Personalized Medicine: Redefining Cancer Treatment

KE5205 – TEXT MINING CA

BALAJI NATARAJ (A0178294N)
VIKNESH KUMAR BALAKRISHNAN (A0178304E)
YESUPATHAM KENNETH RITHVIK (A0178448M)
CHOKKALINGAM SHANMUGASIVA (A0178230J)
ABU MATHEW THOPPAN (A0178303H)
BALAGOPAL UNNIKRISHNAN (A0178398E)
ANJALI SINHA (A0178476L)

Table of Contents

Introduction

Business Understanding

Genetic testing is posed to change the way diseases like cancer are treated by personalizing medicine with precision treatment. But due to the huge amount of manual work required, genetic testing has only partially affected the treatment methods.

One of the uses of genetic testing in cancer treatment is the detection of genetic mutations that drive tumor growth among the various neutral mutations. A cancer tumor can have thousands of different mutations which can be detected in its genetics sequence. The challenge is to distinguish the mutations that contribute to tumor growth (drivers) and the neutral mutations (passengers).

Due to the large number of possible mutations, distinguishing them requires significant domain knowledge and expertise. The interpretation of genetic mutation is done manually by clinical pathologists who review and classify every single genetic mutation by referring to evidence from text-based clinical literature.

The process of personalizing cancer treatment using genetic testing can be significantly improved by automating the process of classifying the genetic mutations by making use of the medical texts.

Problem Statement

The goal of this project is to develop a text-based classification algorithm that can use a knowledge base to learn and classify the genetic mutations automatically. The knowledge base is created by the Memorial Sloan Kettering Cancer Center (MSKCC). It consists of expert annotated examples where world-class researchers and oncologists have manually annotated thousands of mutations.

Additionally, we shall also try to answer some questions by exploratory data analysis. Some of the questions are:

1. Which words are common in texts of each class?
2. Is there a requirement for data engineering to create features apart from the exiting three features?
3. Can we use any pre-trained word2vec model or do we need to train a new model?

Data Understanding

Data Description

The data is collected by the Memorial Sloan Kettering Cancer Center (MSKCC) and is made available on Kaggle as part of a competition (https://www.kaggle.com/c/msk-redefining-cancer-treatment/data). The data has examples of various mutations annotated manually by experts into nine different classes of genetic mutation. Both the train and test data are available separately and are split across four files as follows:

1. Training Variants - a comma separated file containing the description of the genetic mutations used for training. It contains the gene where the mutation is located, the amino acid variation for this mutation and the class for the genetic mutation. It also contains a unique ID for each example.
2. Training Text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. It also contains a unique ID for each example.

3. Test Variants - a comma separated file containing the description of the genetic mutations used for training. It contains the gene where the mutation is located, the amino acid variation for this mutation and the class for the genetic mutation. It also contains a unique ID for each example.
4. Test Text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. It also contains a unique ID for each example.

The Training Variants and Training Text files can be joined using the unique ID of each example. Similarly, Test Variants and Test Text can also be joined.

The data in both training and test data can be described as follows:

No.	Columns	Data Type	Description
1	ID	Integer	This is a unique ID which represents a single example of genetic mutation that must be classified. This is used to join the genetic mutations to the corresponding clinical texts.
2	Gene	String	This contains a string value that represents the Gene where the genetic mutation is located.
3	Variation	String	This is a string value that represents the amino acid variation that occurs for the gene mutation.
4	Text	String	The clinical evidence used by the domain expert to classify the corresponding gene mutation into a class. The length of the text ranges from 275 characters to 448019 characters
5	Class	Integer	The class to which the genetic mutation belongs. There are nine possible classes.

Exploratory Data Analysis

Categorical Data Distribution

The Exploratory data analysis starts with identifying the variation in the data. To begin with, we will identify the unique target classes and their proportion in the train data.

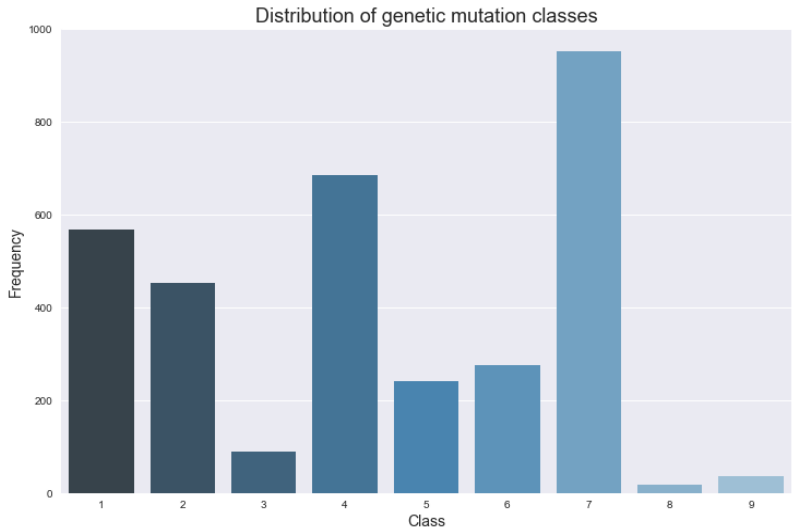

Figure 1 Gene Target Distribution

There are total of 9 classes in the gene-variation datasets. It is evident from the distribution that there is a class imbalance. Classes such as [7, 4, 1, 2] makes more than 80% of the test data. However, classes such as [3, 8, 9] has very few records. This will lead to a bias in the classification as the algorithm will be biased towards these classes with most records, so we need to handle this.

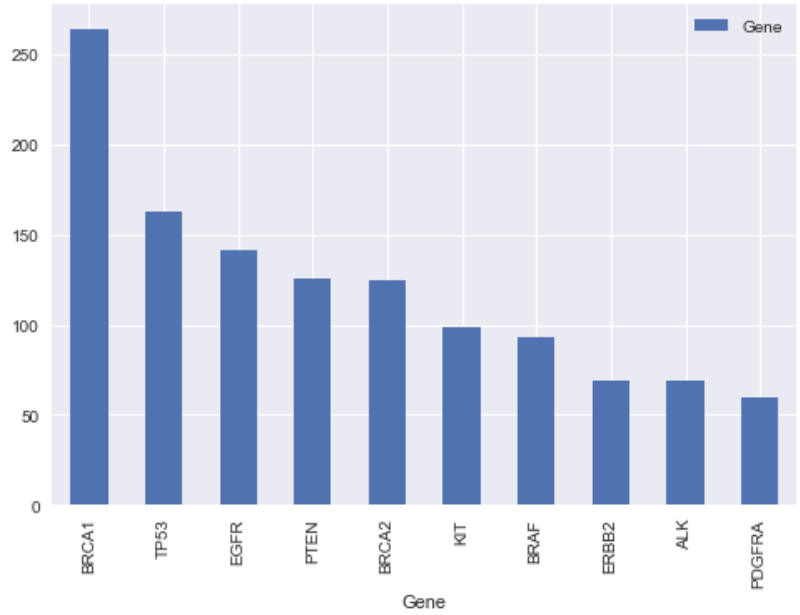

Figure 2 Gene Distribution

Now we need to understand the distribution of Gene. Out of the 3321 records there are 264 unique genes. However, from the distribution we see that few gene sequence like the BRCA1, TP53 have higher chance of occurrence. Compared to them Gene such as KLF4, FGF19 has occurred only once in our training set. If the given test data is a true approximation of the real-world occurrence, this shows few gene sequence occur rarely in nature.

Then the variation in the gene distribution in each class should be studied. We can see that each class consists of multiple gene occurrences. From the distribution it is concluded that each class is related to different gene groups. Thus, the gene group will be an important feature in classifying the classes. WE also observe that few classes like the Clas-5 are highly represented by a single gene variation the BRCA1

In terms of Variation out of the 3321 records there are 2996 unique variations. Thus, the variation in its present form will not be a significant. Thus, feature engineering should be done on the variation strains to extract a meaningful representation which will aide in the classification of target classes.

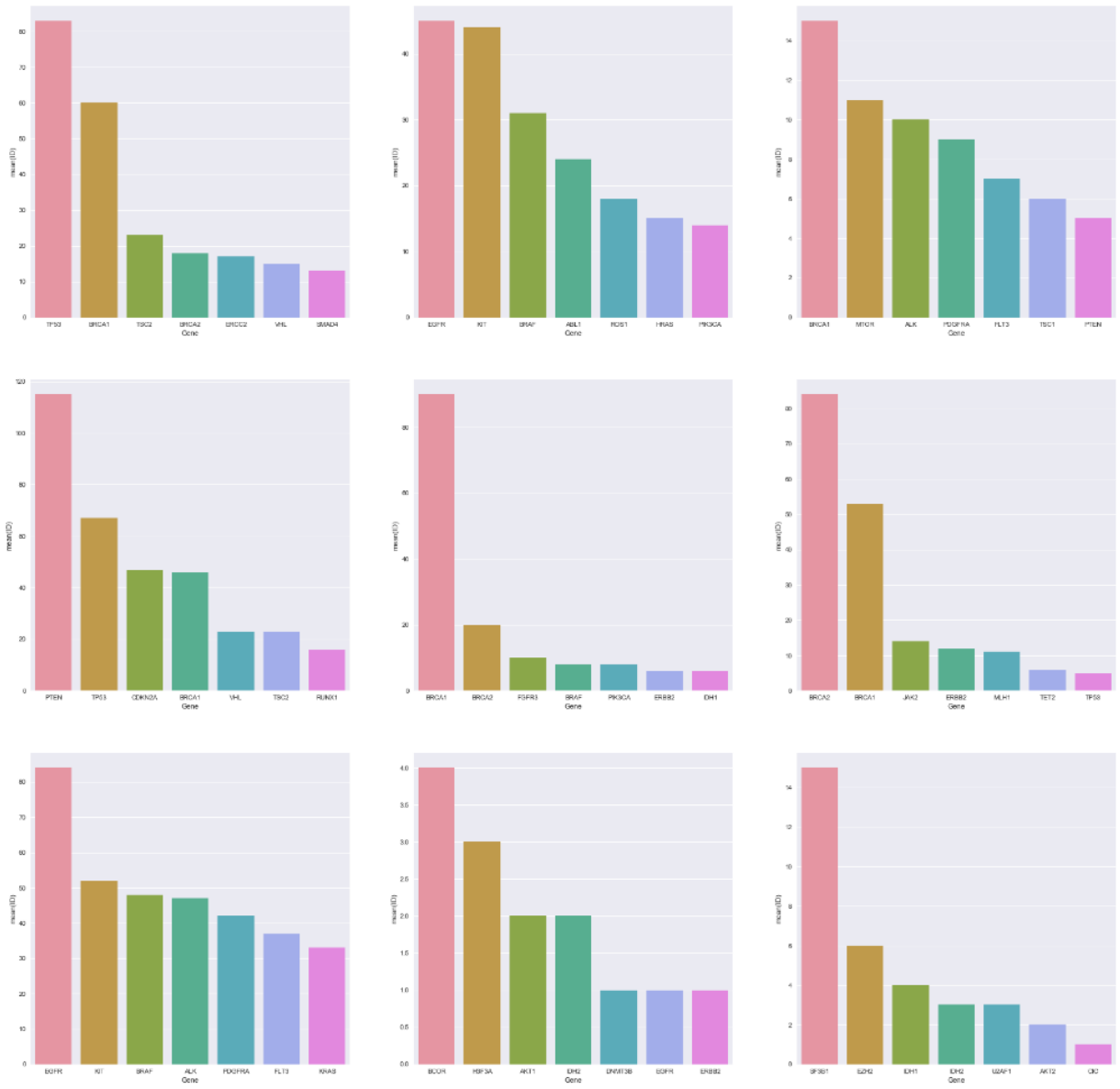

Figure 3 Gene Distribution By Class

Text Data

Text length distribution:

We explore the length distribution of the clinical documents. Most of the documents fall under the 800k to 100k character length.

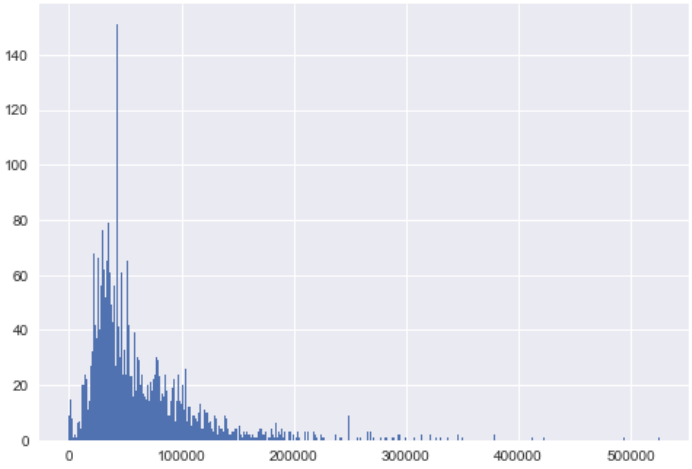

Figure 4 - Doc Length

Then we analyze the length of the document and see the distribution by different target class. As observed most of the document length (# of words) falls between zero and 20k.

From the distribution we can infer that the length could be useful to discriminate some of the classes like class 3, 6 from others. So, the length of each document will be a meaningful input features.

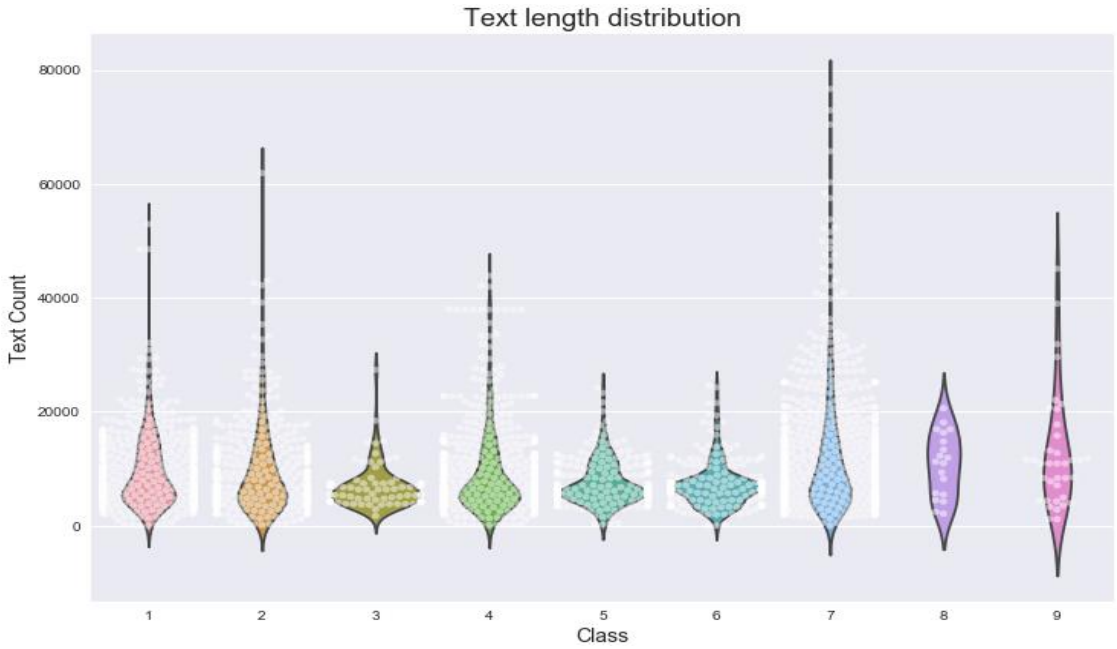

Figure 5 Violin Chart - Doc Length

Tag Cloud:

The most common way to understand text data is to represent is a tag cloud / word cloud. It is a novel visual representation of text data. Each tag is a single word and is represented using different size and color. The Size of each word is a representation of the importance of the word in the document (Frequency). Thus, we get to see the most important words which represent each class, when we generate these tags by class.

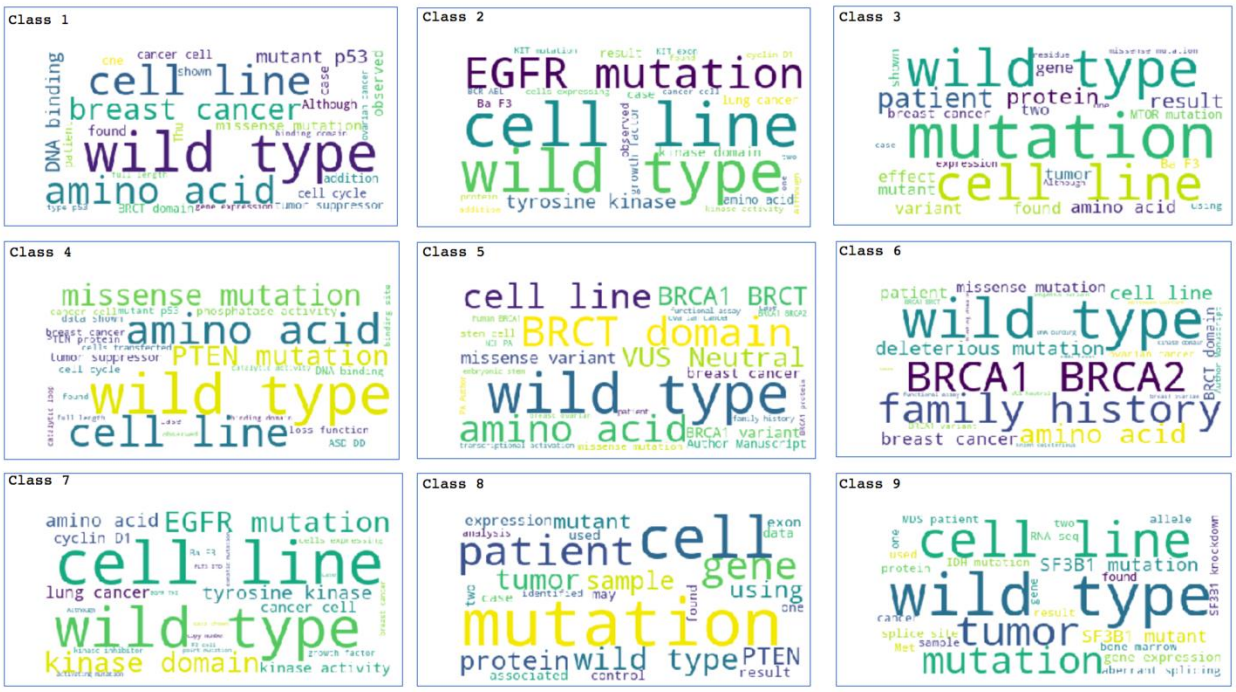

Figure 6 Word clouds generated for each class

From the word clouds we were able to find what was each target class documents talking about.Some Notable Findings:

Class	Key Tags
Class 1	Breast Cancer, P53
Class 2	Lung Cancer, EGFR, Kinase
Class 3	Breast Cancer, Ba F3
Class 4	Breast Cancer, PTEN, Phosphatase
Class 5	Breast Cancer, BRCA1
Class 6	Breast Cancer, BRCA1/2
Class 7	Lung Cancer, EGFR, Kinase
Class 8	PTEN
Class 9	SF3B1

Word2Vec Representation:

The clinical text data can be visualized to show the relationships between words using Word2Vec. Then we will average the word vectors of a document and visualize the relationship between those documents. We'll use genism's word2vec algorithm with Google's pre-trained word2vec model. This is a pre-trained word embedding representation, which was trained on news article.

Using this model a search for gene gave us result such as

['mutations', 'genetic_mutation', 'mutated_gene', 'gene_mutation', 'genetic_mutations', 'gene', 'gene_mutations', 'genetic_variant', 'alleles', 'mutant_gene'] which is still meaning full.

Then for visualizing the data we will have to reduce it to lower dimension. For this purpose, we will be using PCA which gives us a 2D representation. Then we represent this to see how well the clinical documents of each class are separated.

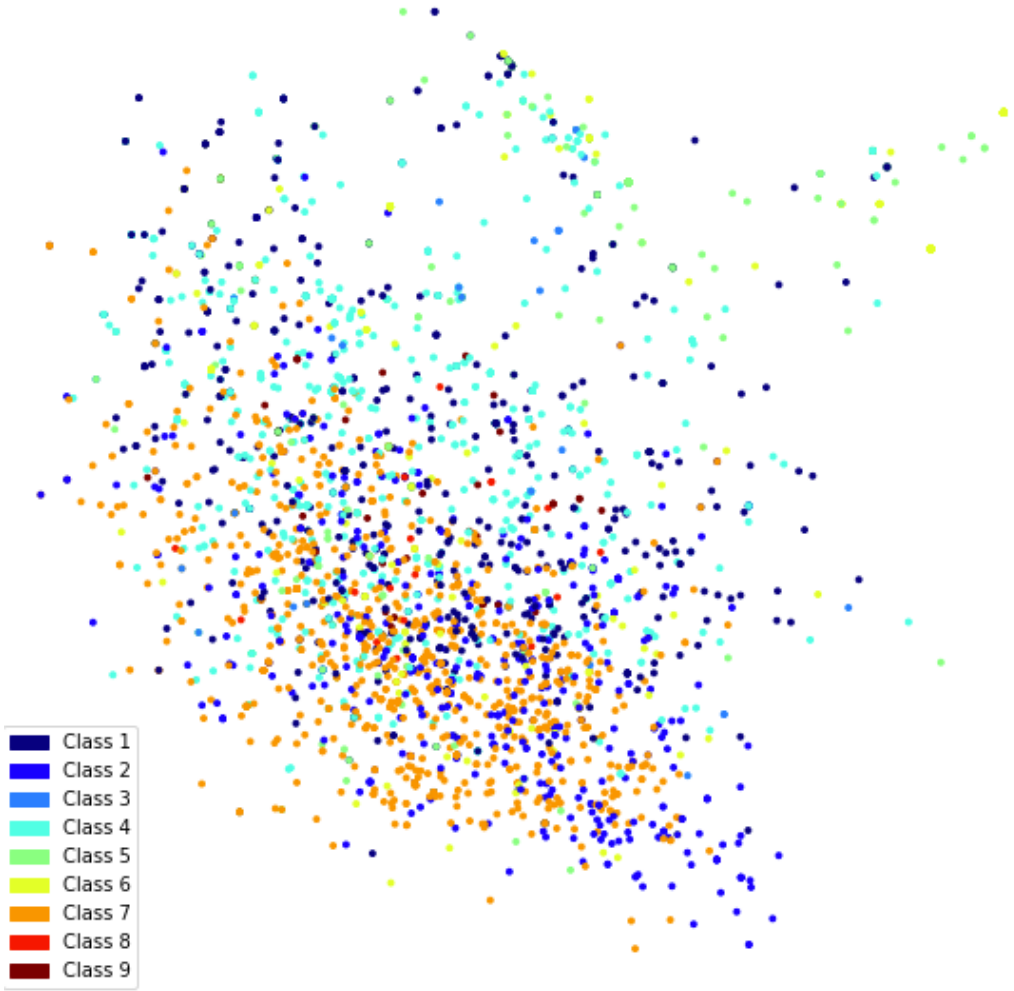

Figure 6 Doc2Vec Representaion

However, this representation does not show any separation between the classes. Thus, it might not be meaning to use this representation for classification. This could be because of two reasons:

We may not be seeing the correlation in 2D after PCA. Or this could be because the word embedding model we are using which is trained on news articles does not give the proper representation for medical data. So there is a need for us to train our own word embedding model which is trained on clinical notes.

Data Preparation

Data preparation is an integral part of CRISP-DM process before the modelling part. Usually, the data obtained will not be in a proper format which will be utilized for creating models in the later stage. Some of the major tasks in data preparation are selecting the appropriate data, clean the selected data for proper values and resolving missing data. As far as selection of data for training/testing, it was already split into parts (Kaggle competition). Rest of the data preparation steps are explained further. Column 'ID' is a unique identifier in the data and does not contribute much to the machine learning process and hence is removed.

Resolving Missing Values

Cancer Mutation dataset had missing values only in the column 'Text' (Clinical notes) for 6 observations. Since it was a textual data, only option was to replace with whitespace or remove the observations as data imputation techniques are mostly applicable for numerical and categorical data. It was decided to remove the 6 missing 'Text' observations as it was less than 1% of the dataset.

Data cleaning

All the data except for the 'Text' column was clean and ready to use. Since 'Text' was obtained from clinician's report, it tends to have the known human error factor. Additionally, special characters were also made the data look even more unclean. To extract better features from this column, proper data cleaning techniques should be employed like removing special characters and <HTML> texts. Below are few examples of unclean data from the dataset.

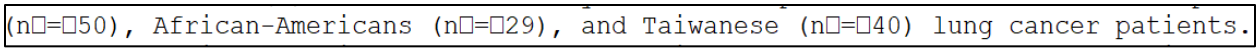

Figure 7: Text with Special Characters

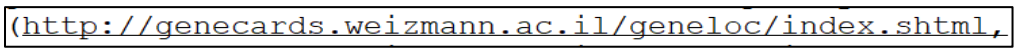

Figure 8: Text with <HTML> text

Feature Engineering

Feature Engineering comes under data transformation part and it is a procedure of extracting useful features from the data by utilizing the domain knowledge to assist the performance of the machine learning algorithm. Below are techniques utilized to generate new features and enrich the existing feature set.

Feature Engineering – 'Gene' Column

Below image gives a brief idea about the data from the 'Gene' column which does not qualify to termed as a categorical or textual data and increases the need for extracting useful feature/information to represent it well.

```
'FAM58A', 'CBL', 'SHOC2', 'TERT', 'DICER1', 'PTPRT', 'RHEB',
'SHQ1', 'CCND2', 'RAD50', 'CCND3', 'RIT1', 'CCNE1', 'RYBP',
'TGFBR1', 'TGFBR2', 'MSH6', 'KMT2D', 'LATS1', 'PBRM1', 'SF3B1',
'LATS2', 'EGFR', 'NKX2-1', 'EIF1AX', 'ARID2', 'BRD4', 'HIST1H1C',
'ERRFI1', 'CHEK2', 'PAK1', 'TMPRSS2', 'H3F3A', 'ELF3', 'ROS1',
'ASXL2', 'CDH1', 'EPCAM', 'EP300', 'EPAS1', 'TP53', 'TP53BP1',
'SMAD2', 'SMAD3', 'SMAD4', 'CDK4', 'AURKB', 'CDK6', 'FBXW7',
'CDK8', 'CDKN1A', 'CDKN1B', 'CDKN2A', 'CDKN2B', 'CDKN2C', 'ASXL1',
```

Figure 9: Sample 'Gene' data

Looking at the data, it is understood that it generally has the combination of few upper-case letters and numbers in different order which can be extracted and represented in a better manner. Below table explains about the features created from the 'Gene' data.

Table 1: Feature Engineering - Gene

Feature Name	Feature Description
gene_char1	Represents the first character of the first textual block after the split based on numbers. Ex.: for gene 'HIST11H29C3', this feature would be 'H' (first textual block is 'HIST')
gene_char1_len	Represents the length of the first textual block after the split based on numbers. Ex.: for gene 'HIST11H29C3', this feature would be 4 (length of first textual block - 'HIST')
gene_char2	Represents the first character of the second textual block after the split based on numbers. Ex.: for gene 'HIST11H29C3', this feature would be 'H' (second textual block is 'H')
gene_char2_len	Represents the length of the first textual block after the split based on numbers. Ex.: for gene 'HIST11H29C3', this feature would be 1 (length of second textual block - 'H')
gene_char3	Represents the first character of the third textual block after the split based on numbers. Ex.: for gene 'HIST11H29C3', this feature would be 'C' (third textual block is 'C')
gene_char3_len	Represents the length of the first textual block after the split based on numbers. Ex.: for gene 'HIST11H29C3', this feature would be 1 (length of third textual block - 'C')
gene_num1	Represents the first number block after the split based on characters. Ex.: for gene 'HIST11H29C3', this feature would be '11'
gene_num2	Represents the second number block after the split based on characters. Ex.: for gene 'HIST11H29C3', this feature would be '29'
gene_num3	Represents the third number block after the split based on characters. Ex.: for gene 'HIST11H29C3', this feature would be '3'

Feature Engineering – 'Variation' Column

As like 'Gene', the column 'Variation' also had some pattern to extract features from them. Below image shows some example values from 'Variation' column.

```
DNA binding domain insertions
C135S
M237K
EP300-MOZ Fusion
A767_V769del
Exon 20 insertions/deletions
```

Figure 10: Sample 'Variation' data

Below table explains about the features created from the 'Variation' data.

Table 2: Feature Engineering - Variation

Feature Name	Feature Description
var_number	Represents any individual number after tokenization if any. Ex.: for variation 'Exon 20 insertions/deletions', this feature would be 20
var_pure_text	Represents all the pure text other than individual numbers and gene combinations if any. Ex.: for variation 'EP300-MOZ Fusion', this feature would be 'Fusion'
var_category	Represents any form of lower-case text occurring along with gene combinations if any. Ex.: for variation 'A767_V769del', this feature would be 'del'
var_combination	Represents any gene combinations if any. Ex.: for variation 'EP300-MOZ Fusion', this feature would be 'EP300-MOZ'
var_comb_char1	Represents the first character of the first textual block after the split based on numbers. Ex.: for variation 'C135S', this feature would be 'C' (first textual block is 'C')
var_comb_char1_len	Represents the length of the first textual block after the split based on numbers. Ex.: for variation 'C135S', this feature would be 1 (length of first textual block - 'C')
var_comb_char2	Represents the first character of the second textual block after the split based on numbers. Ex.: for variation 'C135S', this feature would be 'S' (second textual block is 'S')
var_comb_char2_len	Represents the length of the first textual block after the split based on numbers. Ex.: for variation 'C135S', this feature would be 1 (length of second textual block - 'S')
var_comb_char3	Represents the first character of the third textual block after the split based on numbers. Ex.: for variation 'C135S', this feature would be empty (third textual block is null)
var_comb_char3_len	Represents the length of the first textual block after the split based on numbers. Ex.: for variation 'C135S', this feature would be 0 (length of third textual block - null)
var_comb_num1	Represents the first number block after the split based on characters. Ex.: for variation 'C135S', this feature would be '135'

var_comb_num2	Represents the second number block after the split based on characters. Ex.: for variation 'C135S', this feature would be 0 as no second number block exists
var_comb_num3	Represents the third number block after the split based on characters. Ex.: for variation 'C135S', this feature would be 0 as no third number block exists

Feature Engineering – 'Text' Column

As far as the 'Text' column is concerned, some features were extracted before cleaning data and few were from cleaned text. Below table explains about the features which were extracted unclean data

Table 3: Feature Engineering - Text

Feature Name	Feature Description
text_len	Represents the length of the text data in each document
word_count	Represents the count of tokens in each document
word_density	Calculated value from text_len(len) and word_count(wc) i.e. len/wc
punct_count	Represents the count of individual punctuations and special characters in each document
title_word_count	Represents the count of tokens which are title words in each document
upper_case_word_count	Represents the count of tokens which are completely in upper-case (gene combinations) in each document

Stop word removal

The NLTK toolkit comes with enriched package set to perform some essential tasks. NLTK Stopword package is defined with a set of insignificant words which does not contribute any useful information in the model building and hence it is better remove the stopwords from all the documents to avoid unnecessary introduction of noise into the model.

Table 4: Feature Engineering - Text_stopword

Feature Name	Feature Description
stopword_count	Represents the count of stopword tokens in each document for understanding the noise level in the text data

Figure 11: Stopword – Examples

Stemming and Lemmatization

It is a known fact that the English language has a huge vocabulary and utilizing every unique token in the document corpus would bring in multiple feature and increase the dimensionality of the data. Also, every word may have several derivational and inflectional forms as well. So, here comes the need for stemming and lemmatization techniques to reduce the dimensionality to a little extent.

Stemming is a process of bringing a word back to its original form from the derivational or inflectional form. Lemmatization is a process of grouping the different derived/inflected forms of a word into a single token. There are some disadvantages in both these techniques and to overcome it for a smaller extent, lemmatization was first applied on the documents prior to the stemming process.

POStagger

NLTK also offers packages for getting the parts of speech for the text data like noun, verb, adverbs etc. which can give more contextual representation of the text data. Basically, we have basically used the pos-tags to get few count level features from the 'Text' column.

Table 5: Feature Engineering - Text_postagger

Feature Name	Feature Description
noun_count	Represents the count of tokens which are categorized as noun by the POS tagger in each document
verb_count	Represents the count of tokens which are categorized as verb by the POS tagger in each document
adj_count	Represents the count of tokens which are categorized as adjective by the POS tagger in each document
adv_count	Represents the count of tokens which are categorized as adverb by the POS tagger in each document
pron_count	Represents the count of tokens which are categorized as pronouns by the POS tagger in each document

Once, the text data is cleaned and the other techniques like stopword removal, stemming and lemmatization and postagger were applied on column 'Text'.

Count Vector

The cleaned and engineered set of words that we have obtained so far form our entire vocabulary. Each word in this vocabulary represents a new dimension in our vector space. The documents are all vectors in this multi-dimensional space that are represented by the words.

There are multiple ways to model our vector space. While using Count-Vectorizer, we model each document according to the number of times each word occurs in a document. This is also known as the bag of words approach, this way we can extract quantifiable numeric features from our text data that can be later used for modelling. The frequencies or word counts are all normalized from 0 to 1.

For our vocabulary, we have around 145,350 tokens that are represented by a 3316 x 145,350 matrix where 3316 is the number of training documents we have. The image below depicts each document represented as a vector in a two-dimensional space. The 145,350 dimensions of words are mapped onto two dimensions using PCA. The colors of the document vector are according to the class labels. The magnitude of the vectors is defined by the word counts.

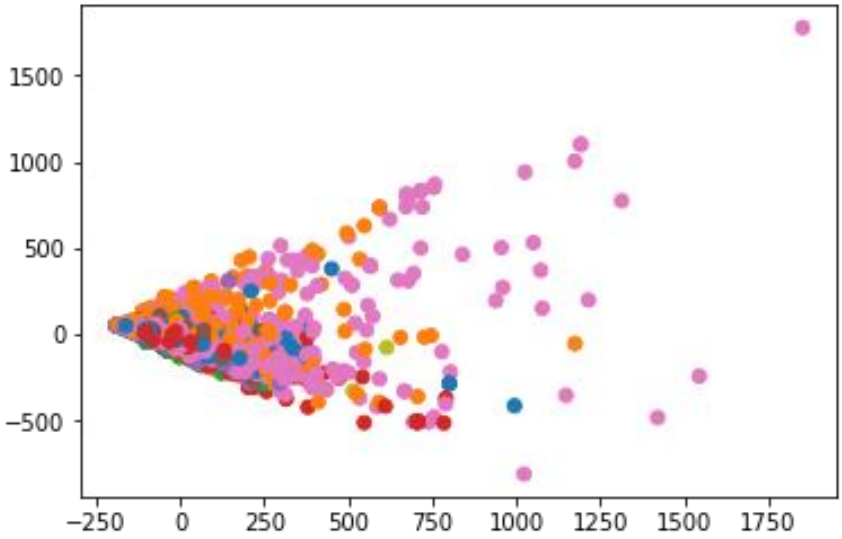

Figure 12 Document vectors represented in 2D space (word counts)

Term Frequency – Inverse Document Frequency

The previous approach takes the word frequency directly. This approach has a downfall, wherein the words that occur a lot throughout all the documents are given more importance while training. The words that occur throughout all the documents are generally of lesser importance while modelling textual data. To counter this, we also perform a count of the number of documents in which the token appears in and then divide the frequency count of each word in a document by this value.

This step reduces the importance of words that occur commonly among all the documents. The image below depicts each document represented as a vector in a two-dimensional space. The 145,350 dimensions of words are mapped onto two dimensions using PCA. The colors of the document vector are according to the class labels. The magnitude of the vectors is defined by the word count – document count ratio.

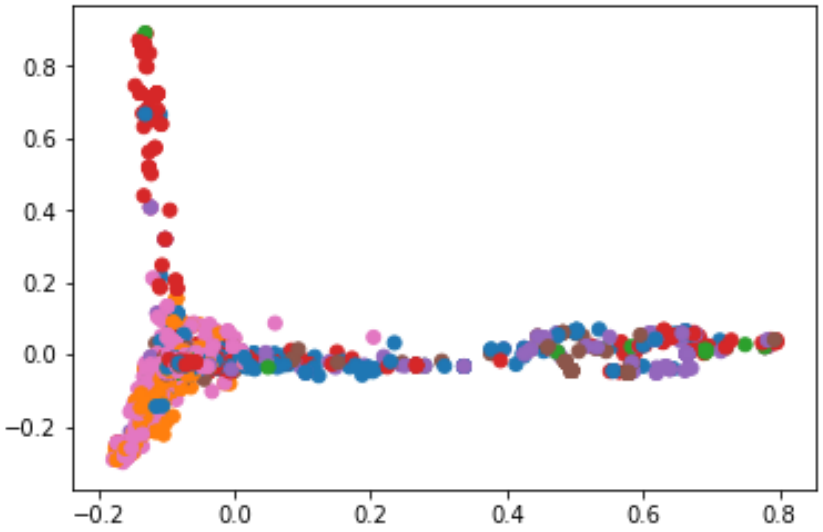

Figure 13 Document vectors represented in 2D space (TF-IDF)

Modelling

At this stage of the project we have come up with different features that describe our textual data about clinical reports, the types of genes and the variation among them. Each of these features can be modelled as a different dimension and can be used as input data while applying machine learning tasks to achieve our results.

The task that we would be performing here is classification of the mutation into 9 classes that have been predefined by medical experts. As we have an initial labelled training data-set of 3321 cases that have been manually labelled by medical experts, this can be used to train different classification models.

Modelling Techniques

We have tried to do the classification process with different types of models and various combinations of hyper-parameters for each model. The models which were most promising have been reported here.

Table 6: Models with Accuracy

Classifier	Accuracy
Naïve Bayes	48%
Random forest	52.26%
Support Vector Machine (linear kernel)	65.6%
Support Vector Machine (sigmoidal kernel)	67.03%
Support Vector Machine (rbf kernel)	62.12%
Word Embedding	68%
Light Gradient Boosting	67.56%
Extreme Gradient Boosting	70.3%

Word Embedding Technique

Word embeddings are a way of representing words and documents as a set of dense vectors. This is quite different from the traditional approach of using the bag of words that treats every word as a different dimension and documents as vectors in that multi-dimensional space. In this approach words are represented by dense vectors. These dense vectors are a projection of the word onto a different vector space. This vector space is learned from the text data available and is based on the surroundings of the text around each word. This vector can be called as the embedding of the word.

Implementation - Word2Vec Averaging

As we have seen in the EDA by using the pre-trained word2vec model we were not able to get a clear representation of the clinical data. So, we trained a word2vec model on the clinical data. This model was used to find the word2vec representation of each word in the document. Then we will find the vector representation of each document by finding the mean of all the word2vector in the document.

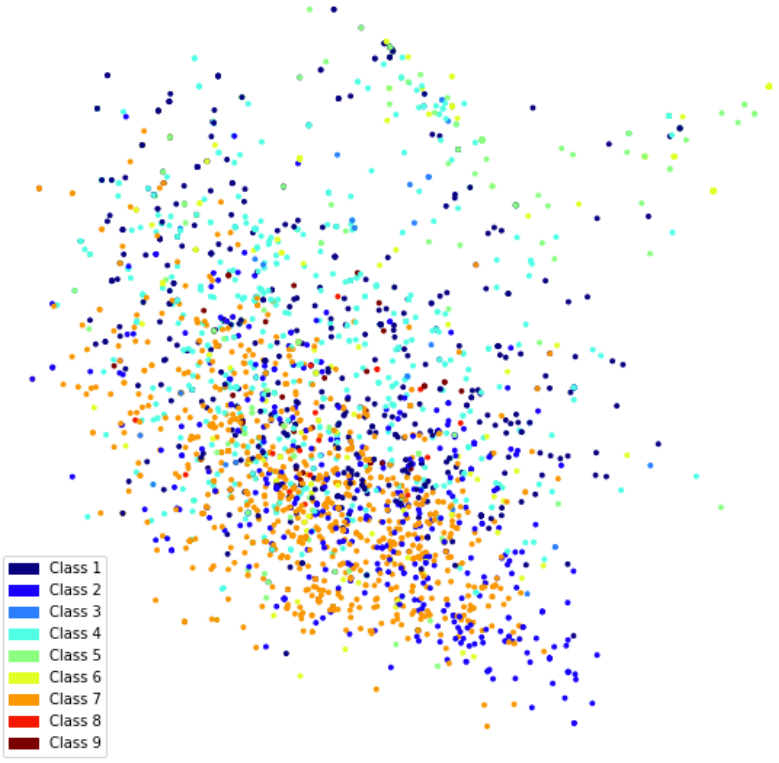

Figure 14 Doc2Vec Representaion

To visualize this representation, we will apply PCA over the document vectors. This gives us a 2D representation of each document. This representation is better compared to the previous representation trained using the Google "News article word embedding model". In this representation the classes are more distinctly seen.

Implementation - Word2Vec Keras layer

Keras provides an embedding layer that can used in the neural networks that work on text data. This layer requires that your entire vocabulary be integer encoded and its weights are randomly initialized. It requires that you specify the dimensionality of the output vector space. We choose a value of 128 dimensions to represent our entire vocabulary. At the end of the training process, the embedding of each word in the vector space is obtained which can be later used in deep learning architectures.

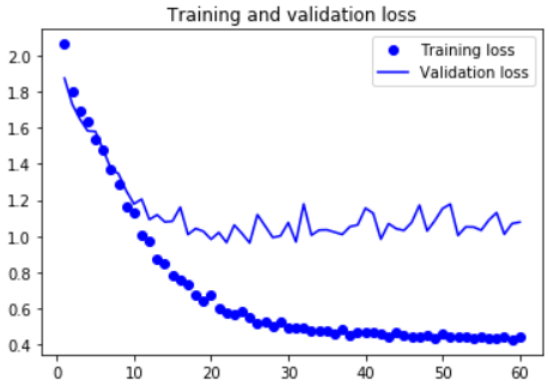

Figure 15 Training and validation loss for CNN model with word embeddings

We used this trained embedding layer to multiple neural network architectures like CNN, RNN and LSTM and obtained our best results from a single dimensional convolutional neural network.

Extreme Gradient Boosting

This is an implementation of gradient boosting decision trees that have been optimized for speed and performance. XGBoost is an optimized distributed gradient boosting technique provided by the XGBoost library for Python and is designed to be highly efficient, flexible and portable. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) under the gradient boosting framework.

The gradient boosting framework interprets the boosting algorithm as an optimization algorithm on a suitable cost function. The algorithms optimize a cost function over function space by iteratively choosing a weak function that points in the negative direction. This is like the boosting algorithm where multiple weak decision trees are trained to perform well on different types of data and then combined.

For this problem, we used the XGBoost algorithm to train over the input features to classify the gene mutations into their respective classes. The hyper-parameters of the algorithm were optimized through grid search to give the best possible result.

Hyper-parameter Optimization

To optimize the hyper-parameters for all the models, we used the 3321 training examples into two parts i.e., train and validation with a ratio of 80:20. Then grid search was performed for each model using a range of values for each hyper-parameter value. The optimized model parameters and the various ranges used are described in this section.

Random Forest

No.	Hyper-parameter	Description	Range	Optimal Value
1	n_estimators	The number of trees in the forest	100, 300, 500	100
2	max_depth	The maximum depth of the tree	10, 15, 20	20
3	max_features	The number of features to consider when looking for the best split	4, 8, 12	12

Support Vector Machine

No.	Kernel	Hyper-parameter	Description	Range	Optimal Value
1	rbf	gamma	Kernel coefficient	1e-2, 1e-4, 1e-5	
		C	Penalty parameter C of the error term	0.001, 0.10, 0.1, 10, 100, 1000	
2	sigmoid	gamma	Kernel coefficient	1e-2, 1e-4, 1e-5	0.01
		C	Penalty parameter C of the error term	0.001, 0.10, 0.1, 10, 100, 1000	1000
4	linear	C	Penalty parameter C of the error term	0.001, 0.10, 0.1, 10, 100, 1000	
5	poly	C	Penalty parameter C of the error term	0.001, 0.10, 0.1, 10, 100, 1000	
		degree	Degree of the polynomial kernel function	2, 3, 4, 5	

Extreme Gradient Boosting

No.	Hyper-parameter	Description	Range	Optimal Value
1	max_depth	Maximum tree depth for base learners	8,12,16	16
2	min_child_weight	Minimum sum of instance weight(hessian) needed in a child	1,5,10	10
3	subsample	Subsample ratio of the training instance	0.5,0.75,1.0	1
4	colsample_bytree	Subsample ratio of columns when constructing each tree	0.5,0.75,1.0	0.5
5	learning_rate	Boosting learning rate	.3, .1, .01	0.1
6	n_estimators	Number of boosted trees to fit	50,100,500,1000	500

Results

- XGBoost Classifier performed better among all the classifiers experimented with maximum accuracy rate of 70.3%
- Feature Engineering from the columns like 'Gene', 'Variation' and 'Text' played a vital role in improving the accuracy as the learnability from the data
- Due to data imbalance, misclassification rate for few classes were high and thus reducing the overall accuracy
- Among all the classes, Class 1, 7 and 9 have the better precision and recall values. Other classes could also be improved with more relevant data.
- From ROC graph, we can infer that all the classes except for Class 8 have better AUC values which signifies a strong classifier

Table 7: Results

Class	Precision	Recall	F1-score	Support
1	0.78	0.73	0.75	94
2	0.61	0.49	0.54	45
3	0.38	0.43	0.4	7
4	0.66	0.74	0.7	65
5	0.64	0.56	0.6	25
6	0.7	0.64	0.67	22
7	0.73	0.82	0.77	101
8	0	0	0	2
9	1	0.83	0.91	6
Avg / Total	**0.7**	**0.7**	**0.7**	**367**

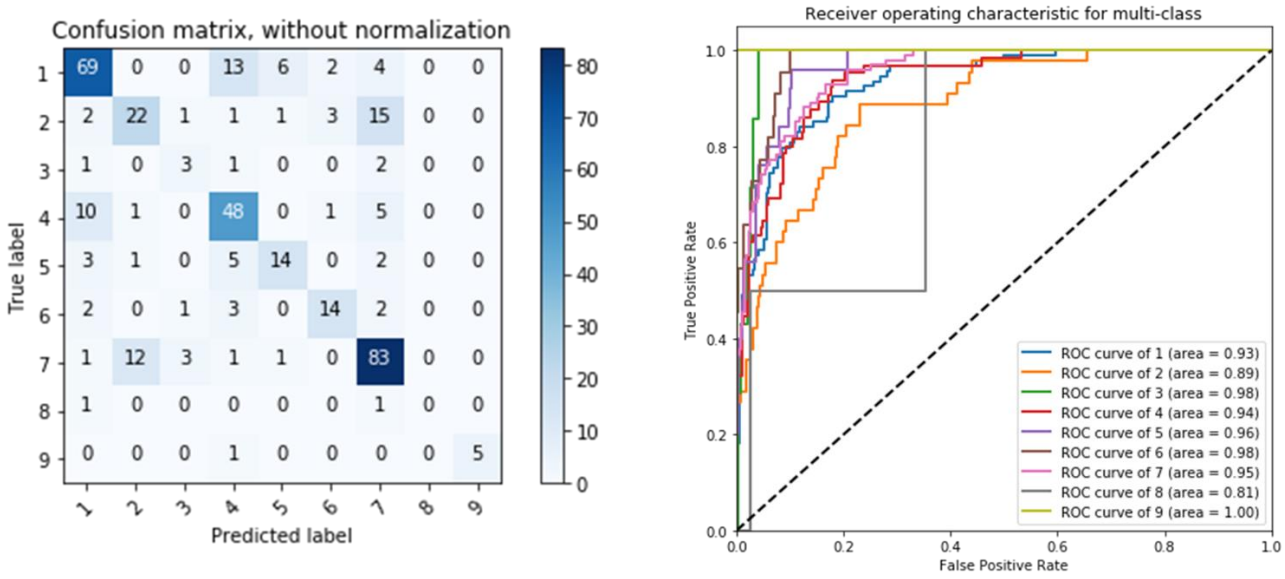

Figure 16: Confusion Matrix & ROC

Conclusion

This assignment has been done to help ease the workload of medical experts while they figure out the type of mutation found and the appropriate treatment that must be given to a cancer patient. While working on it, we were able to get a practical understanding of how different text mining techniques that were taught in class, can be applied to solve real world problems. We were particularly able to get an insight of how we can extract certain features from text data and how we can project words onto different multi-dimensional spaces.

The text data that we worked on is quite complex as it deals with a lot of medical terminology that is even tough for humans to comprehend and understand properly. Considering these aspects and the fact that we have 9 classes to classify, we can say that our model does perform well based on the results obtained. However, there is always a lot of potential for improvement. Some of the things that we could try out to get better results are:

- Coming up with a custom medical dictionary based on expert knowledge.
- Create new or custom named entities based on medical text.
- Create a grammar for how medical details are communicated.

All these improvements would require a lot of effort from knowledge modelers and the experts involved in this field.