. This phenomenon is also seen in the class label column. Of all 3321 data entries, only 19 of them are classified as type 8.

```
##         ID              Gene                      Variation       Class  
##  Min.   :   0   BRCA1   : 264   Truncating Mutations:  93   1:568  
##  1st Qu.: 830   TP53    : 163   Deletion            :  74   2:452  
##  Median :1660   EGFR    : 141   Amplification       :  71   3: 89  
##  Mean   :1660   PTEN    : 126   Fusions             :  34   4:686  
##  3rd Qu.:2490   BRCA2   : 125   Overexpression      :   6   5:242  
##  Max.   :3320   KIT     :  99   G12V                :   4   6:275  
##                 BRAF    :  93   E17K                :   3   7:953  
##                 ALK     :  69   Q61H                :   3   8: 19  
##                 ERBB2   :  69   Q61L                :   3   9: 37  
##                 (Other):2172   (Other)             :3030          
```

The test set shows some different statistics. There are 986 total data entries, with 276 and 945 different types of genes and variations. The difference in gene occurrence between frequent and less frequent genes are not as abrupt, while the variations remains an extreme long trail distribution.

```
##        ID               Gene                      Variation  
##  Min.   :  1.0   TP53    : 40   Truncating Mutations: 18  
##  1st Qu.:247.2   BRCA1   : 29   Deletion            : 14  
##  Median :493.5   EGFR    : 29   Amplification       :  7  
##  Mean   :493.5   SCN4A   : 26   Fusions             :  3  
##  3rd Qu.:739.8   TSHR    : 21   G13R                :  2  
##  Max.   :986.0   ERBB2   : 18   G13S                :  2  
##                  BRAF    : 17   G44D                :  2  
##                  TP63    : 16   A114V               :  1  
##                  PTEN    : 15   A1156T              :  1  
##                  (Other):775   (Other)             :936  
```

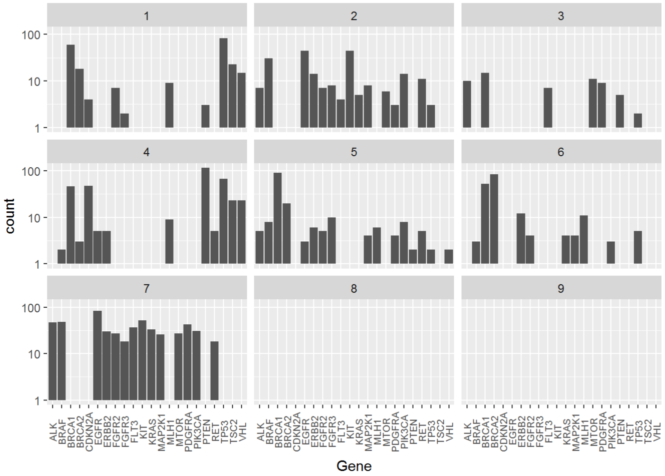
This imbalanced distribution is very visible when we plot the genes' frequency in descending order. We count and sort each gene's occurrence in both data sets and plot them in the same graph shown below. It is observable the two datasets share a very long-tailed distribution in common. In addition, genes that frequent in the train set are likely to be frequent in the test set.



In supplement to the train and test datasets, we have obtained from Kaggle another set of data, which was initially posted as a stage 1 test set but some class labels were provided after the competitive has moved on to stage 2. Of all above 5000 data entries of this set, only 328 of them are labeled. We filtered out these 328 data entries, called them the 2nd train set and appended them to our original train set in some testing. This set of data also shows a imbalanced and long tailed distribution of genes, variations, and class labels.

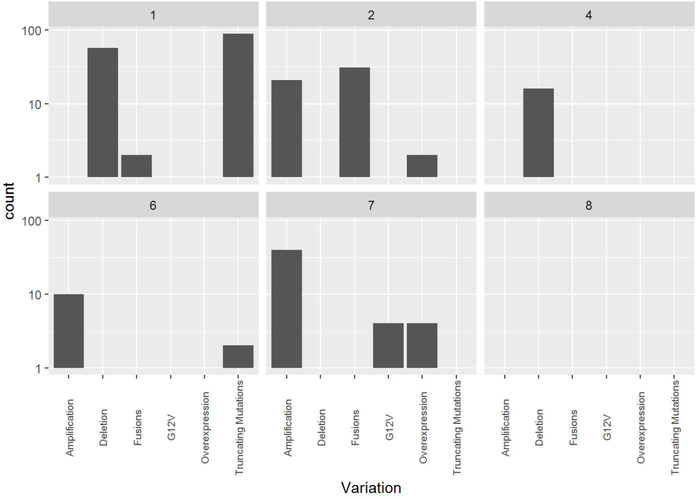
3.3 data insights

Beyond looking at the datasets in each column, we try to find some correlation between individual feature and class label. Because the gene types are so unevenly distributed across the datasets, we want to see if this feature provides immediate knowledge in classifying the class. In other words, we want to find some correlation between gene and class in our train set. Because of the long tailed distribution, we filtered our dataset to only look at the top 10% frequent genes. We plot their frequency in each class and obtain the following result.



The first observation is the empty plots for class 8 and 9, which indicate that none of the entries that are classified to either class 8 or 9 have any frequent gene in their gene column. Remember that only 19 and 37 entries are classified to class 8 and 9 respectively in the train set. Another observation is that the gene “PTEN” occurs mostly in class 4, and the gene “BRCA1” occurs mostly in class 1, 4, 5, and 6. We can see that gene type alone does give us some information to determine its class.

We perform a same analysis with frequent variations and obtain the following result.



This result shows us more knowledge about certain variations, but less knowledge about other classes. For example, we can see that variation “Fusions” is solely related to class 2, variation “Amplification” only occurs in class 6, and variation “Truncating Mutations” occurs mostly in class 1. These results mean that when our learning model comes across these variations, they can make easy predictions with high accuracy. However, our data statistics show that there are so many gene variations and so few of them are frequent that coming across the same variation is very unlikely. Furthermore, the missing plots for class 3, 5, and 9 and empty plot for class 8 indicate that these classes have no frequent variation.

After looking at gene and variation and their correlations to the class label individually, we attempt look at them as a combination, for a certain gene can only undergo one of some set of mutations. We want to investigate if there are combinations of gene and variation that are present in both training and testing set. Unfortunately, result shows that there is surprisingly *zero* combination matches between the train data and test data. In other words, none of the gene and variation combination that occurs in one data set can be found in the other. t

4. Evaluation

From the previous section, we have a sense of the structure of the dataset and we know some of its properties through the statistics and visualization. In this section we will discuss the evaluation metrics of the project, including the official rubric and other metrics we use during the developing phase.

4.1 Official metrics on Kaggle contest

All submissions to original Kaggle contest on “Personalized Medicine: Redefining Cancer Treatment” are evaluated on Multi Class Log Loss between the prediction probability and the ground truth labels. More concretely, it is the cross entropy between the prediction vector and the ground truth labels.

L(p, y) = -1Nj=1Ni=1Cyi, jlog(pi, j)

Through its mathematical form, it is obvious to see that the lower the log score, the better the prediction. Besides, unlike metrics like accuracy that consider single predicted label, log score take the probability vector as input. As a result, it is possible for log loss to distinguish the performance of two prediction with same accuracy, in which one is almost certain about the prediction (with the probability of predicted class close to 1 and that of other class close to 0), while the other is ambiguous about the prediction (with the prediction probability among the same level).

4.2 Submissions and Leaderboard

On Kaggle contest page, each team is allowed to upload their submission 5 times a day to see the result of the prediction. By the time we started it, the contest is already closed. Yet teams are still allowed to submit to get results.

Each submission will be judged and get two scores back, namely “Public Score” and “Private Score”. According to the specification, the two scores are computed on different subset of the test set. “Public Score” is calculated with approximately 76% of the test data, while the “Private Score” is based on the rest 24%. It is also said that, during the contest, only public score is available to competitors while the final results are based on the private score, which is unavailable to the public by then.

In our view, there are two potential drawbacks of this submission setup. First, the dataset especially the test set is small. The private score is not calculated based on sufficient records (the data size is only 200+). What makes it worse is the imbalanced fact of the dataset we mentioned in the previous section, which makes private score not a general and effective metric for the contest. Second, since the contest is closed, both public score and private score are open to public and the organizer of this competition also release some of the labels of the private score test data (the *stage2_private_solution.csv* contains 125 labeled data of the private score test). Thus, we have to note that it is possible, for competitors to conduct excessive tuning on their model using the scores and the released data as feedback, in order to get their rank higher on the leaderboard. Moreover, we also want to report that there is one time in our experiment that we mistakenly predicted all the test data to one class with same probability but surprisingly found that this result even outperform the best result (details of this can be refer to our experiment section 7). It is obvious that predict all the data to the same result is meaningless. Therefore, we think that now the private score may not be fair to evaluate the results any more. So in submission evaluation, we choose to care more about the public score, which is calculated on a bigger test set with unreleased label and therefore is more accurate and challenging. Besides, we also use local evaluation (as mentioned below) based on 10-fold validation to further support our evaluation of different methods.

4.3 Local Evaluation

Despite the fact that the label of test set is unknown to us, we are still able to evaluate our models locally using the following metrics through ten-fold stratified cross-validation.

4.3.1 Accuracy (ACC)

Accuracy measures how much of the prediction is correct based on the ground truth label.

4.3.2 Confusion Matrix

Confusion matrix, also known as error matrix, is widely used in the fields of machine learning and specifically in classification. Each row of the confusion matrix indicates the probability of the records belonging to certain class being classified into each of all classes.

In classification system, a confusion matrix can be a clear indication in which classes the classifier have difficulties distinguish and what the common mistakes are and the corresponding statistics are available as well.

5. Feature selection

In this section, we will introduce the idea, implementation and reasoning of feature selection in our system. Our general idea is to extract and concatenate the features of all three parts of the data to composite the overall representation of each record.

5.1 Sparse encoding

According to the data overview, we know that there are finite number of genes and variations. It inspired us to use one-hot encoding to vectorize these two attributes. We also applied PCA to compress the length of these two feature vectors to length 25 for each. This number is set empirically but we believe it already contains all information we need given the long-tailed distribution of gene and variation. The experiments also support our conjecture when we increase the length of the gene and variation and found that the performance increase is small enough to ignore. To best of our knowledge, we believe the length setting is reasonable for gene and variation so far.

5.2 wording embedding

Now our task is deal with the text features. More specifically, we want to embed each long text into a vector and then use it as our classification model. As states before, compared with TF-IDF and BOW, doc2vec has its advantage that it not only considers the frequency of each word, but also contains the content information. Compared with the deep-learning based method such as LSTM, doc2vec use shallow network so it is easier for training and also more suitable for this task with small dataset. Therefore, our experiments and improvement mainly employ the doc2vec method. However, as shown in the experiment part, we have also tried some of those other word embedding methods and compare their results with word embedding.

Logically, using doc2vec to get the document vectors can be divided into 2 parts: first training word vectors for each word and then getting the vector of each text document based on the word it contains. So typically, there are 2 ways to get the final document vector. One way is to train the word vectors and document vectors together using the training data. While another way is to load the word vectors that is pre-trained on some big corpus and only trained the document vector based on these pre-trained word vectors. As one can imagine, since the pre-trained word vector is based on a big corpus, they are expected to outperform the locally trained word vectors in capturing the words' latent representation. Therefore, the second way is expected to have better results (especially when the task has a small training set). However, some literatures[1] also points out that since word2vec implementation has a (somewhat hidden) "reduce-vocab" function that triggers rare-word removal when the size of the corpus crosses certain threshold, using big corpus may mistakenly remove some useful words in the training process and degrade the equality of the word embedding. In our experiments, we tried both way and compare their results.

In our experiment, we used the word vector trained by Chiu etc.[1] These word vectors are trained on the PubMed database, which has more than 25 million citations that cover the titles and abstracts of biomedical scientific publications. The url of the pre-trained word vectors is https://github.com/cambridgeltl/BioNLP-2016. They have 2 version of word vectors with different windows length (win = 2 and win = 30). In their paper[1], Chiu use both extrinsic and intrinsic tests to judge the quality of the word vectors. Extrinsic evaluation evaluates embeddings in an NLP application or task while intrinsic evaluation tests the quality of representations independent of a specific NLP task [3]. Chiu etc. points out that while word vectors with bigger window size works better in intrinsic evaluation, it may act worse than the vectors of smaller size. One explanation of this phenomenon might be that a larger window emphasizes the learning of domain/topic similarity between words, while a narrow context window leads the representation to primarily capture word function[2]. In our experiments, we also compare the effect of using both windows size. Since our project is task-based. We mainly using the extrinsic evaluation, i.e. we judge the quality of the word embedding based on the result of the final classification.

6. Multi-class Pattern classifications

Since we got the feature vector, we step into classification. In general, for each data, we use the concatenate the 3 parts of features (PCA of gene encoding, PCA of variation encoding and the document vector of the text proof) together as the input vector. We feed the vector into different classification model and we predict the probability that this data belongs to every category. As we mentioned before, since we mainly compare 3 classification models here:

1. Baseline KNN model:

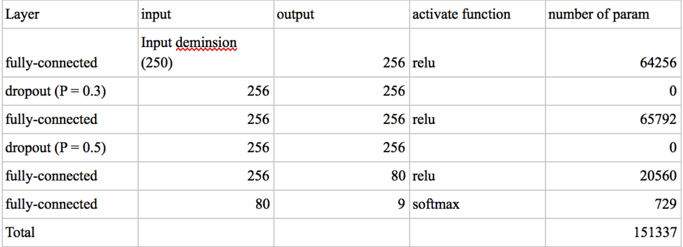
We do prediction of a data by finding its K nearest neighbors. (k = 3) The final probability of this data is the average of the one-hot code of all its neighbors. As one can imagine, this method is very simple and the probability of some categories may be predicted as zero if the data doesn't have any neighbor in this category. Therefore, the cost of mistakenly predict may be very big (deal to take log of zero). As you can see later, this leads to a big log-loss.

This model serves as our baseline model and it help us to get some intuition of the spatial distribution of the data in the feature space and also help us to compare different word embedding methods.

2. Neural network model:

Neural network is commonly used as the classification model deal to its expressive ability. However, it also suffers from the fact that the neural network contains of many parameters, so it is usually not easy for training and may have over fitting. This becomes more serious if the training dataset is small, like in this task. So here, instead of using those complicate structure, we choose a small shallow fully connected neural network, which contains 5 layers. And we use dropout in the model to reduce over fitting. Since we directly use document vector as input which already contain the content information, we don't need to use the CNN structure to extract the locally relationship between each word as Baker etc. did[4].

For the model structure we refer the setting of kaggle kernel by Mr. Aly osama https://www.kaggle.com/alyosama/doc2vec-with-keras-0-77. But we use Adadelta[5] as optimizer for it can automatically adjust the learning rate during training and therefore get better result. The initial learning rate is 0.5 and ρ is 0.95. The structure of the model are shown as follows:



Layer	input	output	activate function	number of param
fully-connected	Input deminsion (250)	256	relu	64256
dropout (P = 0.3)	256	256		0
fully-connected	256	256	relu	65792
dropout (P = 0.5)	256	256		0
fully-connected	256	80	relu	20560
fully-connected	80	9	softmax	729
Total				151337

Table 6.1 Setting of neural network model

3. Xgboost model:

As we say in the previous section, xgboost take the build the final classifier based on many weak classifiers (trees with limited height) and it also employ the regularization term to further reduce the over fitting. Therefore, we think it will work well for our task. Besides, we also employ the early step mechanism by split the training set into training and evaluation part and monitor the performance the model get on the evaluation set. The parameter settings of our xgboost model are as follows:

Max depth of the weak classifier	8
Maximum number of boosting iteration	1000
Objective function	Multiclass softmax probability
Early stop round	100

Table 6.2 Setting of the xgboost model

7. Experiments and Analysis

In this section, we show our experiment results on classification model selection, parameter tuning and comparison against other state-of-arts in the following subsections.

7.1 Locally evaluation of different doc2vec settings

In this subsection, we compare the results using doc2vec model using 4 different parameters setup (with/without loading PubMed corpus; windows size = 2 / 30). We do this evaluation by running 3 different classification models on each setting: K-Nearest-Neighbor (KNN), Neural Network (NN) and xgboost. For each combination, we first do the locally evaluation using 10 fold validation. The results are shown in Table 7.1. (Our pre-trained document vector can be got from this google drive: https://drive.google.com/drive/folders/1703i996nsfiDldvK8_aTT1G2nX4i1Qnu?usp=sharing)

Models	win = 2		win = 2	
	with PubMed		without PubMed	
	ACC	Log score	ACC	Log score

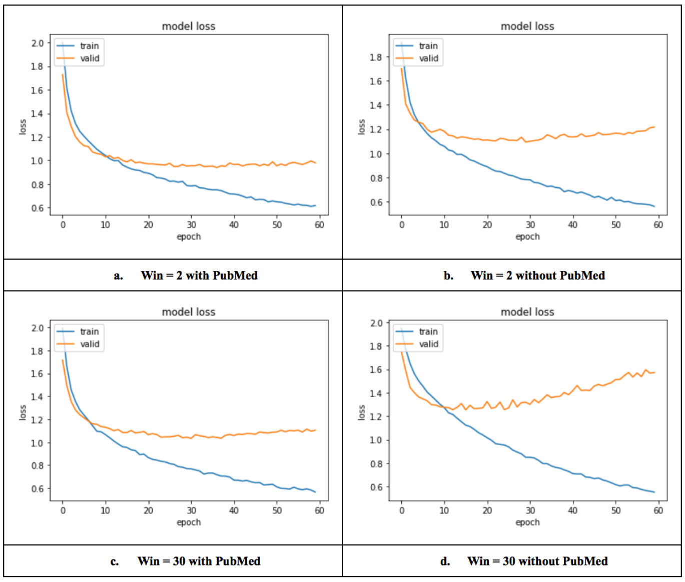
KNN	0.6191 ± 0.0213	6.5382 ± 0.4916	0.6097 ± 0.0242	6.6050 ± 0.4672
NN	0.6618 ± 0.0248	0.9866 ± 0.0873	0.6598 ± 0.0170	1.0273 ± 0.0726
Xgboost	0.6791 ± 0.0205	0.9220 ± 0.0646	0.6791 ± 0.0205	0.9220 ± 0.0646
Models	win = 30		win = 30	
	with PubMed		without PubMed	
	ACC	Log score	ACC	Log score
KNN	0.5853 ± 0.0194	7.1425 ± 0.5590	0.5509 ± 0.0211	8.2882 ± 0.7595
NN	0.6506 ± 0.0134	0.9993 ± 0.0841	0.6301 ± 0.0256	1.2779 ± 0.1272
Xgboost	0.6694 ± 0.0261	0.9426 ± 0.0701	0.6623 ± 0.0199	0.9715 ± 0.0622

Table 7.1 Locally evaluation of Doc2vec on {KNN, NN, Xgboost}

From Table 7.1, if we compare the left part (with PubMed) with the right part (without PubMed), we can clearly see that, given the classification model, and the windows length, the model with PubMed works better (or at least not worse than) in the classification accuracy and log score the model without PubMed, which means loading big corpus has more positive influence than negative influence in this task. Besides, if we compare the upper part with the lower part, we can see that in all settings the document vectors trained with narrower window sizes outperform those with larger window sizes, which accords with the results in the literature[1][2].

7.2 Comparison of different classification model

We also compare the performance of different classification models. Since from the final result, we can see that the xgboost model outperform the neural network model and baseline KNN models in all the settings in our locally evaluation. Figure 7.1 and 7.2 shows the training curve of xgboost and neural network.(Actually in 10-fold validation we will get 10 similar training curves for each setting, to save space, we just show one of these curves here) Since KNN don't have training process and its results are obviously worse than xgboost and NN model, we don't show it here.



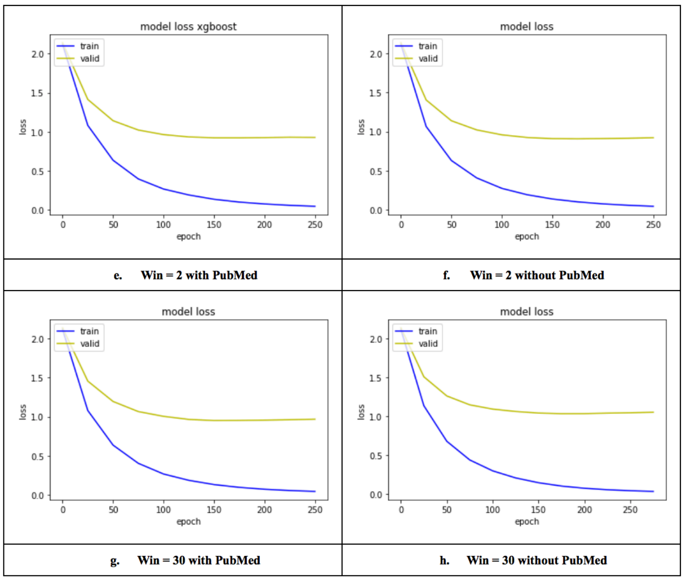
a. Win = 2 with PubMed

b. Win = 2 without PubMed

c. Win = 30 with PubMed

d. Win = 30 without PubMed

Figure 7.1 Training curve of neural network with different settings



e. Win = 2 with PubMed

f. Win = 2 without PubMed

g. Win = 30 with PubMed

h. Win = 30 without PubMed

Figure 7.2 Training curve of the xgboost

From Figure 7.1 we can see that although we already choose a simple model. The over fitting is still quite obvious here. The validation loss tends to increase only after the first several iterations (usually at epoch 5 to 10), which means in most training time, the model is just trying to fit the noise. The learning curve of xgboost shows better property. First, we see the loss keep going done and finally plateau during the training process; this should thanks to both the anti-overfitting ability of xgboost and the early-stop mechanism. And the final loss is lower than the neural network, which mainly thanks to xgboost itself.

We also conducted some baseline experiments comparing the model performance against other state-of-arts classification models. The best results achieved so far are listed in Table 7.2. And the confusion matrix on ten-fold cross validation on the best-performance Xgboost is shown on Figure 7.3

Model	ACC	Log score
GBDT	0.5752 ± 0.0266	3.7298 ± 0.0481
Naïve Bayes	0.5216 ± 0.0268	2.9548 ± 0.0481
SVM	0.6568 ± 0.0163	0.9629 ± 0.0481
NN	0.6618 ± 0.0248	0.9866 ± 0.0873
Xgboost	**0.6791 ± 0.0205**	**0.9220 ± 0.0646**

Table 7.2 Comparison against other Classification Models
GBDT(n_estimators=50, learning_rate=1.0, max_depth=3)
SVM(C=1, degree=3, kernal='rbf', gamma='auto')
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)

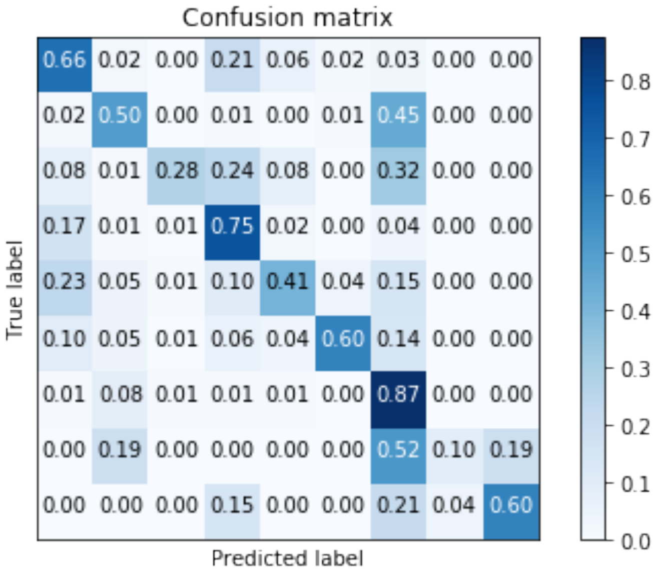


Figure 7.3 Confusion Matrix using Xgboost (win=2, load=true)

7.3 Comparison of different text model

In this section, we compared our word2vec model along with other word models, including Bag of Words and TF-IDF. We conducted the feature selection differently using different text models and then use the same classifier, Xgboost to classify the feature vectors. The results are evaluated using official private score and public score. We can see from the table that word2vec has the best performance.

Method	Private score	Public score
Bag of Words	1.84124	3.83221
TF-IDF	1.48829	3.33638
word2vec	**0.47497**	**3.30170**

Table 7.3 Submission results of different text models

7.4 Submission results of different methods

Method	Public score	Private score
NN win2 with PedMed	1.06085	3.67489
NN win2 without PedMed	1.15167	4.26710
NN win30 with PedMed	1.22998	4.85385
NN win30 without PedMed	1.51563	6.05174
Xgboost win2 with PedMed	0.47497	3.30170
Xgboost win2 without PedMed	0.51468	3.15363
Xgboost win30 with PedMed	0.48911	3.14456
Xgboost win30 without PedMed	0.62125	3.50604

Table 7.4 Submission results of different settings

Here we show the submission result of different settings. (Since the KNN method works bad compared with neural network and xgboost). As we discussed in section 4.2, we thinks the public score is more accurate (we also list the private score and in most comparison, they have similar result as the public score). As one can see, Xgboost outperform the neural network in every setting case. And their difference is more obvious than our local evaluation. We believe this is because the training set and test set both suffers from the unbalanced distribution of data and their distribution might be different. We can also see that the result of loading the pre-trained PubMed word vectors can help the model works better. And the narrower window works better than the larger window. This is similar to our observation when doing locally evaluation.

For the private score, as mentioned in 4.2, we have found that if you predict all the data to the same probability:
[0.11264682578737349, 0.26034555368720336, 0.07818360959353274, 0.10121299942731903, 0.10764782671346829, 0.07602297670755391, 0.11514748484475867, 0.07512451989190694, 0.0736682033468835],
you will scored 1.97191 at the private score and 2.13890 at the public score, which even outperform the best result in this board. We think this ridiculous result mainly because of the very

limited size of the private test data and the unbalanced data distribution of each class. From the released part of private test data *stage2_private_solution.csv* , we find that data in class2, while there is only one data in class 6 and 5 data in class 7. Since we haven't observe such phenomena in the public test data set, so we think the public score is more reliable and the private score should only be used as a supplement of the public result.

In summary, from the series experiment listed above, we prove that in all the settings we try. Using Doc2vec with pre-trained word vectors (windows length equals 2) from PubMed as word embedding and xgboost as classifier given us the best result. Based on the public score, our best performance is 0.47497, which ranks as top 10% among all the teams of this competition.

8. Further discussion and future work

Although we got pretty good results from the method we talked before, we would like to rethink this problem at the end. More specifically, we try to reformulate this task by changing the ways of treating the 3 parts of features. Though deal to limited time, we haven't fully explored this idea and also failed to get positive results. We still think this idea is interesting and thus want to report it here. Some work may be done in the future.

Our method above, like most kernel on this competition, directly concatenate the three parts of the features together as a big vector. This means we treat these 3 parts equally. However, this may not be same as what happen in real life. Let's revisit the 3 features we got: gene location, variation and text features. Those text features, after we carefully search on the Internet, prove to be some abstract or paragraphs from some medical literature. So we think it should act more like a reference that provide us with the knowledge to do the judgment. Think like that, suppose you are a doctor and you are asked to do the judgment of a case with gene and its variation. In order to make your decision, you first see the gene variation and then you will need to search all the related literatures and get useful information. That is, instead of treating all the three features equally, we think that the first 2 features, gene and variation, should be treated as the main features and the text feature should act like a literature database. When solving the problem, we first use the gene and variation relation to find the related literature, and then we extract the useful information from these features to support our decision.

To our knowledge, extract certain knowledge from a given literature is still a challenging NLP problem. But we try a easier method here (We call it enhance method). First, we build a model that use the gene and variation feature to predict the text feature, we called this model a "dictionary", which means when seeing a case, we think of what kind of literature should we refer to.(We use a neural network as this dictionary) Then we search in our training set which documents are close to our prediction. This process is like that we are trying to find the related literature.(We use KNN here and set k = 3). Finally, we concatenate all the related information together and feed them into our classification model. About the what related information to use, we have 2 trials here. One is that we use the document vector of these closely related literatures, which means that we refer to more information in a subset of the big database. Another information is that we use the labels of the related literature in the training set. This can be understand as when we do the prediction, we not only consider the current case information. But also refer to our past experience. We are asking what will the similar cases be classified. More details please refer to our code *enhanced.py* and *demoenhanced.py* . We use the PebMed word vector with windows equals to 2 and the xgboost classifier. The result are as follows (deal to the time, we only do the late submission evaluation):

method	Public Score	Private Score
Using the related document vector	2.08574	3.37184
Using the label of the related case	1.85388	3.73510

Table 8 Submission result of the enhanced method

As you can see, these results are not very positive. But instead of saying this idea is totally wrong, we think the failure is mainly caused by these reasons: First, the dictionary model might not be so accurate. We used a neural network as the dictionary. However, as we talked before, neural network may not be quite suitable with this small dataset. And neural network need carefully setting of the structure and parameters or it might not be able to precisely capture the relation between the gene variation and the related literature. Also, we can also argue how to define similarity between the cases. In our trials, we use the document vector to find the close neighbour. But what if we use other features or word embedding models? For example, for each data, we can see what other gene variation is mentioned in the same literature and we may expect that gene variation mentioned in the same literature has similar mechanism and therefore may belong to same category. Another failure reason may still lies in the unbalanced distribution between each category. Especially when we use the labels of the similar cases. This may result in a more serious overfitting to the training set and therefore not act well for the test set. The result that add more information will lead to worse results can be explained in the training process, the xgboost use same of the noise patterns that only occur in the training set to do the classification (e.g. Some category may occurs more than others). But these patterns may not exist in the test set. Last but not least reason may because in workflow of our 2 trials, there are too many prediction that depends on the result of each other. Like we first use the text to generate the word vector (or load it from outside) and document vector. Then use the document vector to train the dictionary model and use this model to find similarity. Finally, we use all the information we get to do classification. As we can estimate, if each of this 3 part has a accuracy of 80%, then the final result only has a accuracy less than 50%. If one of these part are not so accuracy, the final result will be influenced seriously. And the limited size of the data can aggravate this effect.

In a summary, although we fail to get positive result of this idea. We still believe this idea is interesting and thus may deserve further exploration. The further work may focus on how to find the link between gene, variation and text as well as how to extract useful information from the similar case in the database.

9. Conclusion

In this project, we compared different word embedding methods and classification model. The results shows that the doc2vec which considers both statistical results of words and content information, outperform other methods(TF-IDF, BOW). We also shows that loading the pre-trained word vectors based on the PubMed database can help us improve the result in this task. And the document vectors with narrower contents window shows better result than those with the larger window. Regards to the classification model, our series experiments prove that the xgboost shows the best performance. It beats neural network for the anti-overfitting ability and outperforms KNN, SVM etc by its expressive capacity. Our statistical results on the dataset reveal the problem of the unbalanced distribution of each classes in both training set and test set. This triggers our discussion on the evaluation methods of the results and guide our experiments. Our model with doc2vec and Xgboost ranks top 10% in the public board (which we believe is more convincing now than the private one). Finally, we discuss a new way to formulate this task by rethinking the relationship between each features and we report our 2 trials of this idea. Although the final results of these trials don't outperform our previous doc2vec and Xgboost model, we think it still deserve further study. Further work may relies on finding more effective links between gene, variation and text proof as well as use more power method to extract information from the text database.

Reference

[1] Chiu B, Crichton G, Korhonen A, et al. How to Train good Word Embeddings for BiomedicalNLP[C]// The Workshop on Biomedical Natural Language Processing. 2016:166-174.

[2] Turney, Peter D. "Domain and function: A dual-space model of semantic relations and compositions." *Journal of Artificial Intelligence Research* 44 (2012): 533-585.

[3] Yaghoobzadeh, Yadollah, and Hinrich Schütze. "Intrinsic subspace evaluation of word embedding representations." *arXiv preprint arXiv:1606.07902* (2016).

[4] Baker, S., Korhonen, A., &amp; Pyysalo, S. (2016). Cancer Hallmark Text Classification Using Convolutional Neural Networks. BioTxtM 2016, 1

[5] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).

[6] Mikolov T, Chen K, Corrado G, et al. Efficient Estimation of Word Representations in Vector Space[J]. Computer Science, 2013.

[7] Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality[J]. Advances in Neural Information Processing Systems, 2013, 26:3111-3119.

[8] Le Q V, Mikolov T. Distributed Representations of Sentences and Documents[J]. 2014, 4:II-1188.

[9] Kusner M J, Sun Y, Kolkin N I, et al. From word embeddings to document distances[C]// International Conference on International Conference on Machine Learning. JMLR.org, 2015:957-966.

[10] Yang Z, Yang D, Dyer C, et al. Hierarchical Attention Networks for Document Classification[C]//HLT-NAACL. 2016: 1480-1489.