



Personalized Medicine: Redefining Cancer Treatment

12.07.2017

--- <DataGeeks/> ---

Kamal Nayan Reddy Challa: 904943469

Shivani Soman: 304946159

Shreyas Lakhe: 105026650

Shweta Sood: 905029230

Venkata Sasank Pagolu: 804945548

Introduction	2
Related Work	2
Dataset Description	3
Class Distribution	4
Text Length	5
Length of Text By Class	5
Word Cloud	6
Most Characteristic Words by Class	7
Overview of the Approach	7
Bag of Words and Term Frequency-Inverse Document Frequency	8
Word Embedding Approaches	9
1) Pre-trained glove embedding on 840 billion tokens of web data	10
2) Pre-trained Word2vec embedding on PubMed articles	10
3) Word2vec embeddings induced from our training data	10
4) Doc2vec Embeddings induced from our training data	11
Deep Learning Based Approaches	11
1) Long Short Term Memory Networks	11
Understanding LSTMs	11
2) Convolutional Neural Network (CNN) Based Approach	12
Understanding CNN	12
Machine Learning Classifiers	13
Random Forest	13
Support Vector Machine	13
Gradient Boosting	13
Results	14
Best Performing Model	15
References	17

Introduction

Our project is a part of Kaggle competition [1]. In this project, we aim to classify mutations that cause cancer by examining medical text records. A cancer tumor can have thousands of genetic mutations. The challenge is to distinguish the mutations that contribute to tumor growth from the neutral mutations. This will help in diagnosing cancer causing mutations early and treat the patient's tumor at the preliminary stage. Not only this, an automated system of this kind can help eliminate the effort and time invested by clinical pathologist in manually reviewing and classifying every single genetic mutation based on evidence from text-based clinical literature. We have used machine learning and deep learning models in tandem with natural language processing techniques such as word embeddings, tfidf for classifying texts. We deeply investigated the utility of deep learning models like LSTMs and CNNs for this task. After extensive feature engineering, we have used several machine learning classifiers like Gradient Boosting, Support Vector Machine, Logistic Regression and Artificial Neural networks to classify the extracted features. Training word2vec model on our training data has significantly improved the performance and the developed model in this work has achieved a **score of 1.98 that beats the top score in the Kaggle leaderboard**. We are more than happy that the class liked our work and voted our project as the best. So, we would like to make our contribution by making the code developed for training the models and the word vectors publicly available in this github repository. (<https://github.com/sasankpagolu/msk-redefining-cancer-treatment>).

Related Work

Our project deals with classifying medical texts into categories of genetic mutations which involves efficient text classification techniques. One of the first works in medical text classification used Naive Bayes for classifying patients into syndromic categories based on their chief complaints [2]. Many previous works on text classification used the bag of words [3] and n-gram models for obtaining word representations. The authors in [4] leverage a large amount of unlabeled corpora available alongside labeled training data for classification of malignant liver lesions based on bag-of-words representation for radiology reports. They show that the semi-supervised algorithm, Laplacian SVM can efficiently utilize the unlabeled corpora to improve clinical text classification.

Due to the high dimensionality of the feature space in text classification tasks, various feature selection algorithms have been proposed to efficiently reduce the feature dimensionality. In [5], the authors developed sparse classifiers based on a free text of nursing notes to predict ICU mortality risk. Word embedding approaches like Word2vec[6] and Glove[7] has been found to be performing well for text classification tasks. Recent developments in deep learning have allowed generating context-based rich representation of texts. In this paper [8], the authors present an approach that uses a CNN architecture for classification of medical text at the sentence level. This approach improved the accuracy by at least 15% compared to previous approaches. In [9], the

authors have proposed a Hierarchical Attention Network (HAN) for document classification. Unlike traditional approaches which didn't consider the knowledge of document structure, HAN is designed to emulate the hierarchical structure of documents.

Dataset Description

Kaggle dataset provides four files in total - training_variants, test_variants, training_text and test_text.

1. The **variant files** contain information about genetic mutations. They have 4 columns - ID, Gene (gene causing mutation), Variation (Amino acid change for this mutation), Class (1-9 class where this mutation is classified on). The test_variants file does not contain the last column specifying the class.
2. The **text files** contain clinical evidence that human experts used to classify genetic mutations. The text files are double pipe (| |) delimited files with 2 fields - ID, Text (clinical evidence obtained from various research papers).

Both the text files of training and test datasets are linked with their respective variant files using the ID field. The number of training instances are 3321 and the number of testing instances are 5668 which is 70% more instances than training data.

ID	Gene	Variation	Class
0	FAM58A	Truncating Mutations	1
1	CBL	W802*	2
2	CBL	Q249E	2
3	CBL	N454D	3
4	CBL	L399V	4

Figure 1: Glimpse of Variation File

ID	txt
0	Cyclin-dependent kinases (CDKs) regulate a variety of fu...
1	Abstract Background Non-small cell lung cancer (NSCLC)...
2	Abstract Background Non-small cell lung cancer (NSCLC)...
3	Recent evidence has demonstrated that acquired unipar...
4	Oncogenic mutations in the monomeric Casitas B-lineag...

Figure 2: Glimpse of Text File

The table-1 below shows the most frequent genes and variations in the training and test variants files. There are 264 unique values for Genes in the training data whereas, test data contains 1397 unique gene values. The most frequent genes in the train and test data are completely different. The test data has comparatively less high-frequency genes compared to the train data. The train data contains 2996 unique variations whereas the test data contains 5628 different variations. Like the genes, there are a low number of high frequency variation values in the test data compared to the train data although there are common values in the top most frequent variations in both sets.

GENE				VARIATION			
TRAINING		TEST		TRAINING		TEST	
Dimension: 264 x 2		Dimension: 1397 x 2		Dimension: 2996 x 2		Dimension: 5628 x 2	
Gene	Count	Gene	Count	Variation	Count	Variation	Count
BRCA1	264	F8	134	Truncating Mutations	93	Truncating Mutations	18
TP53	163	CFTR	57	Deletion	74	Deletion	14
EGFR	141	F9	54	Amplification	71	Amplification	8
PTEN	126	G6PD	46	Fusions	34	Fusions	3
BRCA2	125	GBA	39	Overexpression	6	G44D	2
KIT	99	AR	38	G12V	4	A101V	1
BRAF	93	PAH	38	E17K	3	A1020P	1
ALK	69	CASR	37	Q61H	3	A1028V	1
ERBB2	69	ARSA	30	Q61L	3	A1035V	1
PDGFRA	60	BRCA1	29	Q61R	3	A1038V	1

Table 1: Most Frequent Genes and Variations

Class Distribution

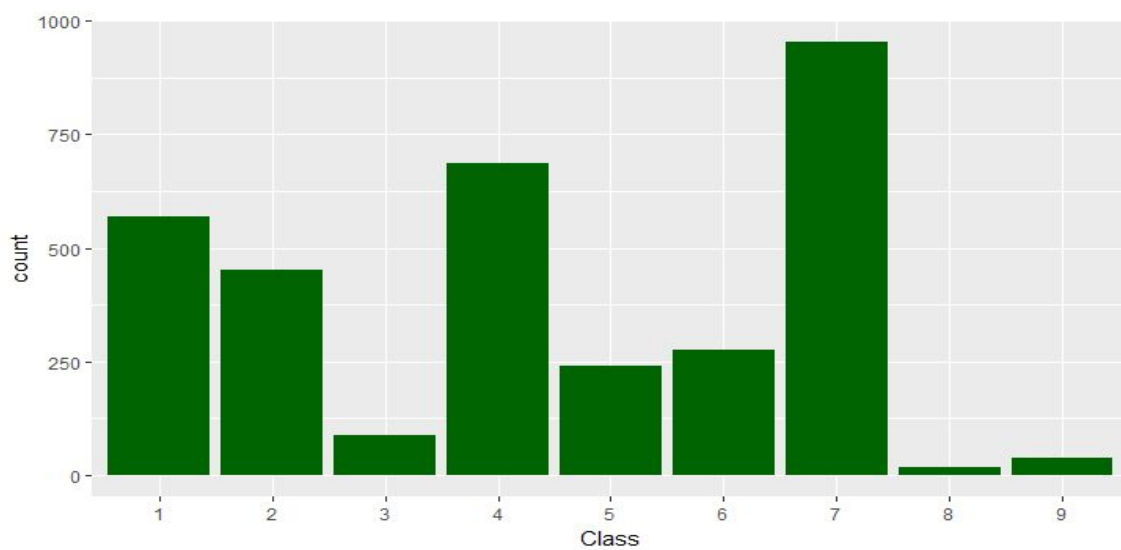


Figure 3: Class Distribution

We can infer from the class distribution that classes 4 and 7 contain significantly higher number of observations. Classes 1 and 2 have medium level frequency and classes 8 and 9 are the least frequent of all classes.

Text Length

We now evaluate the text files *training_text* and *test_text*. The following graph depicts the length of text data in both the files.

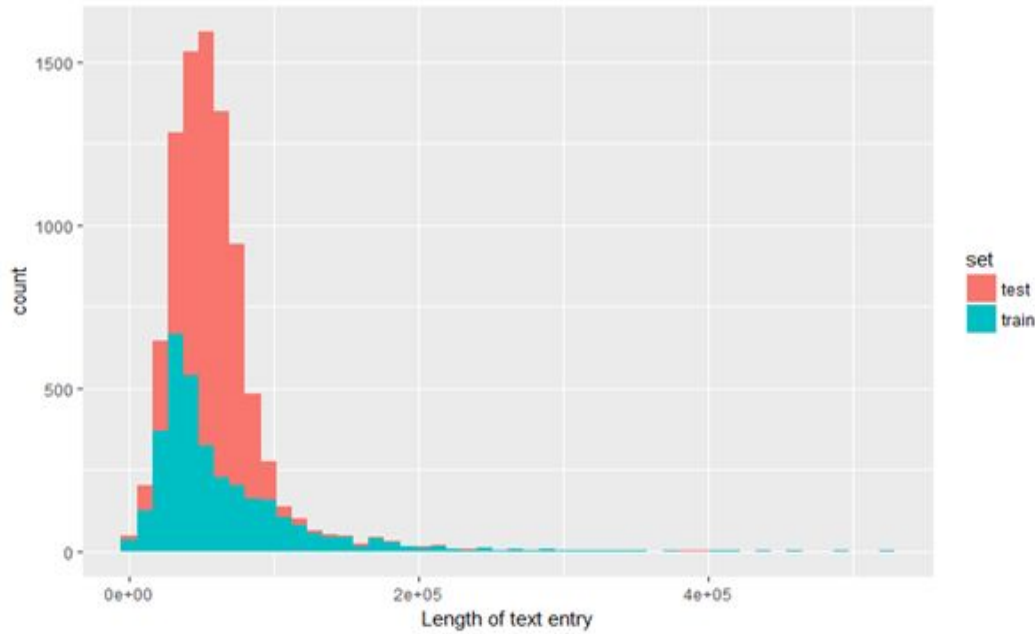


Figure 4: Length of text entry in train and test files

From the graph we observe that most of the entries have text length approximately 6k to 7k in the training data and 4k to 5k in the testing data. The test data text has a lot more variation in length as compared to the training data where more number of entries have similar lengths.

Length of Text By Class

The graph below shows the variation in length of text data in different classes. Classes 3, 5 and 6 have higher density of similar length entries whereas classes 7 and 9 have a lot more variation. Classes 5, 6 and 8 also seem to not have any entries with text length greater than 20K.

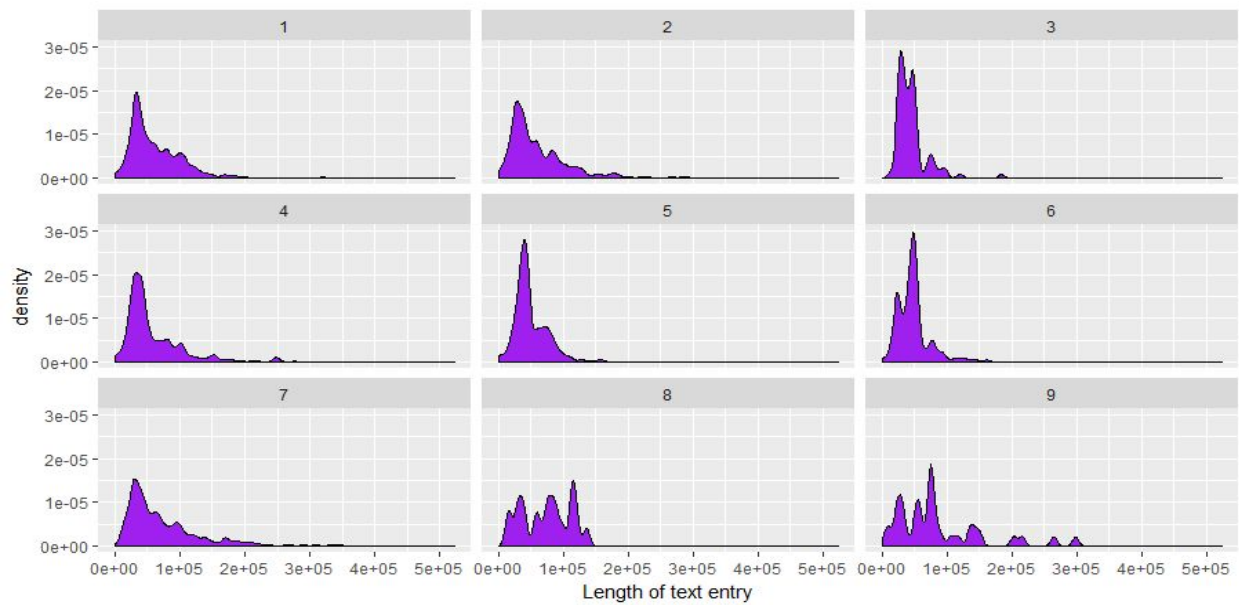


Figure 5: Length of text entry in each class

Word Cloud

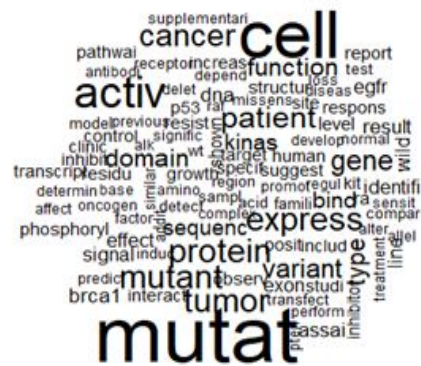


Figure 6: Word Cloud of the most frequent words

The word cloud above shows the most frequent words in the text data after stop word removal of English words. A few words like “figure”, “experiment”, “data”, “table”, “result”, etc. have also been removed as they do not provide any useful information.

Most Characteristic Words by Class

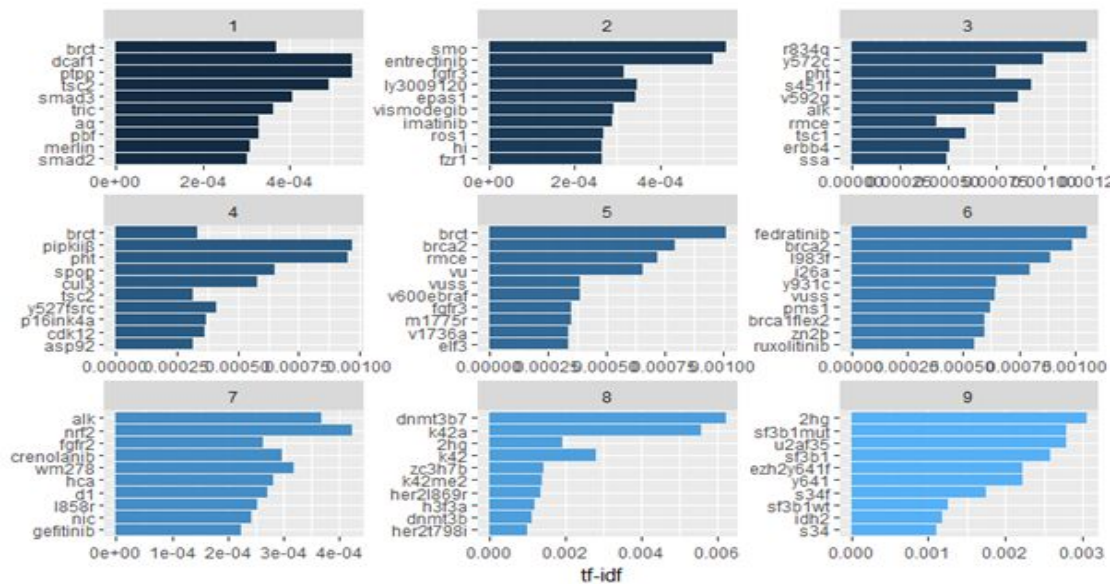


Figure 7: Most characteristic words in each class

The above graph shows the most characteristic words in each class by their tf-idf values. Some words appear in multiple classes like brct, brca2, etc. whereas some are exclusively in a particular class.

Overview of the Approach

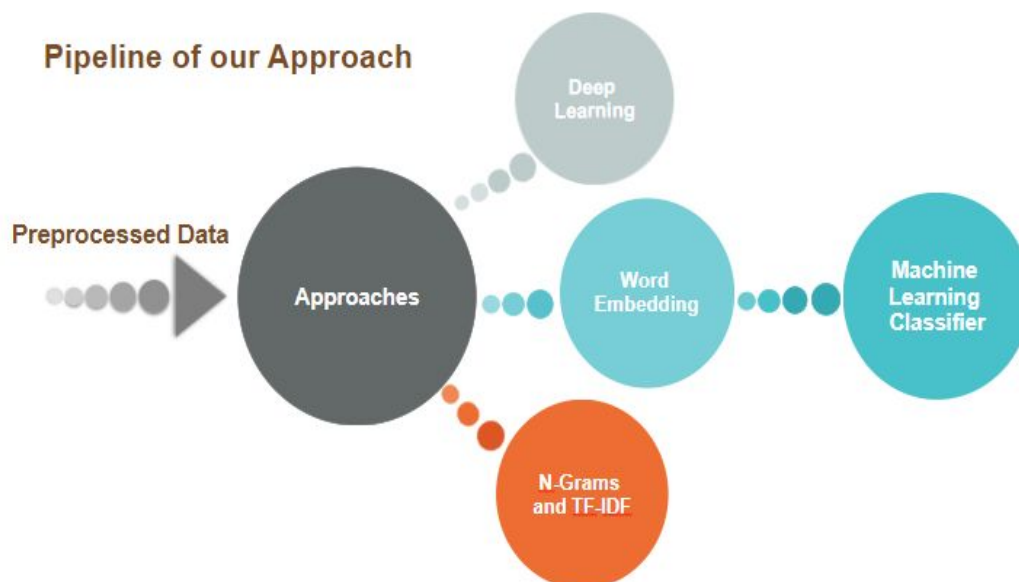


Figure 8: Pipeline of Approach

The approach for the problem can be broadly divided into the following stages:

1. **Pre-processing** - Removal of non-ascii text, stopwords and punctuation
2. **Feature Engineering**
 - a. TF-IDF
 - b. Word Embeddings
 - c. Deep Learning
3. **Machine Learning Classifiers** - Using classifiers like Support Vector Classification, Random Decision Forests, Logistic Regression, Gradient Boosting to obtain final classification of genetic mutations

In the next sections, we would discuss each of these techniques in detail and provide the results obtained.

Bag of Words and Term Frequency-Inverse Document Frequency

We use **Term Frequency and Inverse Document Frequency (TF-IDF)** [10] to analyze both the textual and gene data. Term Frequency (TF) of word ‘w’ measures the number of times the word occurs in a document. However, this metric can be quite misleading as stop words such as ‘is’, ‘are’, ‘the’, etc occur more frequently and don’t convey any important meaning. To overcome this, we introduce Inverse Document Frequency (IDF) with the basic idea to decrease the weight of words that occur frequently in the documents. IDF is formally defined as follows:

$$idf_{term} = \ln(\frac{total\ number\ of\ documents}{number\ of\ documents\ containing\ the\ term})$$

So finally TF of each term is weighted by its IDF value to form a metric which helps in deciding how important a particular word is to document with respect to the entire corpus.

In the current study, we firstly find word level n-grams (unigrams and bigrams) for the entire textual data. Then we find character level n-grams (1-10 grams) for both the gene and variants. This is followed by the calculation of TF-IDF for all the generated n-grams and the resulting vector is considered as our feature vector. Features from TF-IDF are very sparse and to be able to use them efficiently, it is important to reduce the dimensionality. In the present experimentation, we apply Singular Value Decomposition (SVD) to reduce the dimensionality of the feature vector. In addition to the TF-IDF features, we also add features that try to model the different characteristic of the data. The features are listed below:

- Number of words and characters in the text.
- Number of words and characters in the variants.
- Number of words and characters in the gene.
- Number of times the gene occurs in the corresponding text.

- Label encoding of both the gene and variants.

These features were then fed to machine learning classifiers such as random forest, gradient boosting, support vector machines and the results are reported. The architecture of the machine learning model is shown in Figure 9.

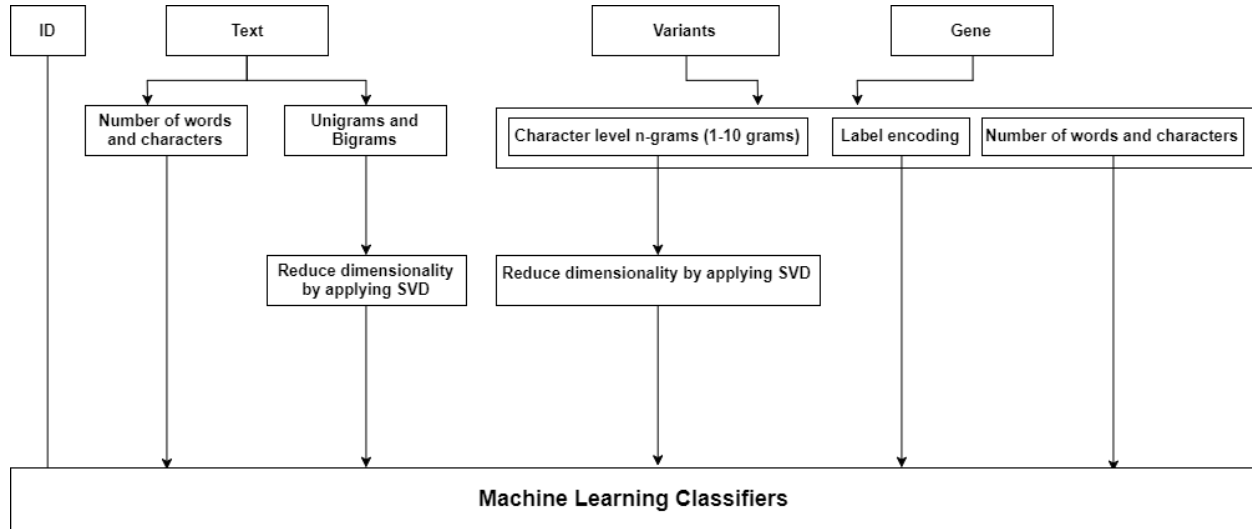


Figure 9: Architecture of TF-IDF Model

Word Embedding Approaches

The data involves analyzing and classifying a text into one of the nine classes of genetic mutations. It goes without saying that understanding the text is the crux of text classification. One of the ways to capture the context is through the use of Word embeddings which allow the classifiers to capture the similarities and relationships between words by mapping words to embedding vectors. Words having the same meaning will have a similar representation. Different models approach the mapping in a different way. We have mainly used three models Word2vec, Doc2vec, Glove and their variants to represent the words as vector embeddings.

Word2vec is developed as a shallow two layer neural network where the word embeddings are obtained from hidden layer weights. Word2vec learns vectors by using two different models, Continuous Bag-of-words(CBOW) and Continuous Skip-gram models. CBOW model learns by predicting the current word based on its surrounding words. Continuous Skip-gram model learns the surrounding words based on the current word. Surrounding words are those that occur in a window of variable size. Using these models, word2vec provides high quality embeddings.

Glove is a different model for learning embeddings by examining statistics across the whole corpus. It works by constructing a co-occurrence matrix representing the frequency of two words co-occurring in the corpus. It derives word vector representations from the constructed matrices.

Doc2Vec[11] is an unsupervised algorithm that learns fixed-length dense vector representation from variable-length sentences, documents, etc. The document/ paragraph vector is trained to predict words in the document. After being trained, the paragraph vectors can be used as features for the paragraph. We can feed these features to various classifiers. The algorithm is Distributed Memory version of Paragraph Vector (PV-DM). It has two main stages: Firstly, training stage that provides word vector, softmax weights and paragraph vectors on already seen paragraphs. Secondly, inference stage to get paragraph vectors for new paragraphs.

We have taken four approaches for obtaining embedding of the given text using the above discussed models and their variants. Along with these vector embeddings, we have joined the features obtained from gene and variant using label encoder which uses one hot encoding.

1) Pre-trained glove embedding on 840 billion tokens of web data

In this approach, word vectors are taken from the glove model trained on 840 billion tokens of web data. These are mappings of words to points in a 300 dimensional space. So, each word has a 300 dimensional vector and we take the average of vectors of all the words present in an instance to arrive at the feature vector for an instance. But, we found that these vector embeddings are not of fine quality as the embeddings are induced by training model on general web data which is not particularly related to our medical text corpus. So, we moved on to our second approach.

2) Pre-trained Word2vec embedding on PubMed articles

In this approach, word vectors were induced by training the word2vec model on PubMed articles database [12]. The dimensionality for the vector embeddings is set as 400. As the dataset at hand consists of elite medical terminology, the idea of taking the embedding trained on medical article data seems reasonable. Similar to the first approach, we take average of vectors mapped to words in an instance and get the feature vector to that instance.

3) Word2vec embeddings induced from our training data

We found that there is a lot of noise in the embeddings derived from the PubMed articles and motivated to train word2vec model on our own training data. We set the dimensionality parameter to be 100. This is a hyperparameter and could be fixed according to the dataset. Google has trained a word2vec model on Google News Corpus with nearly 100 billion words by setting dimensionality as 300 for the embedding. As our dataset is not that huge, we fixed the dimensionality in our approach to be 100. In this way, we reduced the number of parameters that are to be learned and allowed the model to update the embeddings properly. The model considers all the words in the training data and uses an unsupervised approach using context window described earlier. The features for the instances are formed by taking the average of embeddings of all the words in the text of the instance. We observed that the quality of the embeddings

induced are of much better quality.

4) Doc2vec Embeddings induced from our training data

In this part, we generate doc2vec embedding for the input training and testing data. The doc2vec is trained on the training data to generate 100 dimensional vectors. These embedding are then fed to Machine Learning classifiers.

Deep Learning Based Approaches

1) Long Short Term Memory Networks

Given that we had a lot of textual data, we tried modelling the textual data using Long Short Term Memory (LSTM) Networks[13]. Unlike traditional neural networks, which consider the input to be independent, LSTMs are capable of learning order dependence in sequence prediction problems. This makes LSTMs a very good candidate for analyzing textual data.

Understanding LSTMs

Long Short Term Memory Networks (LSTMs) are a variant of Recurrent Neural Network that are capable of learning long dependencies. Unlike RNNs which suffer from vanishing gradient problem, LSTMs are able to get rid of these shortcomings. LSTMs contain a more complicated repeating unit as compared to a simple RNN. Each LSTM cell consists of a forget gate, input gate and output gate. The basic architecture of the LSTM cell is shown in the figure.

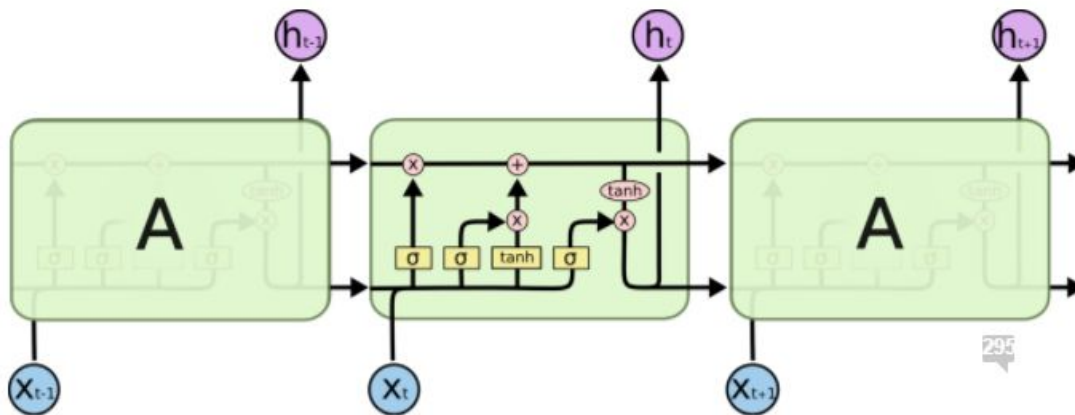


Figure 10: The repeating module of LSTM
(src: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>)

In the current approach, we have used a LSTM-based deep learning architecture to understand the medical text and automatically classify the type of genetic mutation. The LSTM architecture was developed on top of word2vec embeddings which were trained on the training data. The main idea behind using these embeddings is that word vectors trained on the training data

capture the representation better as compared to vectors trained on general domains. Considering the limited number of training instances we used a two layer LSTM network to model the sequential data. These LSTM layers were followed by two dense layers with ‘*relu*’ and ‘*softmax*’ activation respectively for the final prediction. The model was trained for 20 epochs on *Google Cloud Platform*[14].

As each of the training instances had a lot of text, LSTMs trained on word level were unable to capture the long dependencies in it. To overcome this problem, we developed LSTM models to capture the hierarchical structure of the documents[15]. The representation of the document is constructed by first building representations of sentences and then aggregating those to form document representation. The architecture consists of two components: (i) a word level sequence encoder and (ii) a sentence level encoder on top of the word level encoder. The word level encoder constructs the representation of each sentence using a bidirectional LSTM. The encoding of the document is further constructed by another bidirectional LSTM which takes as input the sentence level encoding generated by the previous LSTM. The resulting document representation is used for the classification task. The model was trained for 20 epochs on *Google Cloud*.

2) Convolutional Neural Network (CNN) Based Approach

In recent years Convolutional Neural Networks have been quite popular in the area of Computer Vision. The success of these networks on various Computer Vision related tasks have led people to try them to the area of Natural Language Processing. CNN's already have a few advantages over RNN's. They are much faster than RNN's due to the fact that computing the convolution is less expensive and you can process each convolution in parallel. Additionally they can process whole sequence without needing to process the input sequentially and are able to extract complex feature representations without increasing the complexity of the model. So this makes them a good candidate for handling textual data.

Understanding CNN

The goal here was to apply Convolutional Neural Networks for the problem of text classification.

CNNs work by applying a set of convolutions over the input to generate the output. In each convolution a set of filters with different kernel sizes are applied and their results are combined to be given to the next layer in the network. Additionally max pooling may be used to subsample the output generated by each convolutional layer. The CNN's have been found out to be able to extract rich features from the dataset given a large corpus. This ability of complex feature representation has seen them perform better than the traditional approaches on many similar problems.

The input given to these CNN's was a word level matrix [8]. Each word in the sentence was represented by either using a one-hot encoding with respect to the words in the vocabulary or by using word embeddings. For getting the word embeddings we trained the Word2Vec with a

hidden layer size of 100. We kept the maximum length of the sentence to 200, if the sentences were shorter than this length then we did a post padding on the sentences with value 0 and if the sentences were longer than this length then they were truncated from the end to make them of the given length. We tokenized the document and got an embedding for every word in the document, limiting the vocabulary size to 5000. With the given input, we trained the network to classify into one of the nine classes. For training the CNNs, we tried various configurations. The best results were obtained by adding two sets of two Convolutional Neural Network layers with Relu activation follow. After each set we kept a Max pooling layer and after both the sets a dropout layer with rate 0.5. For each of the convolutional layers we kept the number of filters to 100 and the filter size to 5. At the end, for classification, we kept two dense layers one of size 128 and one of size 9 for the nine classes with relu and softmax activation respectively. In between these two dense layers we added a dropout layer with rate 0.5. To adjust for class imbalance, we adjusted the loss function to penalize according to the class weights based on relative occurrence of each class.

Machine Learning Classifiers

Random Forest

Random Forest [16] is a widely used machine learning technique well known for its efficiency over large datasets. Random Forest is an ensemble of Decision Trees used for classification of data by bootstrap aggregating and feature bagging. To classify a new vector, the input is passed down each of the trees in the ensemble and they vote for a class. Final prediction is made based on the majority vote in the ensemble.

Support Vector Machine

Support Vector Machine (SVM) [17] is a supervised machine learning algorithm which can be used for classification of data. Each data item is plotted as a vector in an n-dimensional space where 'n' represents the number of features in the data set. Then SVM constructs a hyperplane to differentiate the clusters of vectors from each other. Using the kernel technique, SVMs can efficiently perform nonlinear classification. The basic intuition behind the kernel trick is to map the feature vectors to a higher dimensional space where they can be linearly separable.

Gradient Boosting

Gradient Boosting [18] is a machine learning technique used in a variety of applications to generate predictive models from large quantities of data. It is typically used in problems involving classification and regression. In some cases, it is even used as a ranking algorithm like in commercial web search engines, Yahoo and Yandex. It considers an ensemble of various prediction models that are weak and enhances them. Generally, decision trees are the weak learners that are used in the process of gradient boosting. The trees are generated in an additive model where a gradient descent algorithm is used to minimize a differentiable loss function.

Results

Algorithm	Private Score	Public Score
Vanilla LSTMs	3.22	1.31
Hierarchical LSTMs	3.18	1.30
CNN	2.50	2.42
TFIDF + SVD + Gradient Boosting	2.18	1.35
Doc2Vec (Training Data) + Gradient Boosting	3.27	1.13
Glove + Logistic Regression	2.80	1.44
Word2Vec (PubMed) + Gradient Boosting	2.45	1.35
Word2Vec (Training Data) + Gradient Boosting	1.98	1.22

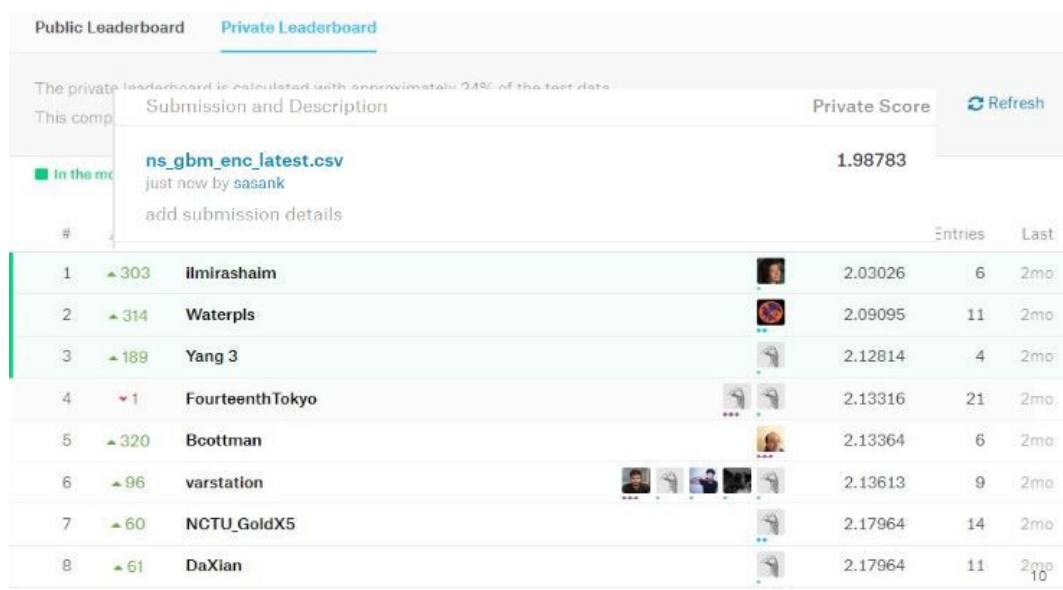
Table 2: Kaggle scores of our approaches

For conducting the experiments we took the dataset provided by the Kaggle Competition and trained and tested our models on it. We used Keras [19], Gensim [20], XGBoost[21] and LightGBM [22] libraries for building our machine learning models, training word2vec vectors and for using gradient boosting classifier respectively. From the results, we can observe that deep learning based approaches like CNN and LSTMs do not perform very well. The main reason behind this is that the model overfits to the training data. Given that each instance contains a bulk of data (5000 words at average) and in addition to that limited number of instances make it difficult for the model to learn a large of parameters that generalize well. Additionally the classes in the data are imbalanced with classes 3, 8 and 9 under-represented in the dataset. Trying normal traditional CNN architectures with a large number of filters leads to the model overfitting the data. The performance of these models can be improved by reducing the filters in accordance with the input size and using dropout layers to avoid overfitting. Similarly for LSTMs, we can observe that the performance can be improved by using hierarchical LSTMs instead of vanilla LSTM as their architecture takes into account the document structure. Also hierarchical model was faster as it used less number of words for each bidirectional LSTM.

The features generated by the TF-IDF approach discussed previously were fed to a variety of machine learning classifiers such as random forest, support vector machines and gradient boosting. Out of all the classifiers, we observed that gradient boosting implemented using LightGBM performed the best. It can be observed that the TF-IDF architecture performs really

well as compared to other complex approaches. The main reason behind this is the quality and variety of features fed to the classifier provide it with relevant information to efficiently do the specific task. Also, by employing feature reduction techniques such as Singular Value Decomposition, we are able to convert the sparse feature representation to dense which significantly helps in improving the performance of the model.

Best Performing Model



Public Leaderboard		Private Leaderboard	
The private leaderboard is calculated with approximately 94% of the test data.			
Submission and Description			Private Score
ns_gbm_enc_latest.csv just now by sasank			1.98783
add submission details			Refresh
#	Rank Change	Submission	Score
1	▲ 303	llmirashalm	2.03026
2	▲ 314	Waterpls	2.09095
3	▲ 189	Yang 3	2.12814
4	▼ 1	FourteenthTokyo	2.13316
5	▲ 320	Bcottoman	2.13364
6	▲ 96	varstation	2.13613
7	▲ 60	NCTU_GoldX5	2.17964
8	▲ 61	DaXian	2.17964

Figure 11: Best Performing Model Rank on Kaggle Leaderboard

The results obtained from the experiments show the significance of using domain specific word embeddings. We are able to improve the results by using embeddings induced from PubMed articles compared to the embeddings from general web data. The performance significantly improved when we have derived the word embeddings from our training data. This strengthens the hypothesis that the quality of word embeddings could be improved as we restrict our corpus attention to a domain closely related to our data. The efficiency of this approach could be understood from the similarities discovered by the model.

We could observe that the model has managed well to capture similar words. The above table shows the words similar to the word 'genetic' learnt by the model along with their cosine distance. Words genomic and chromosomal are found more similar to genetic. In the same way, the model has found the words alteration, polymorphism etc. similar to the word 'mutation'. We also tried Doc2vec approach for generating embeddings. However, this approach didn't yield good results because Doc2vec requires much larger number of documents for training. Though Doc2vec shows a good public score, it performs poorly on the private score. This is because it is overfitting on the public dataset. It is not efficiently able to learn the representation of documents

from the small number of instances and is not a robust model for the provided dataset.

mutation	Cosine Distance	genetic	Cosine Distance
substitution	0.269	genomic	0.638
variant	0.297	chromosomal	0.667
alteration	0.319	molecular	0.799
mutational	0.369	oncogenetic	0.844
polymorphism	0.389	biology	1.061

Table 3: Words similar to ‘mutation’ and ‘genetic’

Below, we have provided the 3D visualization for the 100 dimensional vectors of the words ‘genetic’ and ‘mutation’ along with their closely positioned points in the space. Dimensionality reduction from 100 to 3 is achieved using PCA. We used (projector.tensorflow.org) for the visualization purpose by uploading tsv files of our vectors and corresponding word labels.

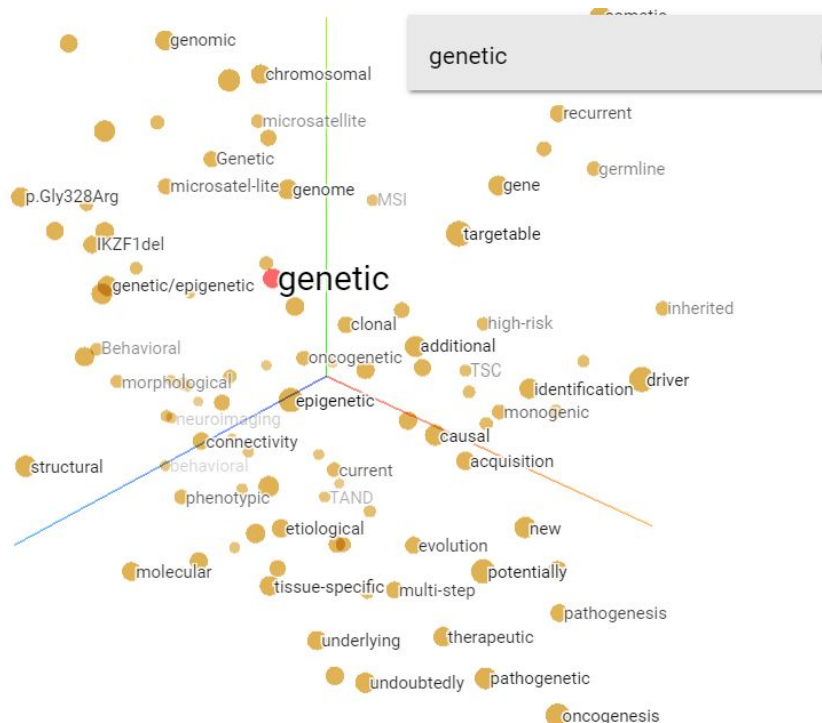


Figure 12: Words similar to ‘genetic’

Conclusion

In this project our goal was to use Machine Learning models to accurately classify genetic mutations from expert annotated knowledge bases and text based clinical literature into a set of predefined classes. For achieving this goal we tried a variety of approaches from using N-grams and TF-IDF, word embedding based approaches to deep learning based approaches. We obtained the best results by training Word2Vec on our dataset and using LightGBM classifier. The approach gave a log-loss of 1.98 on the Kaggle private leaderboard and was **ranked number 1 out of 1386 total teams**. Based on the results that we obtained by trying the various methods, we conclude that if the size of the dataset is good enough, training word embeddings can significantly improve the performance. In addition to that, if gradient boosting methods like XGBoost and LightGBM are tuned properly they can give a significant improvement over the deep learning based approaches when trained on a relatively smaller dataset.

References

- [1] <https://www.kaggle.com/c/msk-redefining-cancer-treatment>
- [2] Olszewski RT. Bayesian classification of triage diagnoses for the early detection of epidemics. Proc 16th Int FLAIRS Conference. 2003:412–6.
- [3] Salton, McGill (1983). Orderless document representation: frequencies of words from a dictionary.
- [4] Garla, V., Taylor, C., & Brandt, C. (2013). Semi-supervised clinical text classification with Laplacian SVMs: an application to cancer case management. Journal of biomedical informatics, 46(5), 869-875.
- [5] Cavnar, W. B., & Trenkle, J. M. (1994). N-gram-based text categorization. Ann Arbor MI, 48113(2), 161-175.
- [6] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
- [7] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).
- [8] Hughes, M., Li, I., Kotoulas, S., & Suzumura, T. (2017). Medical Text Classification using Convolutional Neural Networks. arXiv preprint arXiv:1704.06841.
- [9] Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. J., & Hovy, E. H. (2016). Hierarchical Attention Networks for Document Classification. In HLT-NAACL (pp. 1480-1489).
- [10] Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), 513-523.
- [11] Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on Machine Learning (ICML-14) (pp. 1188-1196).
- [12] <http://bio.nlplab.org>

- [13] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.
- [14] <https://cloud.google.com>
- [15] Du, Y., Wang, W., & Wang, L. (2015). Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1110-1118).
- [16] Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R news*, 2(3), 18-22.
- [17] Suykens, J. A., & Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3), 293-300.
- [18] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.
- [19] Chollet, F. (2015). *Keras*. Chicago.
- [20] <https://radimrehurek.com/gensim>
- [21] <https://github.com/dmlc/xgboost>
- [22] <https://github.com/Microsoft/LightGBM>