Personalized Medicine: Redefining Cancer Treatment

Predict the effect of Genetic Variants to enable Personalized Medicine

Advising Professor
Dr. Handan Liu

Avinash Chourasiya 001427579
Gurjot Kaur 001449692
Rinkal Agarwal 001495613
Rohan Naik 001824278
Vaibhav Raj 001406181

TABLE OF CONTENTS

Sr No.	Particulars	Page No
1	Abstract	2
2	Introduction	3
3	Goals	4
4	Methodology	5-34
	4.1 Data Analysis	5
	4.2 Data PreProcessing	6
	4.3 Merging the Data Files	7
	4.4 Cleaning the Data and Handling the Missing Values	8
	4.5 Creating Train, Test and Cross-Validation Set	8
	4.6 Data Distribution In Train, Test and Cross-Validation Set	9
	4.7 Implementation Strategy	11
	4.8 Data Evaluation Concepts	11
	4.9 Evaluating the Columns	12
	4.10 Data Preparation	16
	4.11 Combining the Features	18
	4.12 Building Machine Learning Algorithms	19-34
	4.12.1 Naïve-Bayes	19
	4.12.2 K-Nearest Neighbors	21
	4.12.3 Logistic Regression	23
	4.12.4 Linear Support Vector Machine	26
	4.12.5 Random Forest Classifier	28
	4.12.6 Stacking Model	32
	4.12.7 Maximum Voting Classifier	34
5	Specifications	35
6	Results	36
7	Conclusion	37
8	References and Sources	38

ABSTRACT

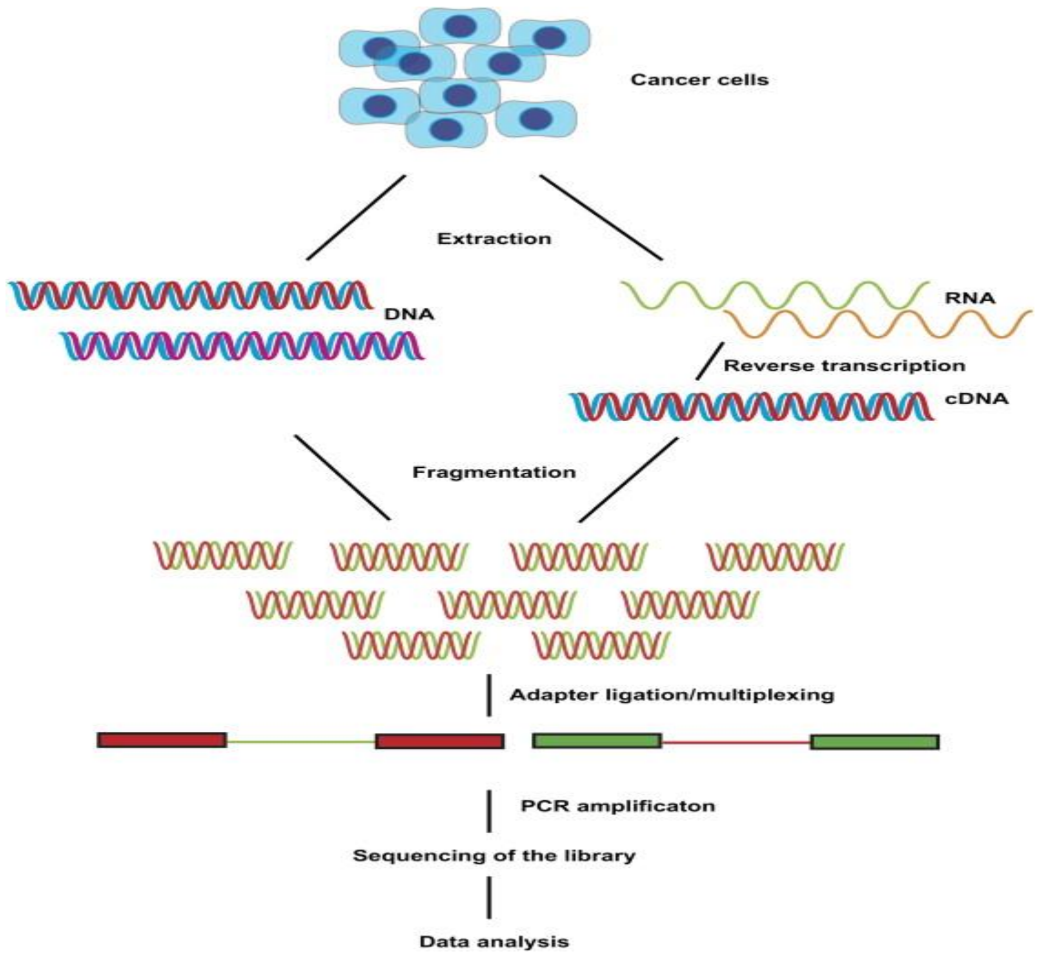
Ascertaining a diagnosis through exome sequencing can provide potential benefits to patients, insurance companies, and the healthcare system. Yet, as diagnostic sequencing is increasingly employed, vast amounts of human genetic data are produced that needs careful curation. We discuss machine learning methods for accurately assessing the clinical validity of gene-disease relationships to interpret new research findings in a clinical context and increase the diagnostic rate. The classes of a gene-disease scoring system adapted for use in a clinical laboratory are described. Our analysis of all the research reports using machine learning algorithms was able to match 65.8% to the gene variations.

Keywords: diagnostic gene sequencing, clinical validity, gene-disease association, gene classifications

Genetic Mutation

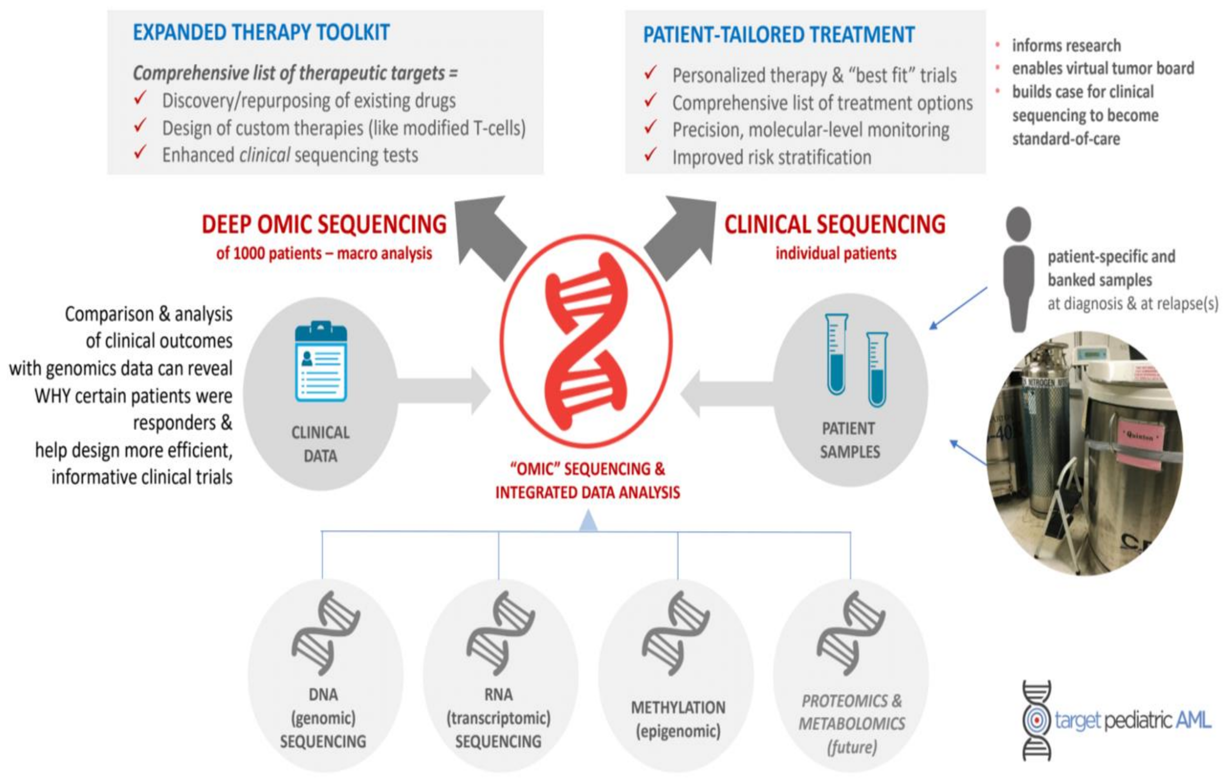
INTRODUCTION

Gene sequencing has rapidly moved from the research domain into the clinical setting. In the past couple of years, a lot of research efforts are concentrated on genetically understanding the disease and selecting the treatment that is most suited to the patients. Genetic testing is one of the groundbreaking precision medicine technique and the way diseases like Cancer are treated. The major hurdle in using gene classification is a huge amount of manual work is still required. Memorial Sloan Kettering Cancer Center (MSKCC) launched a competition, accepted by the NIPS 2017 Competition Track, as help was required to take personalized medicine to its full potential. MSKCC is a cancer treatment and research institute in New York. It is the largest and oldest private cancer center in the world.



A Cancer tumor can have thousands of genetic mutations. The challenge is to distinguish the mutations that contribute to tumor growth from neutral mutations. Currently, this interpretation is done manually and is a very time-consuming task where a clinical pathologist manually reviews and classifies every single genetic mutation based on evidence from the text-based clinical research.

GOALS


EXPANDED THERAPY TOOLKIT
Comprehensive list of therapeutic targets =
✓ Discovery/repurposing of existing drugs
✓ Design of custom therapies (like modified T-cells)
✓ Enhanced clinical sequencing tests
PATIENT-TAILORED TREATMENT
✓ Personalized therapy & "best fit" trials
✓ Comprehensive list of treatment options
✓ Precision, molecular-level monitoring
✓ Improved risk stratification
• informs research
• enables virtual tumor board
• builds case for clinical sequencing to become standard-of-care
DEEP OMIC SEQUENCING
of 1000 patients – macro analysis
CLINICAL SEQUENCING
individual patients
patient-specific and banked samples at diagnosis & at relapse(s)
Comparison & analysis of clinical outcomes with genomics data can reveal WHY certain patients were responders & help design more efficient, informative clinical trials
CLINICAL DATA
"OMIC" SEQUENCING & INTEGRATED DATA ANALYSIS
PATIENT SAMPLES
DNA (genomic) SEQUENCING
RNA (transcriptomic) SEQUENCING
METHYLATION (epigenomic)
PROTEOMICS & METABOLOMICS (future)
target pediatric AML

- The first important goal of classifying clinically actionable genetic mutations is to reduce and minimize the manual efforts put in by Clinical pathologists in identifying and cross-referencing the gene class types from the research literature.

- The second important goal is to evaluate the Log Loss metric for various suitable machine learning algorithms and achieve a model with the lowest Log loss.

- Also, to accurately predict the class of the mutant gene and give recommendations for treatment of patients to the doctors.

METHODOLOGY

Gene has a particular sequence and if any variation is observed, there are chances of developing a benign or cancerous tumor.

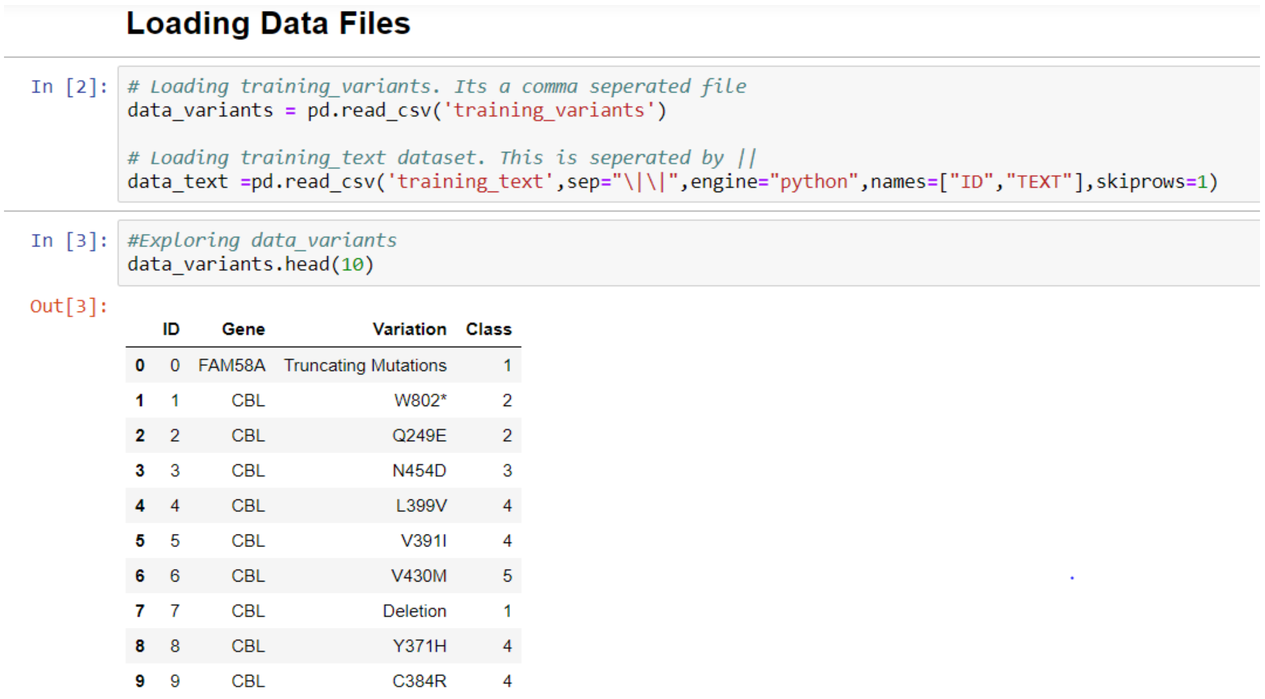
To solve the problem, this process of predicting the class has to be automated using machine learning algorithms.

4.1 DATA ANALYSIS

We have two training files i.e., training_text, training_variants:

1. Training_text has an ID, TEXT columns

2. Training_variants has an ID, GENE, VARIATION, CLASS columns



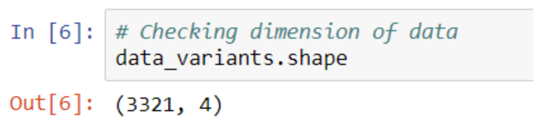
Loading Data Files

```
In [2]: # Loading training_variants. Its a comma seperated file
        data_variants = pd.read_csv('training_variants')

        # Loading training_text dataset. This is seperated by ||
        data_text =pd.read_csv('training_text',sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
```

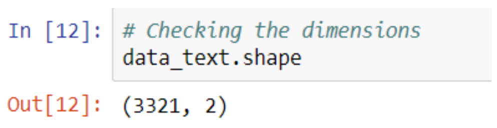
```
In [3]: #Exploring data_variants
        data_variants.head(10)
```

Out[3]:

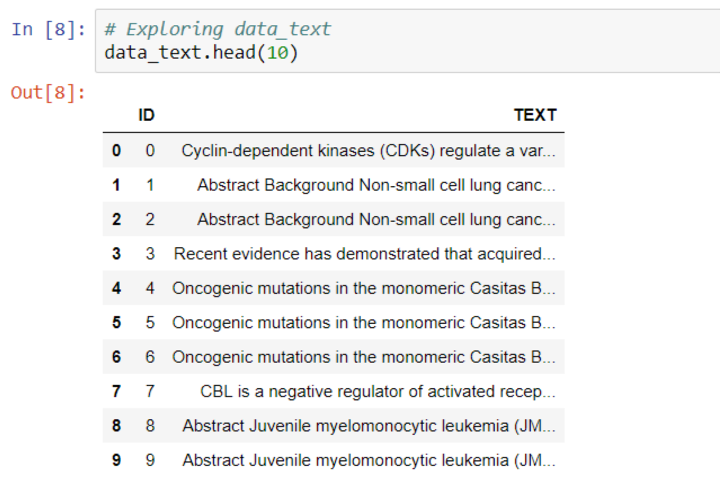
	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4
5	5	CBL	V391I	4
6	6	CBL	V430M	5
7	7	CBL	Deletion	1
8	8	CBL	Y371H	4
9	9	CBL	C384R	4



```
In [6]: # Checking dimension of data
        data_variants.shape

Out[6]: (3321, 4)
```



```
In [12]: # Checking the dimensions
         data_text.shape

Out[12]: (3321, 2)
```



```
In [8]: # Exploring data_text
        data_text.head(10)
```

Out[8]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...
5	5	Oncogenic mutations in the monomeric Casitas B...
6	6	Oncogenic mutations in the monomeric Casitas B...
7	7	CBL is a negative regulator of activated recep...
8	8	Abstract Juvenile myelomonocytic leukemia (JM...
9	9	Abstract Juvenile myelomonocytic leukemia (JM...

The training_text has 3321 rows and training_variants has 3321 rows. The training_variants dataset has 9 unique classes that would be predicted based on Gene, Variation, and Text columns. The output is discrete so it's a **Multi-Class Classification problem**.

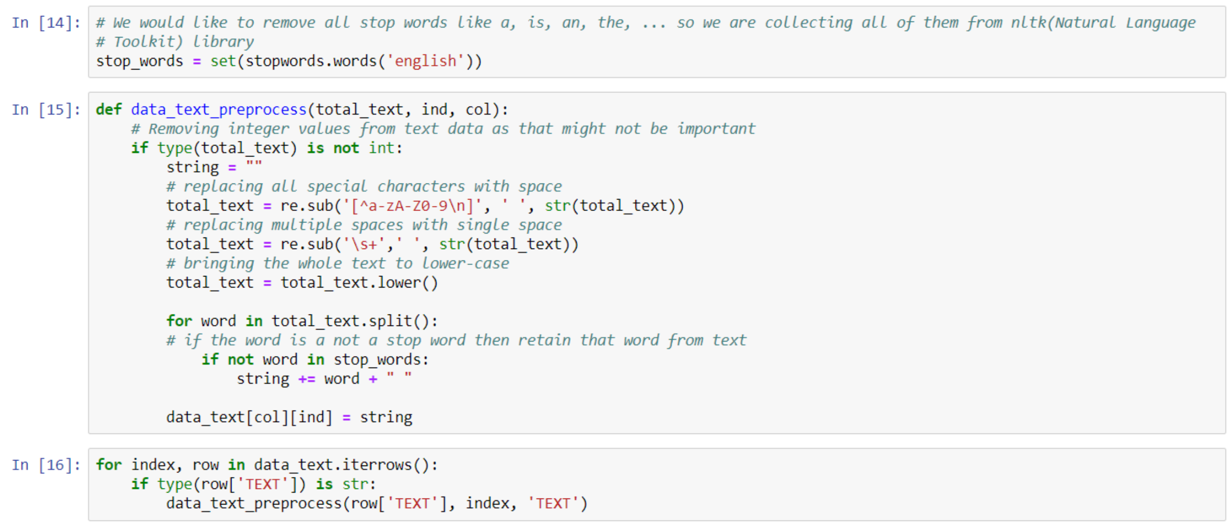
While building a model for a medical problem errors need to be minimized and to decrease the error component and answer the prediction in probability terms as required in the problem description more evidence will be needed to reduce ambiguity and get the most probable class. While predicting the Class for the patient, we can also attach the reason to the Class result explaining the reasons for our prediction. Hence, interpretability will be an evident component of this prediction.

It was decided to use some of the highly interpretable algorithms such as Naive Bayes, Logistic Regression and Linear Support Vector Machine to achieve this. It was also tried to get good predictions from less interpretable models such as Random Forest and K-Nearest Neighbor. It was realized during problem interpretation that the latency can be compromised to some extent for this prediction without increasing the error rate.

4.2 DATA PREPROCESSING

To achieve the models, the Text had to be pre-processed using Natural Language Toolkit (NLTK) so that it can be fed to the Machine Learning algorithm. The Text component actually contains the research papers relevant to every class that was predicated in that case manually. The Text, therefore, has a lot of numbers and stop-words. Since it is a research paper there also might be inappropriate spaces that need to be dealt with. These unnecessary portions were removed using the NLTK library which is a platform used for Python programs that work with human language data for applying in Statistical Natural Language Processing (NLP). It contains Text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

We processed the Text by removing numbers, inappropriate spaces and converting it into the lower case to avoid errors using Regex.



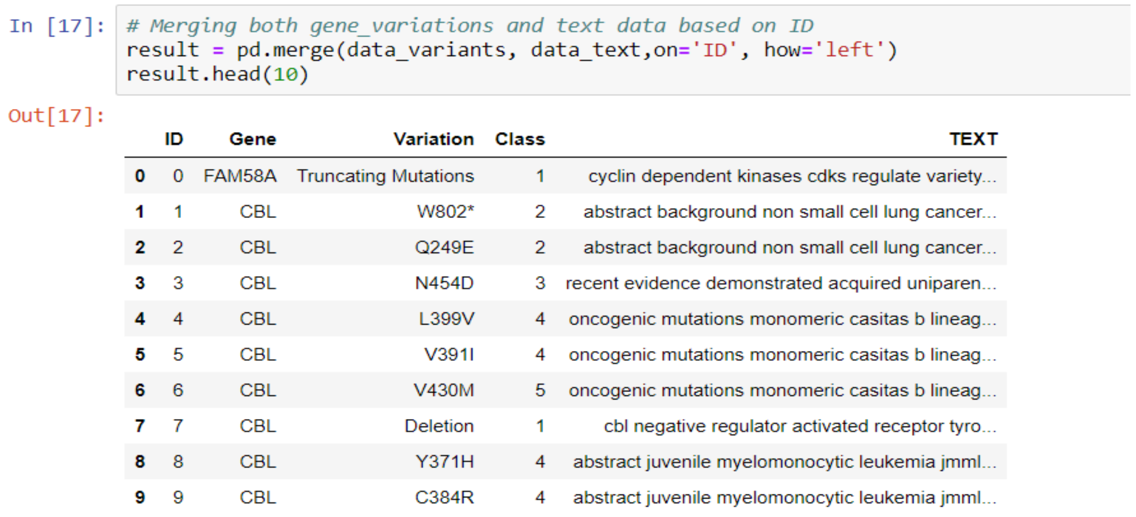
```
In [14]: # We would like to remove all stop words like a, is, an, the, ... so we are collecting all of them from nltk(Natural Language
         # Toolkit) library
         stop_words = set(stopwords.words('english'))
```

```
In [15]: def data_text_preprocess(total_text, ind, col):
             # Removing integer values from text data as that might not be important
             if type(total_text) is not int:
                 string = ""
                 # replacing all special characters with space
                 total_text = re.sub('[^a-zA-Z0-9\n]', ' ', str(total_text))
                 # replacing multiple spaces with single space
                 total_text = re.sub('\s+',' ', str(total_text))
                 # bringing the whole text to lower-case
                 total_text = total_text.lower()

                 for word in total_text.split():
                 # if the word is a not a stop word then retain that word from text
                     if not word in stop_words:
                         string += word + " "

                 data_text[col][ind] = string
```

```
In [16]: for index, row in data_text.iterrows():
             if type(row['TEXT']) is str:
                 data_text_preprocess(row['TEXT'], index, 'TEXT')
```

4.3 MERGING THE DATA FILES

Once, the Text is processed we merged the train_variants and train_text data to achieve a result set comprising ID, GENE, VARIATION, CLASS columns.



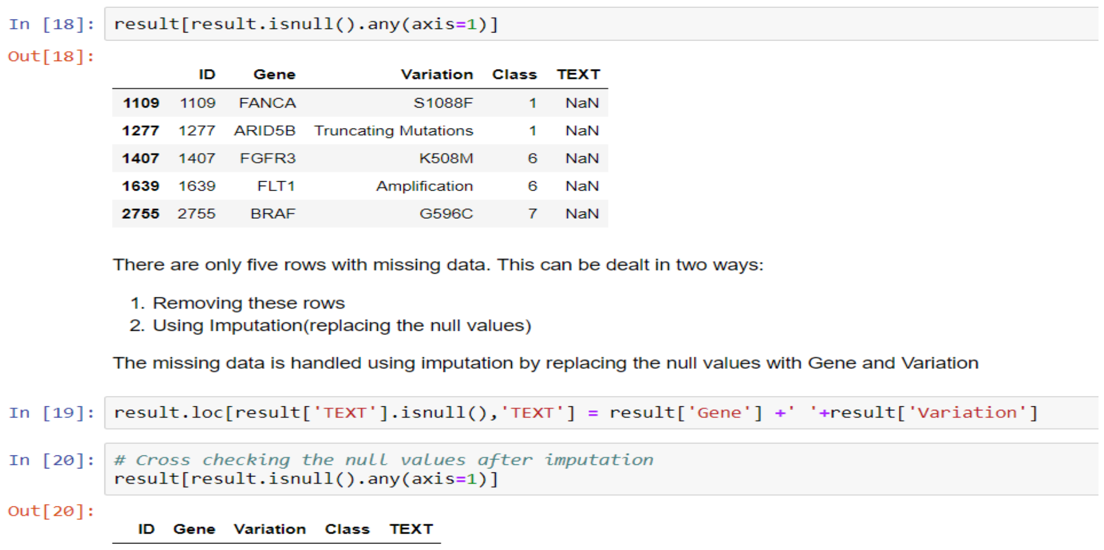
```
In [17]: # Merging both gene_variations and text data based on ID
         result = pd.merge(data_variants, data_text,on='ID', how='left')
         result.head(10)
```

Out[17]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...
5	5	CBL	V391I	4	oncogenic mutations monomeric casitas b lineag...
6	6	CBL	V430M	5	oncogenic mutations monomeric casitas b lineag...
7	7	CBL	Deletion	1	cbl negative regulator activated receptor tyro...
8	8	CBL	Y371H	4	abstract juvenile myelomonocytic leukemia jmml...
9	9	CBL	C384R	4	abstract juvenile myelomonocytic leukemia jmml...

4.4 CLEANING THE DATA AND HANDLING MISSING VALUES

The result set was analyzed to determine any missing values so that they could be handled and won't cause disruptions in the final predictions. There were only five missing values which could be handled in two ways:
1. Removing the rows
2. Using Imputation (Replacing the null values)
It was decided to handle the missing data using Imputation by replacing the null values with the concatenation of GENE and VARIATION column. The data were cross-checked for null values after imputation.



```
In [18]: result[result.isnull().any(axis=1)]
Out[18]:
```

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

There are only five rows with missing data. This can be dealt in two ways:

1. Removing these rows
2. Using Imputation(replacing the null values)

The missing data is handled using imputation by replacing the null values with Gene and Variation

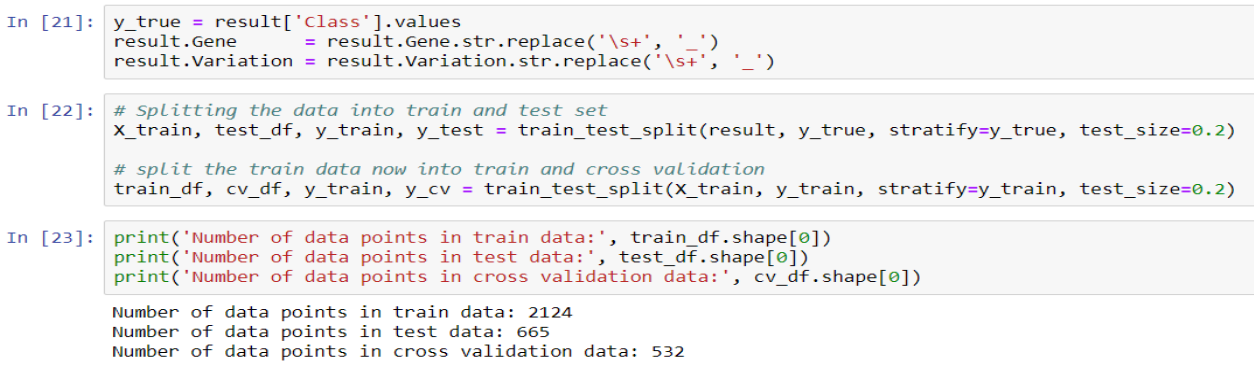
```
In [19]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [20]: # Cross checking the null values after imputation
         result[result.isnull().any(axis=1)]
Out[20]:
```

ID	Gene	Variation	Class	TEXT

4.5 CREATING TRAIN, TEST AND CROSS-VALIDATION SET

Before the data was split, all the spaces in the Gene and Variation column were replaced by _. The data was split into training, testing, and validation because when hyperparameter tuning is done, we try to improve accuracy with respect to Test set which can sometimes lead to bad models. The result data set was, therefore, first divided into an 80% and 20% split of Train and Test and then the 80% of Train was further divided into 80% and 20% for the final Train and Validation set.



```
In [21]: y_true = result['Class'].values
         result.Gene      = result.Gene.str.replace('\s+', '_')
         result.Variation = result.Variation.str.replace('\s+', '_')
```

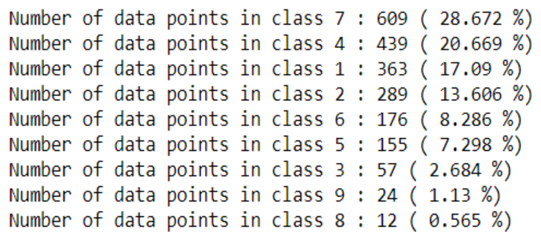
```
In [22]: # Splitting the data into train and test set
         X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)

         # split the train data now into train and cross validation
         train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

```
In [23]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0])

         Number of data points in train data: 2124
         Number of data points in test data: 665
         Number of data points in cross validation data: 532
```

4.6 DATA DISTRIBUTION IN TRAIN, TEST AND CROSS-VALIDATION SET

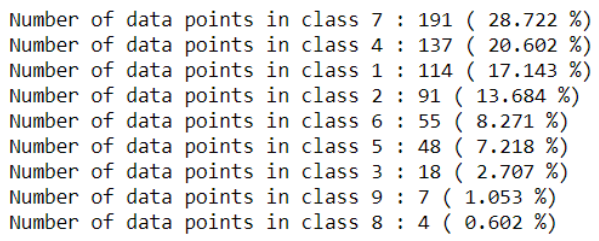
It is important that the distribution of data is similar in all three sets for good predictions. It was verified that the distribution of all these sets was almost similar.

Train_class_distribution



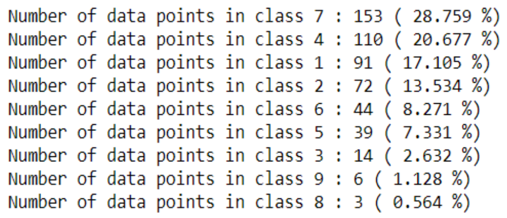
```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
```

Test_class_distribution



```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
```

CV_class_distribution



```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

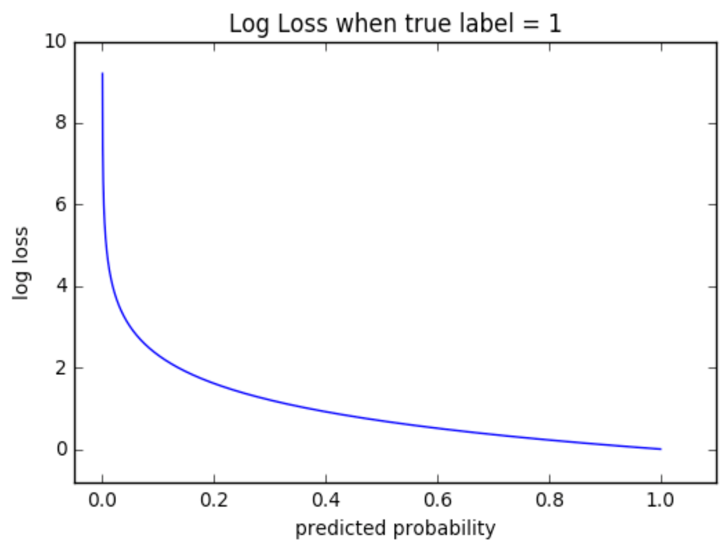
As the Multi-Class log-loss is being evaluated, the main accuracy measures would be log-loss and Confusion Matrix and to minimize error rate, log-loss should be minimal.

Logarithmic loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of our machine learning models is to minimize this value. A perfect model would have a log loss of 0. Log loss increases as the predicted probability diverge from the actual label. So, predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high log loss.

Log Loss vs Accuracy

- Accuracy is the count of predictions where your predicted value equals the actual value. Accuracy is not always a good indicator because of its yes or no nature.
- Log Loss takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view of the performance of our model.

Visualization of Log Loss
The graph below shows the range of possible log loss values given a true observation (isDog = 1). As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predictions that are confident and wrong!



Multi-class Classification

In multi-class classification (M>2), we take the sum of log loss values for each class prediction in the observation.

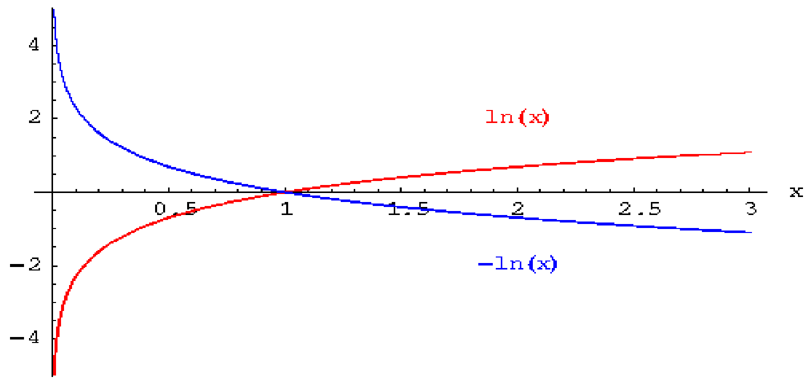
$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

$\sum$ is shorthand for summation or in our case the sum of all log loss values across classes

$c = 1$ is the starting point in the summation (i.e. the first class)

Why the Negative Sign?

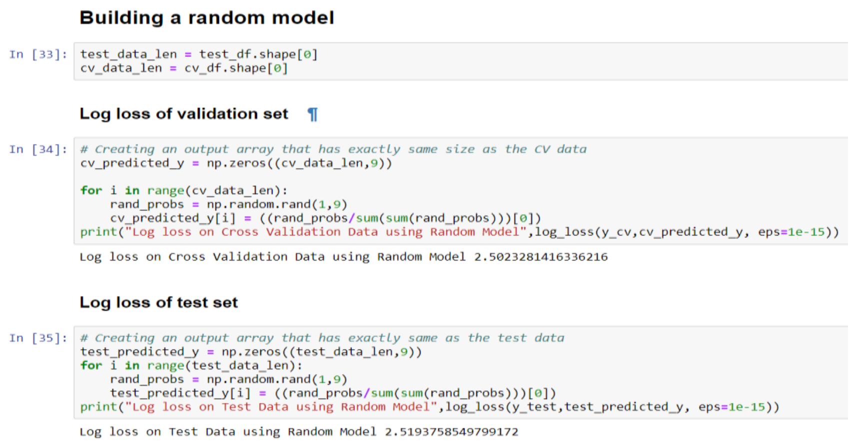
Log Loss uses the negative log to provide an easy metric for comparison. It takes this approach because of the positive log of numbers < 1 returns negative values, which is confusing to work with when comparing the performance of two models.



Log-loss can be a good prediction judge as it has a lower value when a good prediction is achieved and has a higher value when the prediction is ambiguous. It also penalizes when there is ambiguity or low probability for something. The log-loss value ranges from zero to infinity with zero being the best log-loss value and generating the perfect model which is impossible.

4.7 IMPLEMENTATION STRATEGY

As log-loss is being considered as the main evaluation parameter a Random Model was generated to compare the log-loss against. Any successful model would return a lower log-loss than this Random Model.
To evaluate the log-loss, of the cross-validation data for the random Model, an array of zeros was generated having the same length as the cross-validation data. The random probability was generated for all nine Class values in every row and stored in this array with respect to length. A numpy function argmax was used to get the index where the probability of getting a particular class was high. Hence, the log-loss for the cross-validation was evaluated. This value was around 2.50. Similarly, the log-loss of the Test data was evaluated and was around 2.51



Building a random model

```
In [33]: test_data_len = test_df.shape[0]
         cv_data_len = cv_df.shape[0]
```

Log loss of validation set ¶

```
In [34]: # Creating an output array that has exactly same size as the CV data
         cv_predicted_y = np.zeros((cv_data_len,9))

         for i in range(cv_data_len):
             rand_probs = np.random.rand(1,9)
             cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
         print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))

         Log loss on Cross Validation Data using Random Model 2.5023281416336216
```

Log loss of test set

```
In [35]: # Creating an output array that has exactly same as the test data
         test_predicted_y = np.zeros((test_data_len,9))
         for i in range(test_data_len):
             rand_probs = np.random.rand(1,9)
             test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
         print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

         Log loss on Test Data using Random Model 2.5193758549799172
```

Finally, an array of predicted Class values was generated, and the Confusion, Precision, and Recall Matrix were generated.

4.8 DATA EVALUATION CONCEPTS

Independent Columns: Gene, Variation, Text
Dependent Columns: Class [values from 0-9]

Next step is to check, how the gene is impactful on the class column. And if it is, categorical information of columns has to be converted to appropriate format.

Categorical data are variables that contain label values rather than numeric values.

Problem with Categorical Data:
Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric. In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations

on the algorithms themselves. This means that categorical data must be converted to a numerical form, so a machine learning algorithm can process it.

Methods to convert:

One-hot encoding

Response encoding (Mean Imputation)

One-hot and Response Encoding are a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). Using one-hot encoding, the number of columns is increased because all the unique values of the independent variable are joined to the original table as columns and their presence is determined by one whereas absence is determined by zero. Using Response Encoding (mean imputation) generates lesser columns as only the unique or mean values of the dependent variable are joined to the original table as columns.

Laplace Smoothing

Laplace/Additive Smoothing, which is a technique for smoothing categorical data. A small-sample correction, or pseudo-count, will be incorporated in every probability estimate. Consequently, no probability will be zero. The alpha value is taken as 1 in this project. Given an observation from a multinomial distribution with N trials, a "smoothed" version of the data gives the estimator.

Calibrated Classifier are probabilistic classifiers for which the output of the predict_proba method can be directly interpreted as a confidence level. For instance, a well calibrated (binary) classifier should classify the samples such that among the samples to which it gave a predict_proba value close to 0.8, approximately 80% actually belong to the positive class. This project uses a sigmoid function to ensure everything is returned in probability format

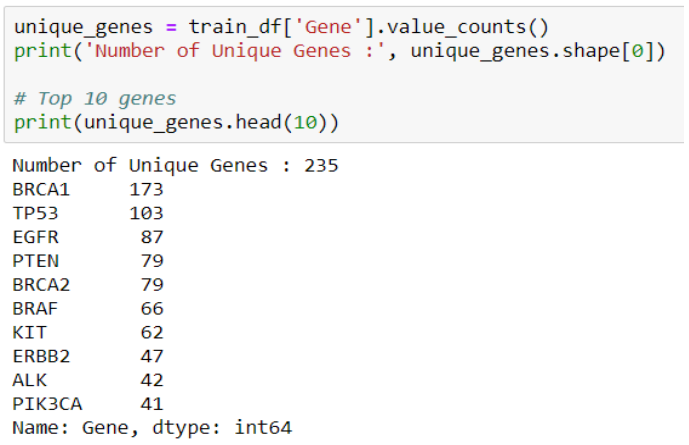
Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression which is being used in this project

4.9 EVALUATING THE COLUMNS

The columns were evaluated to check whether they are impactful for predicting the Class column which is the dependent variable.

Gene Column

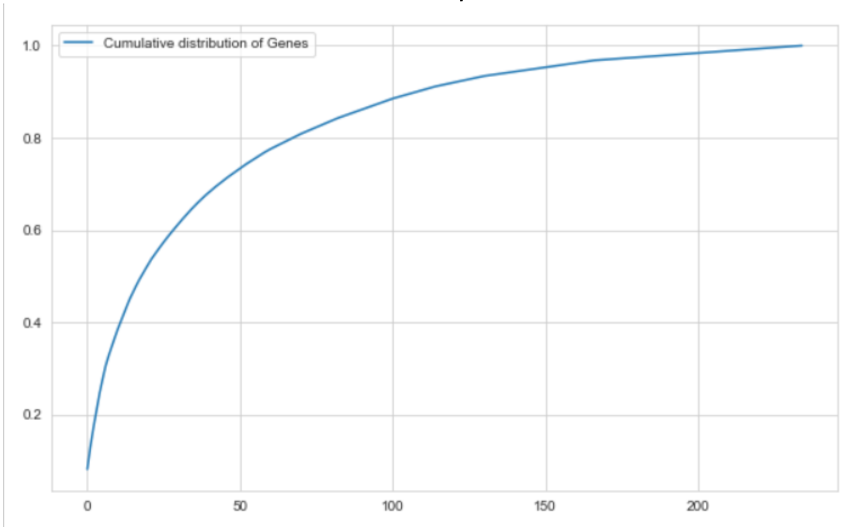
From the Cumulative graph of Gene column, it was observed that the first 50 unique genes, contributes almost 75% of the total values. The Laplace Smoothing was performed on the Gene column and it was evaluated that the train, cross-validation, and test log-loss were approximately 1.04, 1.20, 1.20 for alpha = 0.0001. Also, the overlap between train, test, and train, cross-validation was found to be 95.78% and 96.05% respectively. It was therefore observed that Gene column is good prediction variable as it has very low log-loss value compared to Random Model and high stability as the overlap is high.



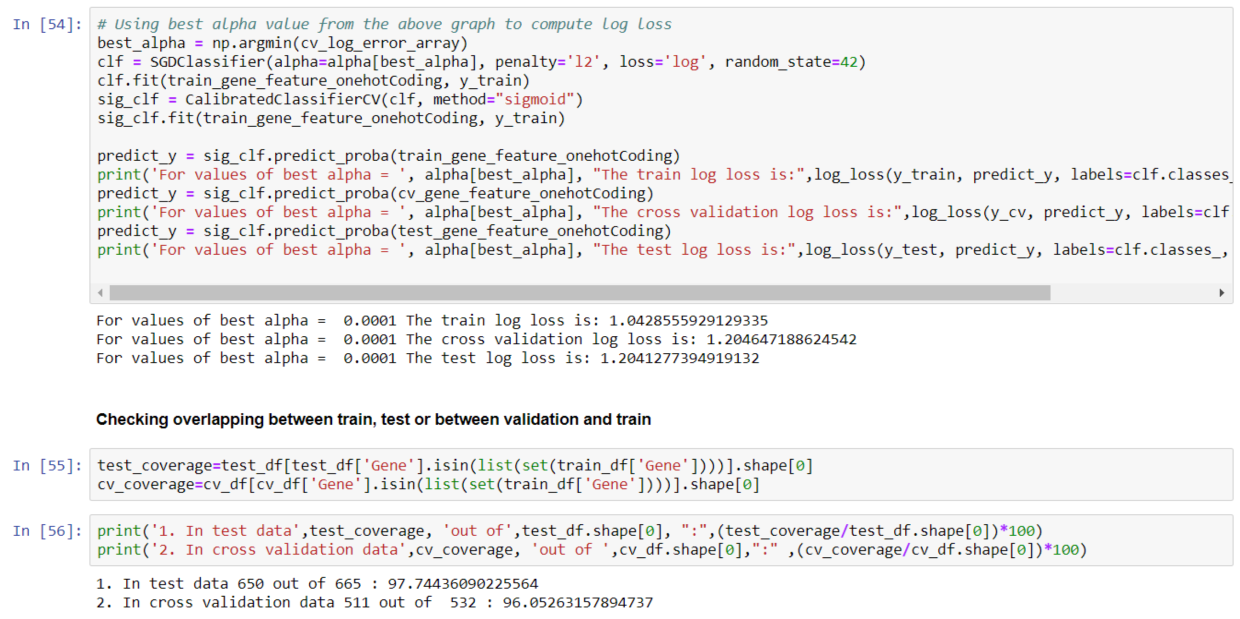
```python
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])

# Top 10 genes
print(unique_genes.head(10))
```

```
Number of Unique Genes : 235
BRCA1     173
TP53      103
EGFR       87
PTEN       79
BRCA2      79
BRAF       66
KIT        62
ERBB2      47
ALK        42
PIK3CA     41
Name: Gene, dtype: int64
```

Number of Unique Genes



Cumulative Distribution of Genes



```
In [54]: # Using best alpha value from the above graph to compute log loss
         best_alpha = np.argmin(cv_log_error_array)
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
         clf.fit(train_gene_feature_onehotCoding, y_train)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_gene_feature_onehotCoding, y_train)

         predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
         predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
         predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,

         For values of best alpha =  0.0001 The train log loss is: 1.0428555929129335
         For values of best alpha =  0.0001 The cross validation log loss is: 1.204647188624542
         For values of best alpha =  0.0001 The test log loss is: 1.2041277394919132
```

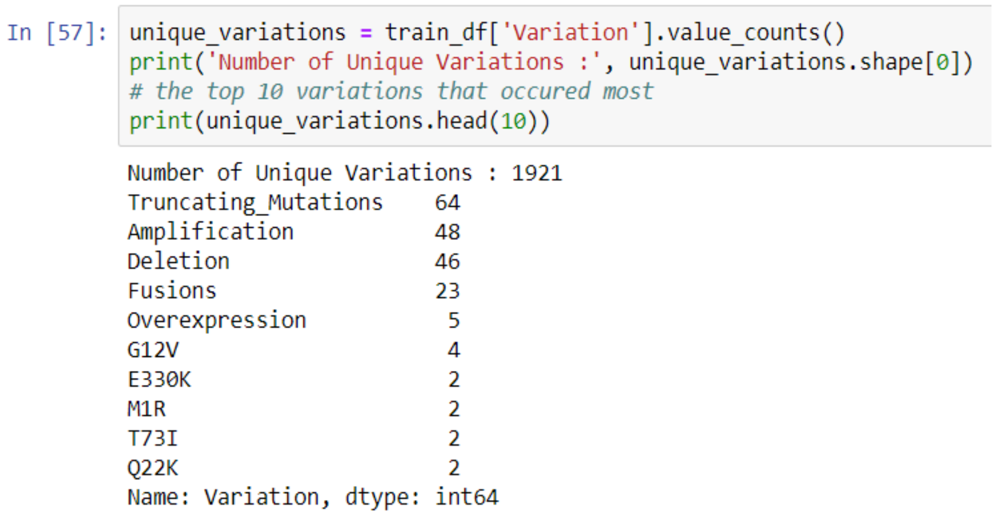
Checking overlapping between train, test or between validation and train

```
In [55]: test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
         cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

In [56]: print('1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*100)
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)

         1. In test data 650 out of 665 : 97.74436090225564
         2. In cross validation data 511 out of  532 : 96.05263157894737
```

Evaluation of log loss and overlap for gene column

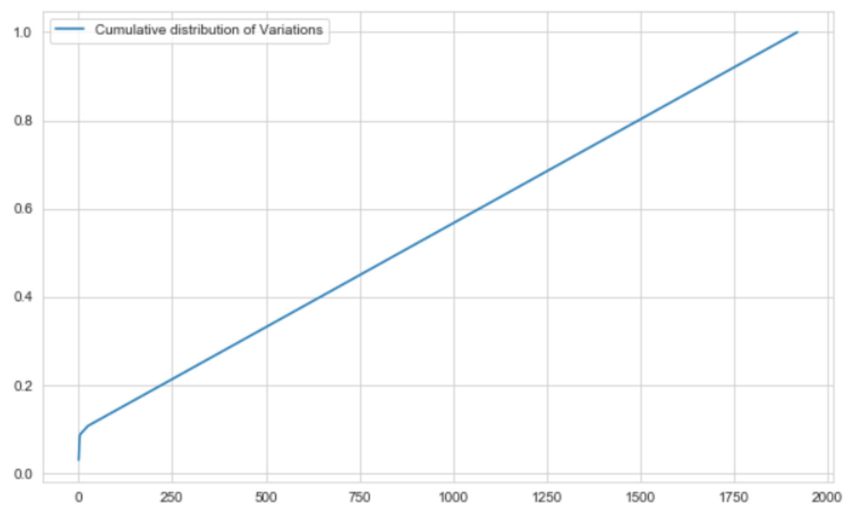
Variation Column

From the Cumulative graph of Variation column, it was observed that first 1500 unique variations, contributes almost 80% of the total values. The Laplace Smoothing was performed on the Variation column and it was evaluated that the train, cross-validation, and test log-loss were approximately 1.05, 1.69, 1.74 for alpha = 0.01. Also, the overlap between train, test, and train, cross-validation was found to be 7.51% and 12.40% respectively. It was therefore observed that Variation column is good prediction variable as it has very low log-loss value compared to Random Model even though the stability is low due to low overlap.

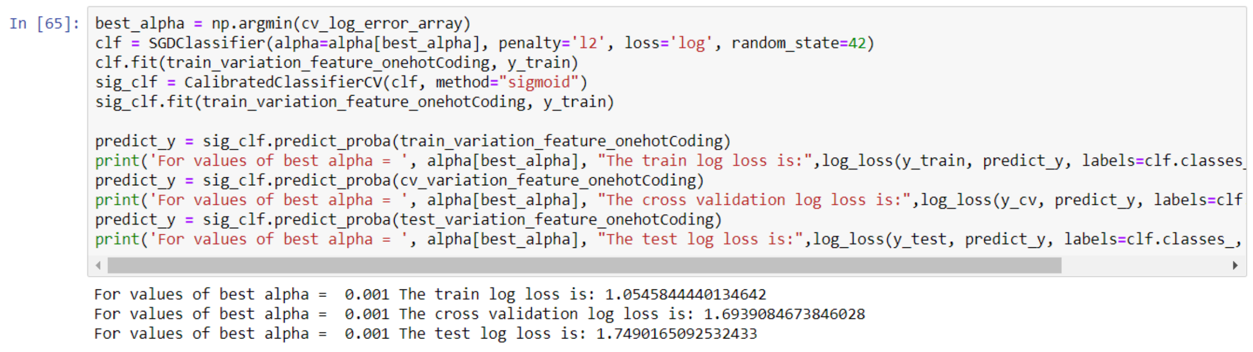


```
In [57]: unique_variations = train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         # the top 10 variations that occured most
         print(unique_variations.head(10))

         Number of Unique Variations : 1921
         Truncating_Mutations    64
         Amplification           48
         Deletion                46
         Fusions                 23
         Overexpression           5
         G12V                     4
         E330K                    2
         M1R                      2
         T73I                     2
         Q22K                     2
         Name: Variation, dtype: int64
```

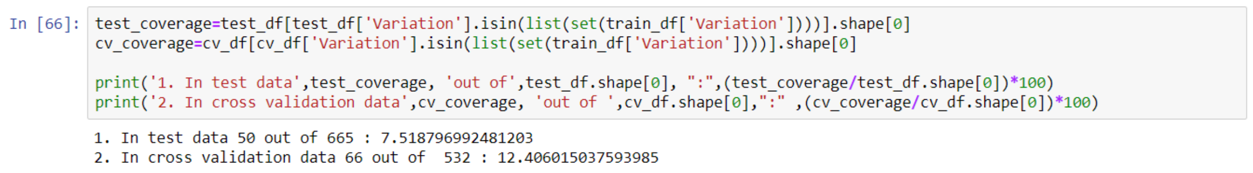
Number of Unique Variations



Cumulative Distribution of Variations



```
In [65]: best_alpha = np.argmin(cv_log_error_array)
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
         clf.fit(train_variation_feature_onehotCoding, y_train)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_variation_feature_onehotCoding, y_train)

         predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
         predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
         predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,

         For values of best alpha =  0.001 The train log loss is: 1.0545844440134642
         For values of best alpha =  0.001 The cross validation log loss is: 1.6939084673846028
         For values of best alpha =  0.001 The test log loss is: 1.7490165092532433
```

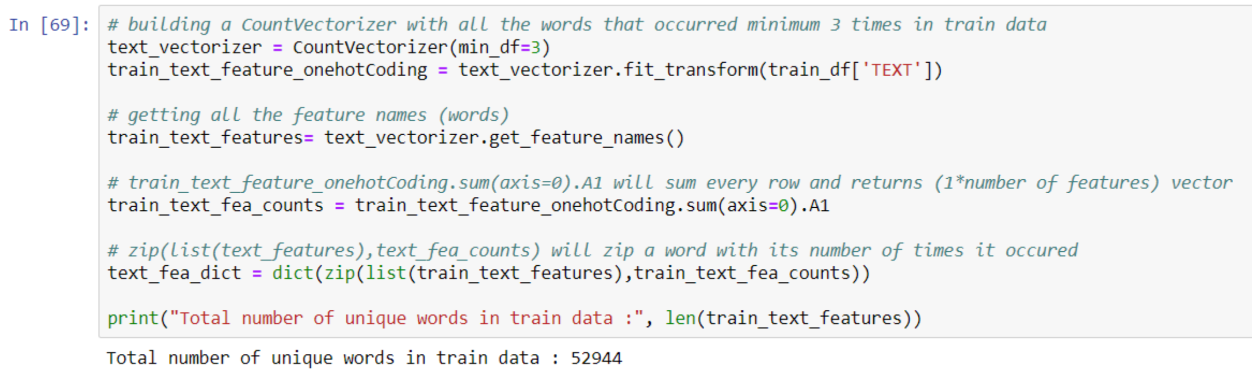


```
In [66]: test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
         cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]

         print('1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*100)
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)

         1. In test data 50 out of 665 : 7.518796992481203
         2. In cross validation data 66 out of  532 : 12.406015037593985
```

Evaluation of log loss and overlap for variations column

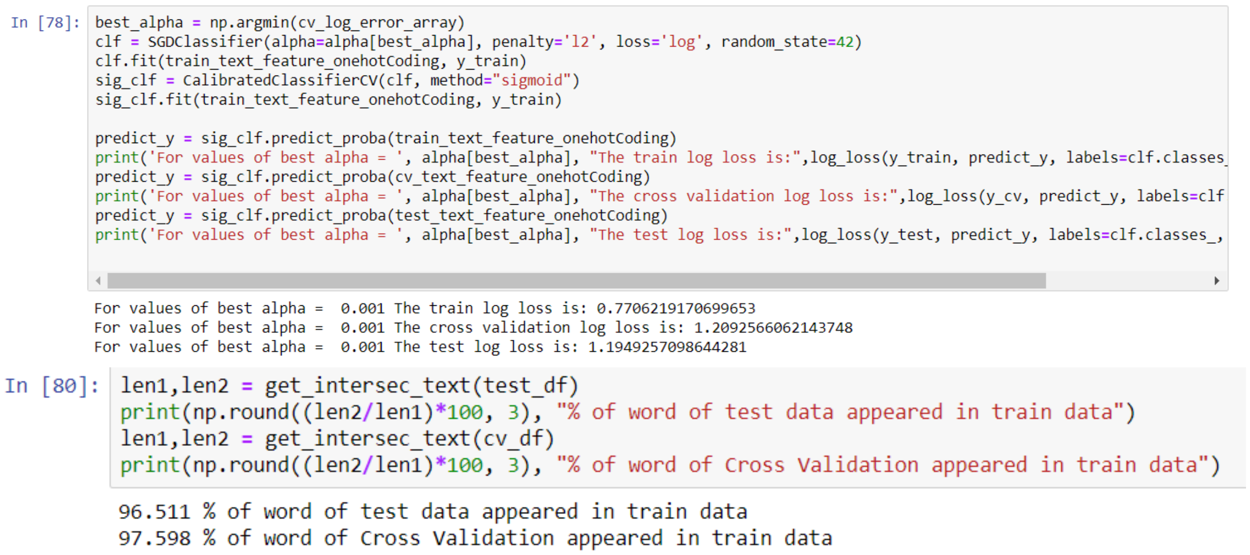
Text Column

For the Text column, the unique words were stored in a dictionary and the corresponding count value is incremented every time the word is encountered. A count vectorizer was built with all the words that occurred a minimum three times in the data. The total unique words were found to be 52944. Using Response Encoding values in each row were converted such that the sum was one. Every feature is normalized and is trained using the same Vectorizer as the train data. It was evaluated that the train, cross-validation, and test log-loss were approximately 0.77, 1.20, 1.19 for alpha = 0.001. Also, the overlap between train, test, and train, cross-validation was found to be 96.51% and 97.59% respectively. It was therefore observed that the Text column is good prediction variable as it has very low log-loss value compared to Random Model and high stability as the overlap is high.



```
In [69]: # building a CountVectorizer with all the words that occurred minimum 3 times in train data
         text_vectorizer = CountVectorizer(min_df=3)
         train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])

         # getting all the feature names (words)
         train_text_features= text_vectorizer.get_feature_names()

         # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
         train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

         # zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
         text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))

         print("Total number of unique words in train data :", len(train_text_features))

         Total number of unique words in train data : 52944
```

Number of Unique words in Text

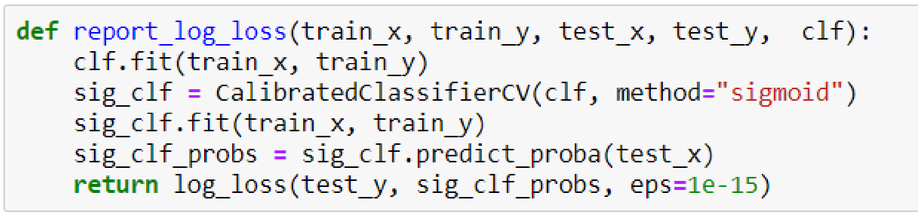


```
In [78]: best_alpha = np.argmin(cv_log_error_array)
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
         clf.fit(train_text_feature_onehotCoding, y_train)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_text_feature_onehotCoding, y_train)

         predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
         predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
         predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,

         For values of best alpha =  0.001 The train log loss is: 0.7706219170699653
         For values of best alpha =  0.001 The cross validation log loss is: 1.2092566062143748
         For values of best alpha =  0.001 The test log loss is: 1.1949257098644281

In [80]: len1,len2 = get_intersec_text(test_df)
         print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
         len1,len2 = get_intersec_text(cv_df)
         print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")

         96.511 % of word of test data appeared in train data
         97.598 % of word of Cross Validation appeared in train data
```

Evaluation of log loss and overlap for variations column

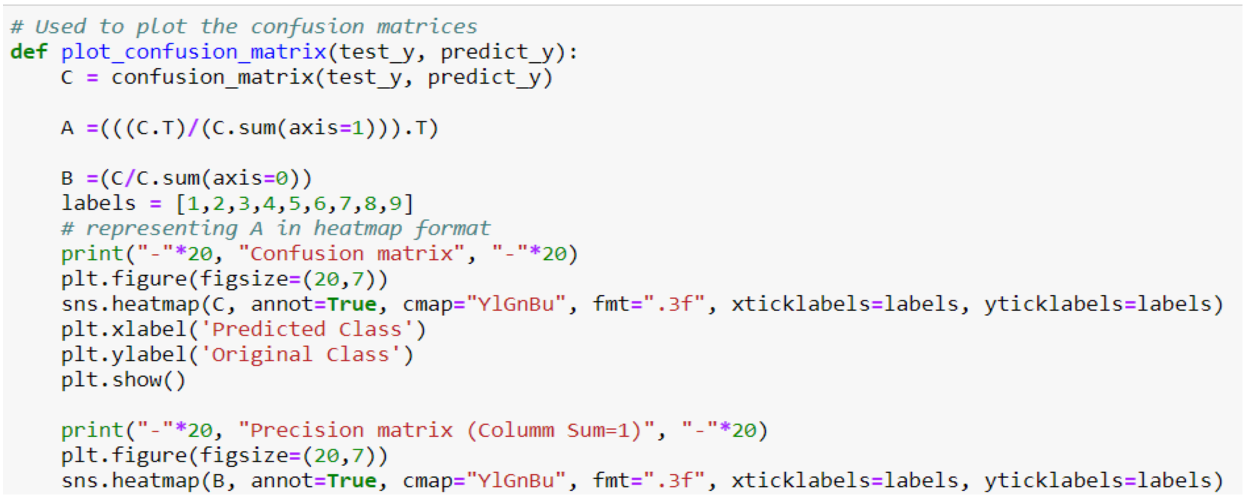
4.10 DATA PREPARATION

For the successful generation of the log-loss confusion matrix, precision matrix, recall matrix and determining the misclassified points functions were prepared in advance. Another function was prepared for the Naïve-Bayes Model which would check the feature is present in test point text or not.

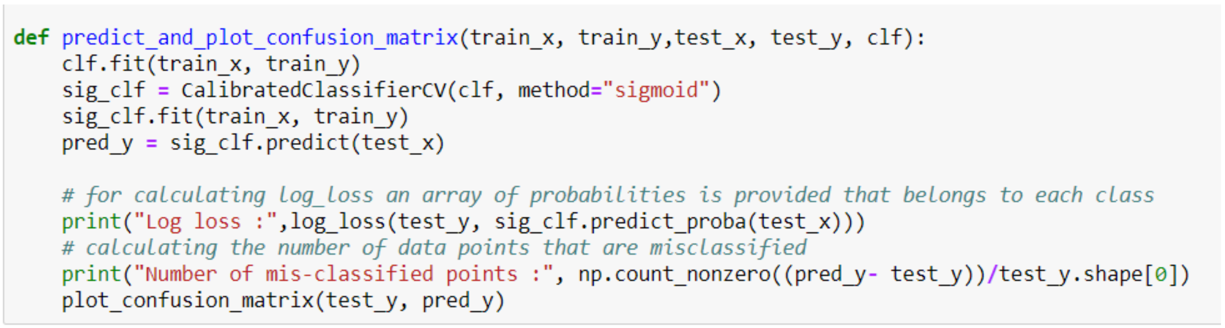


```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

Log loss function

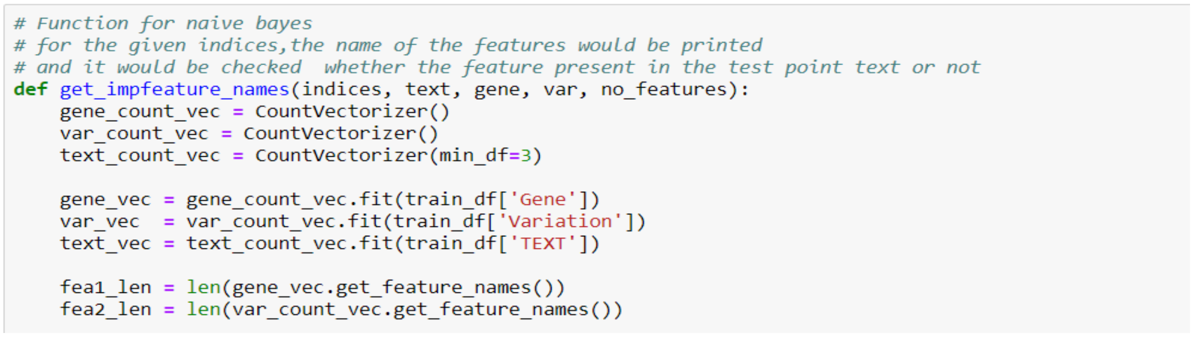


```
# Used to plot the confusion matrices
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
```



```
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss an array of probabilities is provided that belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

Some part of confusion, precision and recall matrix functions



```
# Function for naive bayes
# for the given indices,the name of the features would be printed
# and it would be checked  whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())
```

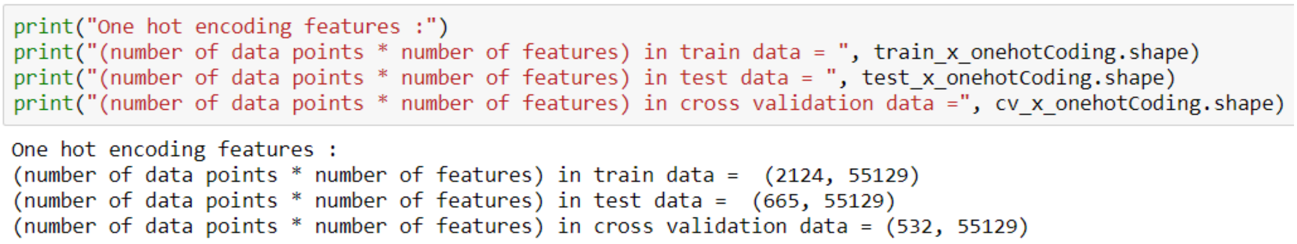
Some part of feature function to be used for Naïve Bayes

4.11 COMBINING THE FEATURES

All the three variables as generated using One-hot Encoding and Response Encoding were combined together respectively using numpy function hstack.

According to one-hot encoding the features are:

1. Train (2124, 55129)
2. Test (665, 55129)
3. Cross-Validation (532, 55129)



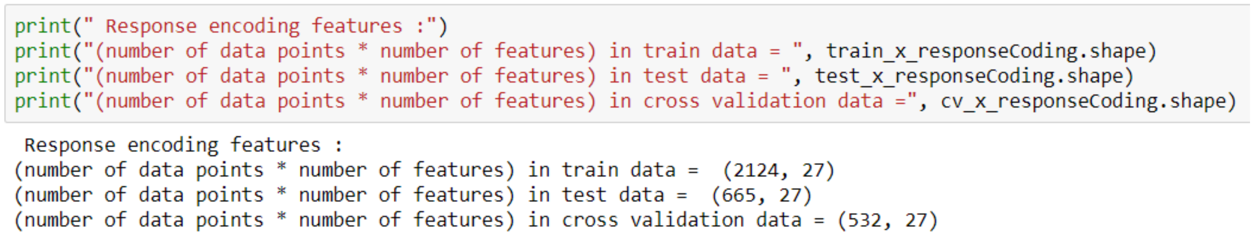
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55129)
(number of data points * number of features) in test data =  (665, 55129)
(number of data points * number of features) in cross validation data = (532, 55129)
```

One hot encoding features

According to Response encoding the features are:

1. Train (2124, 27)
2. Test (665, 27)
3. Cross-Validation (532, 27)



```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

Response encoding features

Hence, we can deduce that Response Encoding would limit the number of columns in the best possible way.

4.12 BUILDING MACHINE LEARNING MODELS

4.12.1 Naïve Bayes

In machine learning, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts.

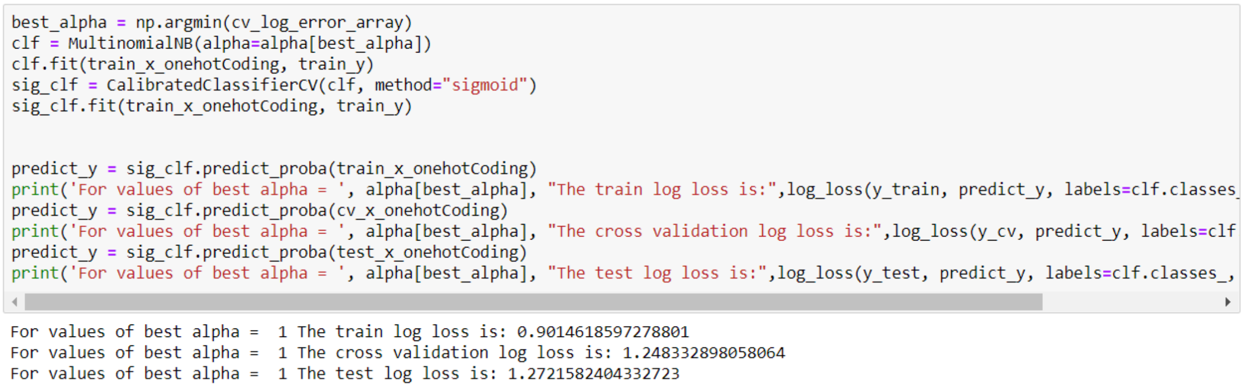
$$\begin{aligned}\log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k)\prod_{i=1}^{n} {p_{ki}}^{x_i}\right)\\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki}\\ &= b + \mathbf{w}_k^\top \mathbf{x}\end{aligned}$$

where $b = \log p(C_k)$ and $w_{ki} = \log p_{ki}$.

The formula for Multinomial Naïve Bayes

The Multinomial Naïve Bayes classifier was used in this project as we have discrete data in the form of 9 classes and also because the Naïve Bayes model is highly interpretable.

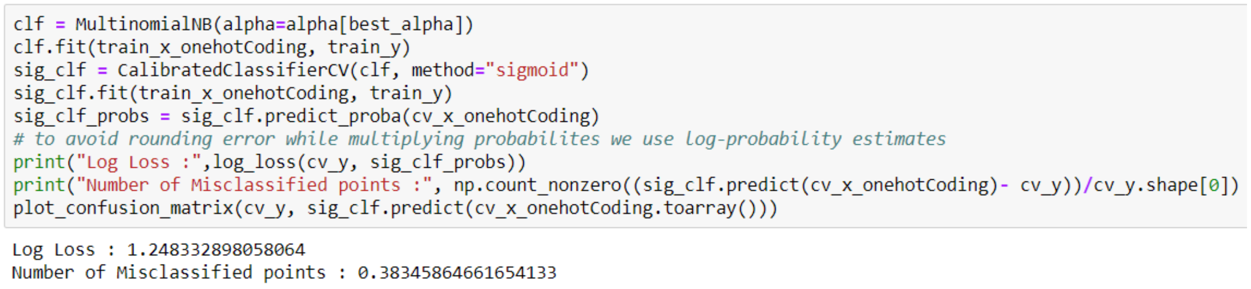
The best alpha value for Naïve Bayes Model is 1 which produces the train, cross-validation and test log-loss values as 0.90, 1.24, 1.27 respectively. When the Naïve Bayes Model was tested on the testing data using the best alpha the log-loss for the model was generated as 1.24 and misclassified points were 38.34% which means that the model predicts 61.66% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,
```

```
For values of best alpha =  1 The train log loss is: 0.9014618597278801
For values of best alpha =  1 The cross validation log loss is: 1.248332898058064
For values of best alpha =  1 The test log loss is: 1.2721582404332723
```

Log loss for train, test, and cross-validation set using Multinomial Naïve Bayes

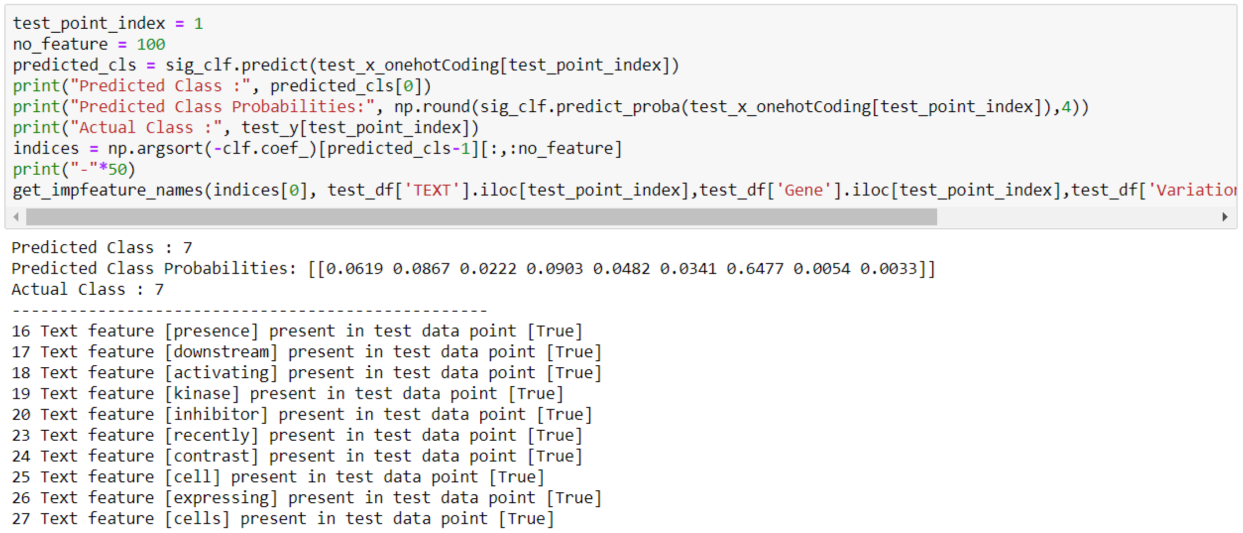


```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of Misclassified points :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.248332898058064
Number of Misclassified points : 0.38345864661654133
```

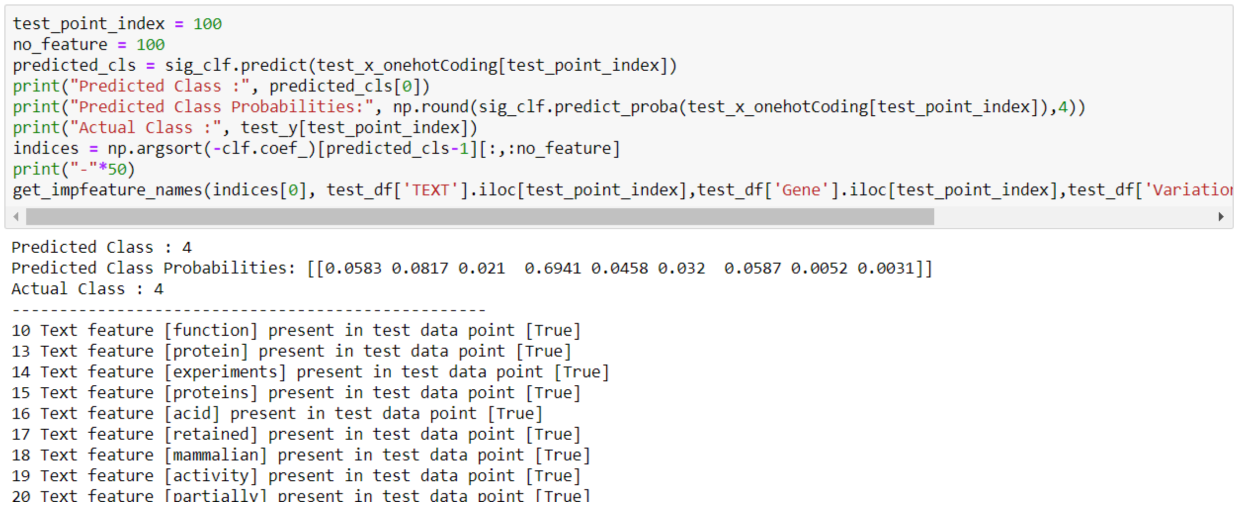
Log loss for the model and misclassified points using Multinomial Naïve Bayes

We also tested, the interpretability of the model using two random data points.



```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variatio
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0619 0.0867 0.0222 0.0903 0.0482 0.0341 0.6477 0.0054 0.0033]]
Actual Class : 7
--------------------------------------------------
16 Text feature [presence] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
18 Text feature [activating] present in test data point [True]
19 Text feature [kinase] present in test data point [True]
20 Text feature [inhibitor] present in test data point [True]
23 Text feature [recently] present in test data point [True]
24 Text feature [contrast] present in test data point [True]
25 Text feature [cell] present in test data point [True]
26 Text feature [expressing] present in test data point [True]
27 Text feature [cells] present in test data point [True]
```



```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variatio
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0583 0.0817 0.021  0.6941 0.0458 0.032  0.0587 0.0052 0.0031]]
Actual Class : 4
--------------------------------------------------
10 Text feature [function] present in test data point [True]
13 Text feature [protein] present in test data point [True]
14 Text feature [experiments] present in test data point [True]
15 Text feature [proteins] present in test data point [True]
16 Text feature [acid] present in test data point [True]
17 Text feature [retained] present in test data point [True]
18 Text feature [mammalian] present in test data point [True]
19 Text feature [activity] present in test data point [True]
20 Text feature [partially] present in test data point [True]
```

4.12.2 K-Nearest Neighbours

In pattern recognition, the K-Nearest Neighbors algorithm (KNN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether KNN is used for classification or regression:

- In KNN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In KNN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

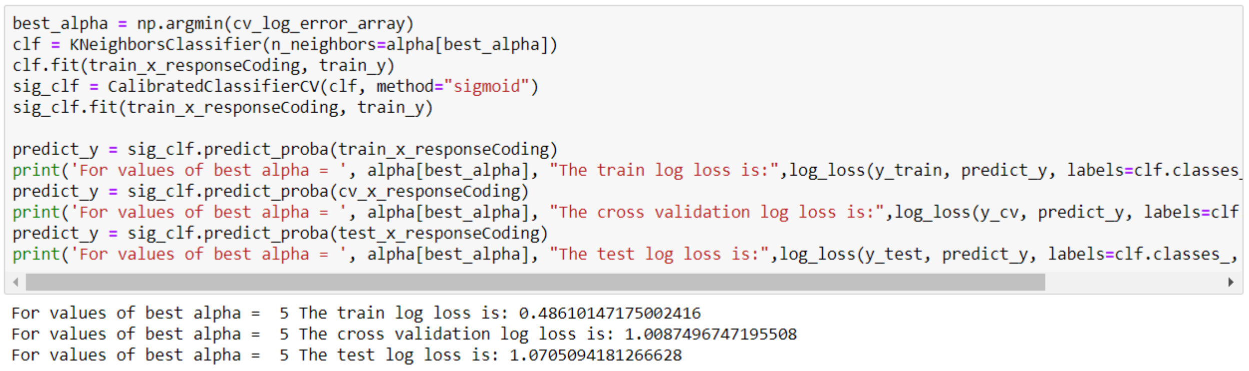
KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The KNN algorithm is among the simplest of all machine learning algorithms. A peculiarity of the KNN algorithm is that it is sensitive to the local structure of the data.

$$y = \frac{1}{K}\sum_{i=1}^{K} y_i$$

Formula for KNN

Here yi is the ith case of the examples sample and y is the prediction (outcome) of the query point.

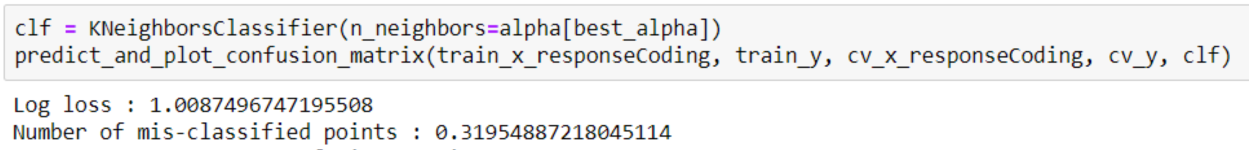
The KNN algorithm was used because although it is not very interpretable it was believed to produce good results for the problem. The KNN model is using the alpha values as n_neighbors which are used to predict the point based on the number of the nearest neighbors. The best alpha/ n_neighbors value for KNN Model is 5 which produces the train, cross-validation, and test log-loss values as 0.48, 1, 1.07 respectively. When the KNN Model was tested on the testing data using the best alpha the log-loss for the model was generated as 1.008 and misclassified points were 31.95% which means that the model predicts 68.05% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,
```

```
For values of best alpha =  5 The train log loss is: 0.48610147175002416
For values of best alpha =  5 The cross validation log loss is: 1.0087496747195508
For values of best alpha =  5 The test log loss is: 1.0705094181266628
```

Log loss for train, test, and cross-validation set using K-Nearest Neighbors

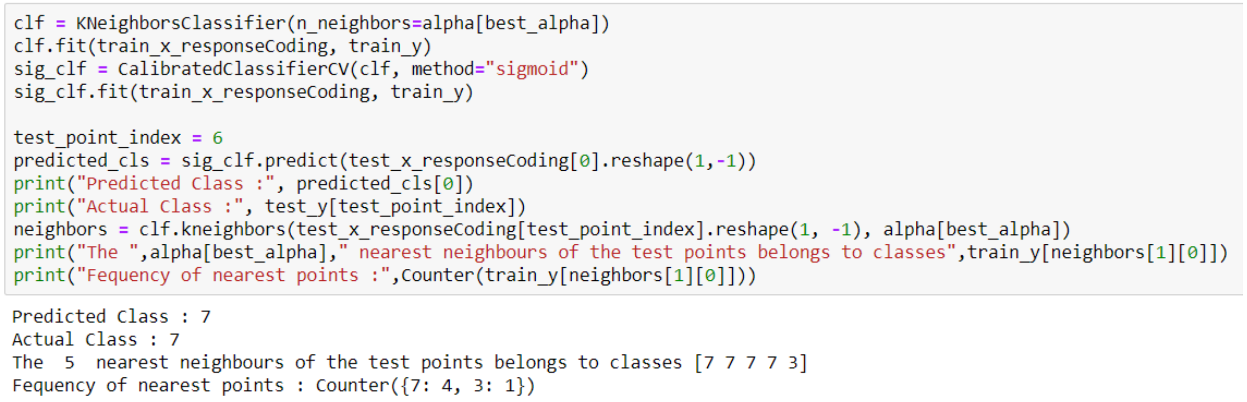


```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.0087496747195508
Number of mis-classified points : 0.31954887218045114
```

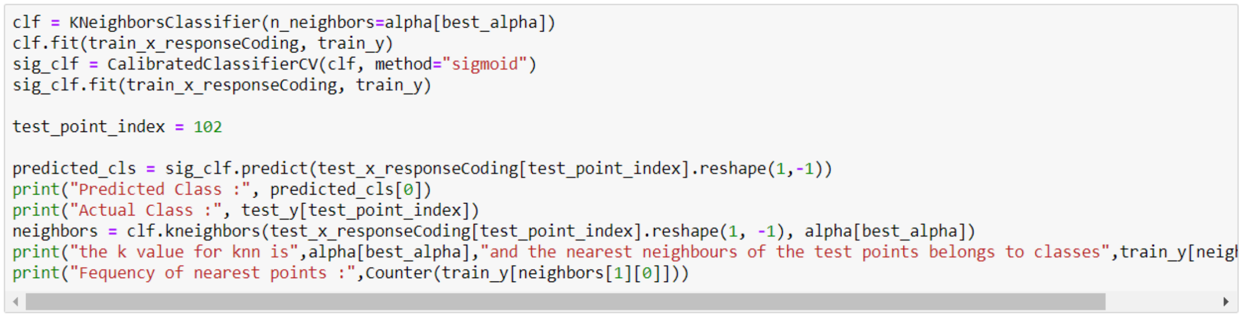
Log loss for the model and misclassified points using K-Nearest Neighbors

We also tested, the model using two random data points.



```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 6
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
The  5  nearest neighbours of the test points belongs to classes [7 7 7 7 3]
Fequency of nearest points : Counter({7: 4, 3: 1})
```



```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 102

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test points belongs to classes",train_y[neig
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 3
Actual Class : 3
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [3 3 3 4 5]
Fequency of nearest points : Counter({3: 3, 4: 1, 5: 1})
```

4.12.3 Logistic Regression

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). The logistic regression model predicts P(Y=1) as a function of X.

$$P = \frac{e^{a+bX}}{1+e^{a+bX}}$$

The formula for computing logistic regression probability

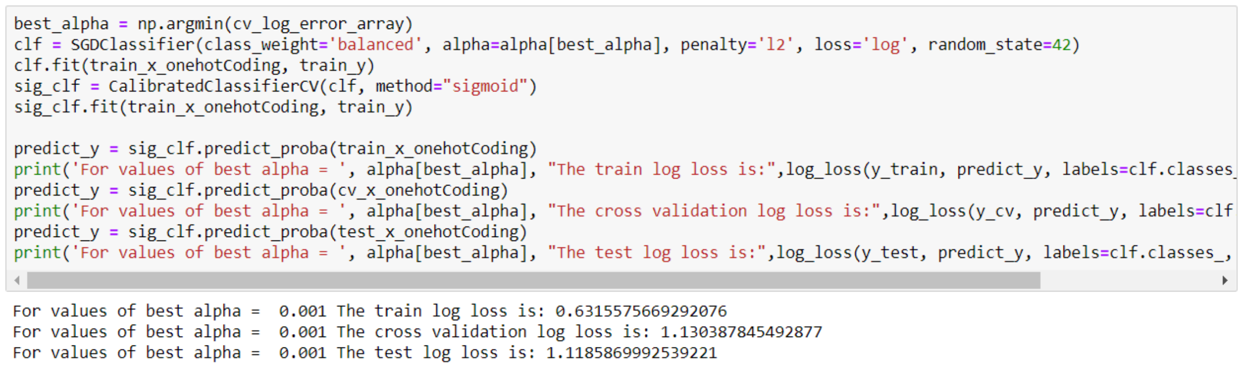
Here P is the probability of a 1 (the proportion of 1s, the mean of Y), e is the base of the natural logarithm (about 2.718) and a and b are the parameters of the model. The value of a yields P when X is zero and b adjusts how quickly the probability changes with changing X a single unit (we can have standardized and unstandardized b weights in logistic regression, just as in ordinary linear regression). Because the relation between X and P is nonlinear, b does not have a straightforward interpretation in this model as it does in ordinary linear regression.

Logistic regression fits an S-shaped logistic function. The curve goes from 0-1. It is as simple as classifying the waste like this.

The Logistic Regression model was used because it is highly interpretable and can be used for Multi-class classification easily.

For the LR model we would use two methods:
1. Over-sampling which means classes with fewer data would be balanced out with respect to other classes.
2. Without balancing which means classes won't be balanced out.

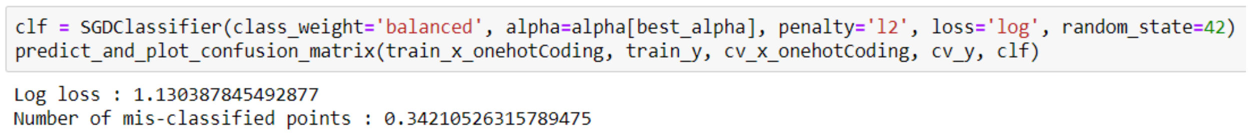
In the first case, the best alpha value for this model is 0.001 which produces the train, cross-validation and test log-loss values as 0.63, 1.13, 1.11 respectively. When the LR Model was tested on the testing data using the best alpha the log-loss for the model was generated as 1.13 and misclassified points were 34.2% which means that the model predicts 65.83% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,
```

```
For values of best alpha =  0.001 The train log loss is: 0.6315575669292076
For values of best alpha =  0.001 The cross validation log loss is: 1.130387845492877
For values of best alpha =  0.001 The test log loss is: 1.1185869992539221
```

Log loss for train, test, and cross-validation set using Logistic Regression with balancing

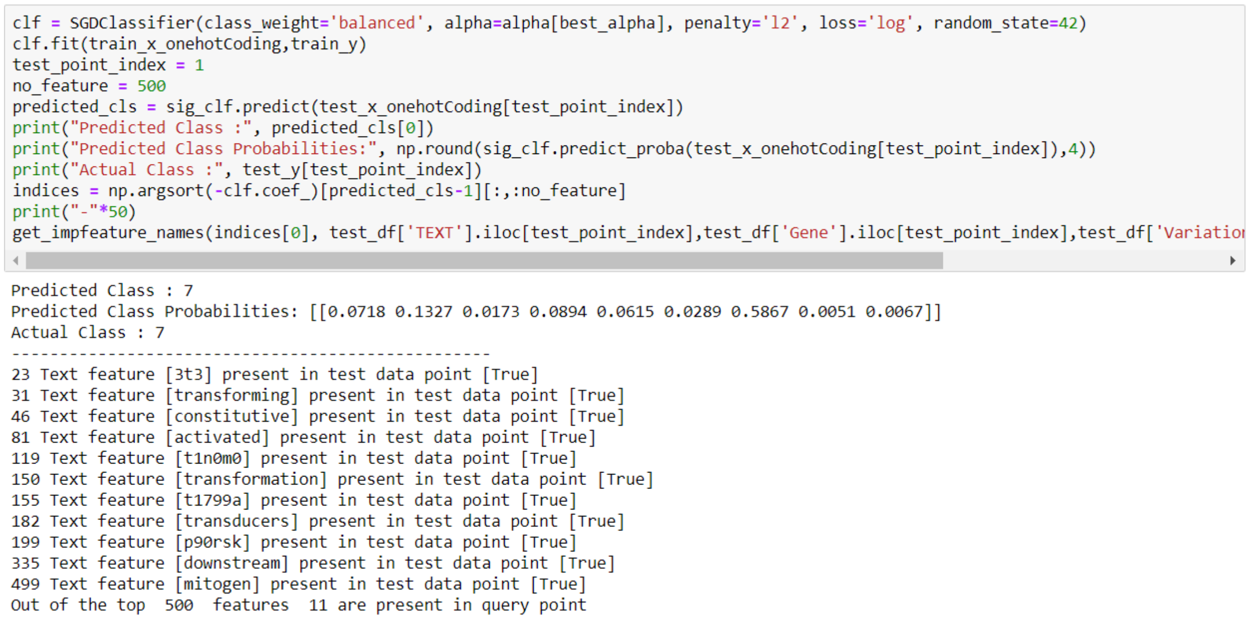


```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.130387845492877
Number of mis-classified points : 0.34210526315789475
```

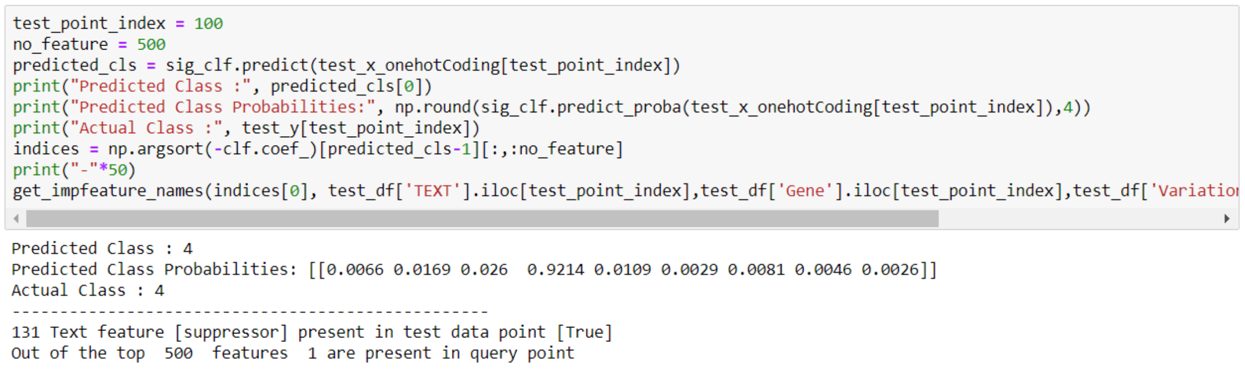
Log loss for the model and misclassified points using Logistic Regression with balancing

We also tested, the model using two random data points.



```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variatio
```

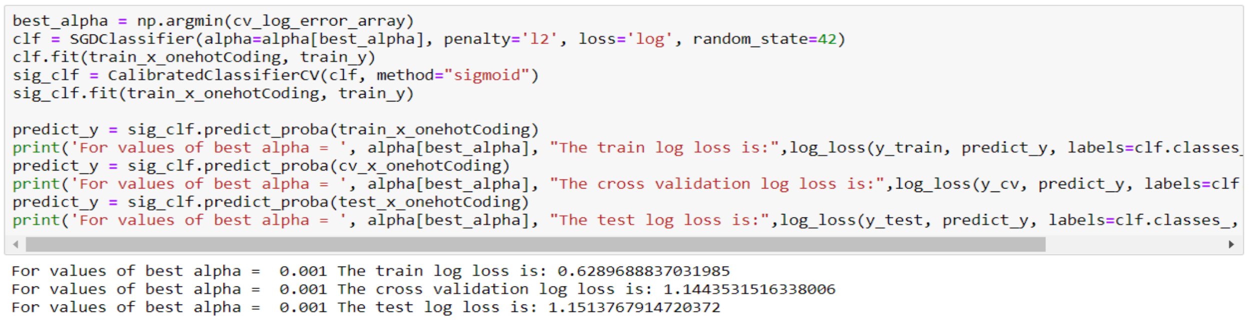
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0718 0.1327 0.0173 0.0894 0.0615 0.0289 0.5867 0.0051 0.0067]]
Actual Class : 7
--------------------------------------------------
23 Text feature [3t3] present in test data point [True]
31 Text feature [transforming] present in test data point [True]
46 Text feature [constitutive] present in test data point [True]
81 Text feature [activated] present in test data point [True]
119 Text feature [t1n0m0] present in test data point [True]
150 Text feature [transformation] present in test data point [True]
155 Text feature [t1799a] present in test data point [True]
182 Text feature [transducers] present in test data point [True]
199 Text feature [p90rsk] present in test data point [True]
335 Text feature [downstream] present in test data point [True]
499 Text feature [mitogen] present in test data point [True]
Out of the top  500  features  11 are present in query point
```



```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variatio
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0066 0.0169 0.026  0.9214 0.0109 0.0029 0.0081 0.0046 0.0026]]
Actual Class : 4
--------------------------------------------------
131 Text feature [suppressor] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

In the second case, the best alpha value for this model is 0.001 which produces the train, cross-validation and test log-loss values as 0.62, 1.14, 1.15 respectively. When the LR Model was tested on the testing data using the best alpha the log-loss for the model was generated as 1.14 and misclassified points were 33.08% which means that the model predicts 66.92% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,
```

```
For values of best alpha =  0.001 The train log loss is: 0.6289688837031985
For values of best alpha =  0.001 The cross validation log loss is: 1.1443531516338006
For values of best alpha =  0.001 The test log loss is: 1.1513767914720372
```

Log loss for train, test, and cross-validation set using Logistic Regression without balancing

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.1443531516338006
Number of mis-classified points : 0.3308270676691729
```

Log loss for the model and misclassified points using Logistic Regression without balancing

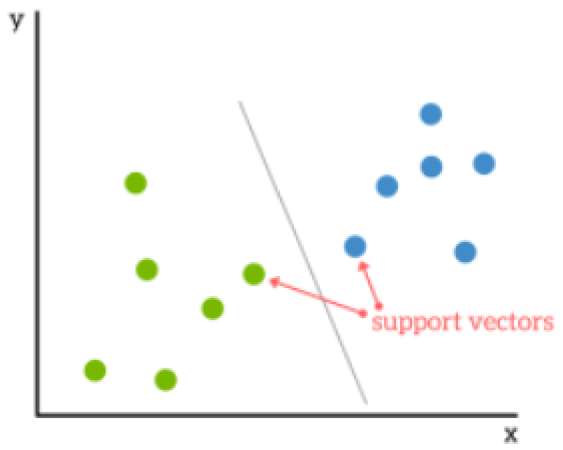
We also tested, the model using one random data points.

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variatio
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.074  0.1312 0.015  0.0946 0.0597 0.0295 0.5871 0.0052 0.0037]]
Actual Class : 7
--------------------------------------------------
64 Text feature [3t3] present in test data point [True]
85 Text feature [transforming] present in test data point [True]
133 Text feature [t1799a] present in test data point [True]
151 Text feature [constitutive] present in test data point [True]
200 Text feature [activated] present in test data point [True]
202 Text feature [transformation] present in test data point [True]
334 Text feature [t1n0m0] present in test data point [True]
386 Text feature [mitogen] present in test data point [True]
404 Text feature [p90rsk] present in test data point [True]
460 Text feature [transducers] present in test data point [True]
488 Text feature [ptcs] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

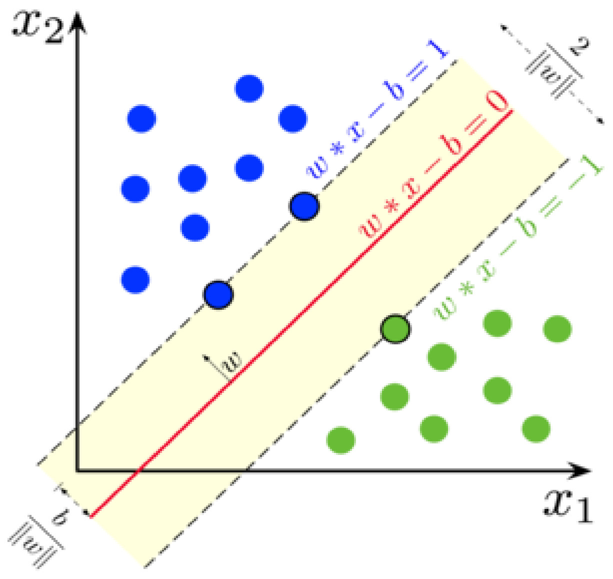
4.12.4 Linear Support Vector Machine

Support Vector Machines are based on the idea of finding a hyperplane that best divides a dataset into two classes, as shown in the image below.



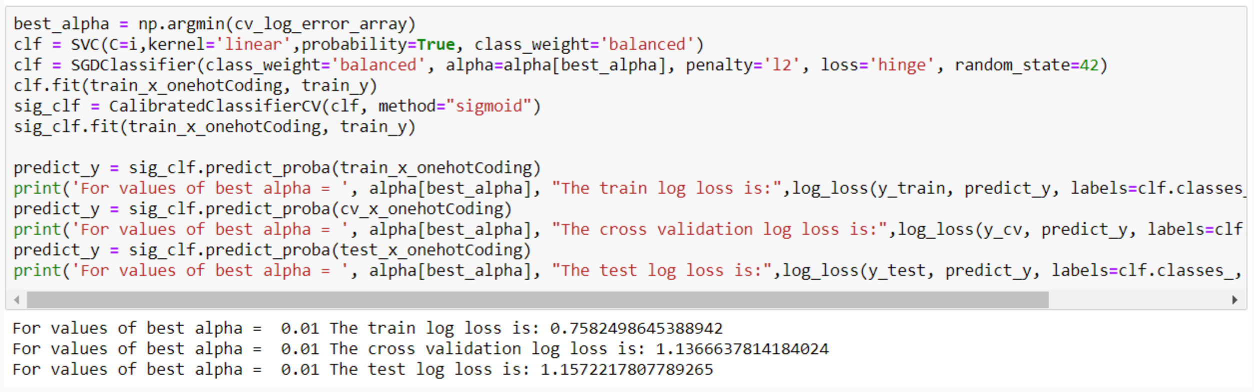
Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set.

As a simple example, for a classification task with only two features, you can think of a hyperplane as a line that linearly separates and classifies a set of data. Intuitively, the further from the hyperplane our data points lie, the more confident we are that they have been correctly classified. We, therefore, want our data points to be as far away from the hyperplane as possible, while still being on the correct side of it. So when new testing data is added, whatever side of the hyperplane it lands will decide the class that we assign to it. The distance between the hyperplane and the nearest data point from either set is known as the margin. The goal is to choose a hyperplane with the greatest possible margin between the hyperplane.



The Linear Support Vector Machine was used because it is a highly interpretable model.

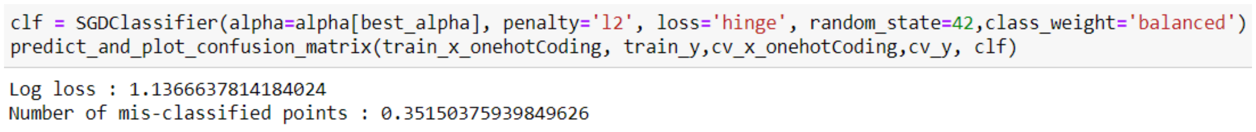
The best C/alpha value for Linear SVM Model is 0.01 which produces the train, cross-validation, and test log-loss values as 0.75, 1.13, 1.15 respectively. When the Linear SVM Model was tested on the testing data using the best C/alpha the log-loss for the model was generated as 1.13 and misclassified points were 35.15% which means that the model predicts 64.85% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_,
```

```
For values of best alpha =  0.01 The train log loss is: 0.7582498645388942
For values of best alpha =  0.01 The cross validation log loss is: 1.1366637814184024
For values of best alpha =  0.01 The test log loss is: 1.1572217807789265
```

Log loss for train, test, and cross-validation set using Linear Support Vector Machine

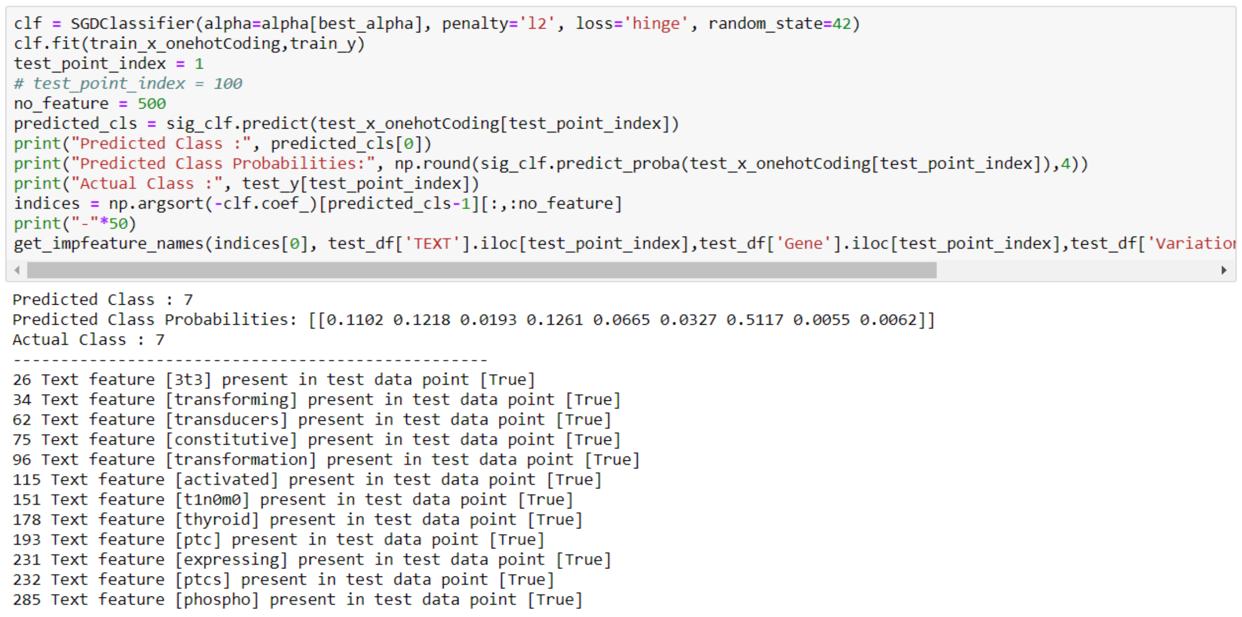


```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1366637814184024
Number of mis-classified points : 0.35150375939849626
```

Log loss for the model and misclassified points using Linear Support Vector Machine

We also tested, the model using one random data point.



```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variatio
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1102 0.1218 0.0193 0.1261 0.0665 0.0327 0.5117 0.0055 0.0062]]
Actual Class : 7
--------------------------------------------------
26 Text feature [3t3] present in test data point [True]
34 Text feature [transforming] present in test data point [True]
62 Text feature [transducers] present in test data point [True]
75 Text feature [constitutive] present in test data point [True]
96 Text feature [transformation] present in test data point [True]
115 Text feature [activated] present in test data point [True]
151 Text feature [t1n0m0] present in test data point [True]
178 Text feature [thyroid] present in test data point [True]
193 Text feature [ptc] present in test data point [True]
231 Text feature [expressing] present in test data point [True]
232 Text feature [ptcs] present in test data point [True]
285 Text feature [phospho] present in test data point [True]
```

4.12.5 Random Forest Classifier

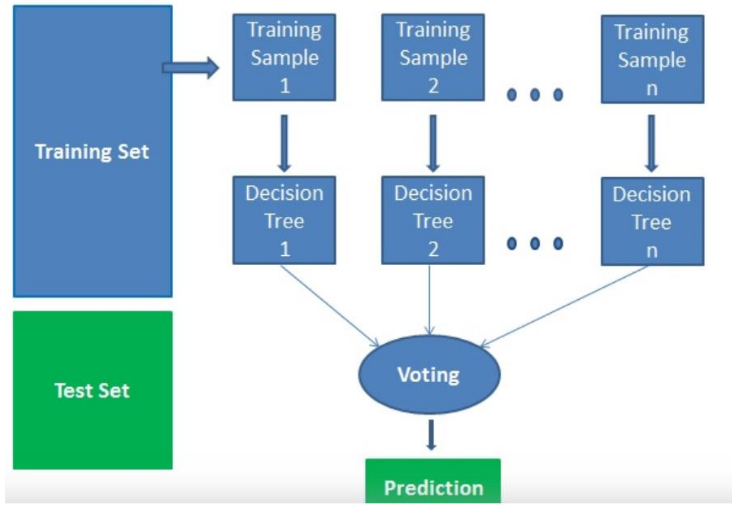
Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use the algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, gets a prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance. Random Forest Classifier is an ensemble algorithm. Ensemble algorithms are those which combines more than one algorithms of a same or different kind for classifying objects. For example, running prediction over Naive Bayes, SVM, and Decision Tree and then taking a vote for final consideration of class for a test object.

The important term Bagging directs to the ensemble, which means that a group of thing viewed as a whole. In order to ensure the ensemble, we need to do the following:

- We should create multiple models
- We should combine their results.

Example: Suppose training set is given as : [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], the random forest may create three decision trees taking input of subset for example,
[X1, X2, X3]
[X1, X2, X4]
[X2, X3, X4]
So finally, it predicts based on the majority of votes from each of the decision trees made.

Alternatively, the random forest can apply weight concept for considering the impact of result from any decision tree. Tree with a high error rate is given low weight value and vis-a-vis. This would increase the decision impact of trees with low error rate.
It technically is an ensemble method (based on the divide-and-conquer approach) of decision trees generated on a randomly split dataset. This collection of decision tree classifiers is also known as the forest. The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute. Each tree depends on an independent random sample. In a classification problem, each tree votes and the most popular class is chosen as the final result. In the case of regression, the average of all the tree outputs is considered as the final result. It is simpler and more powerful compared to the other non-linear classification algorithms.



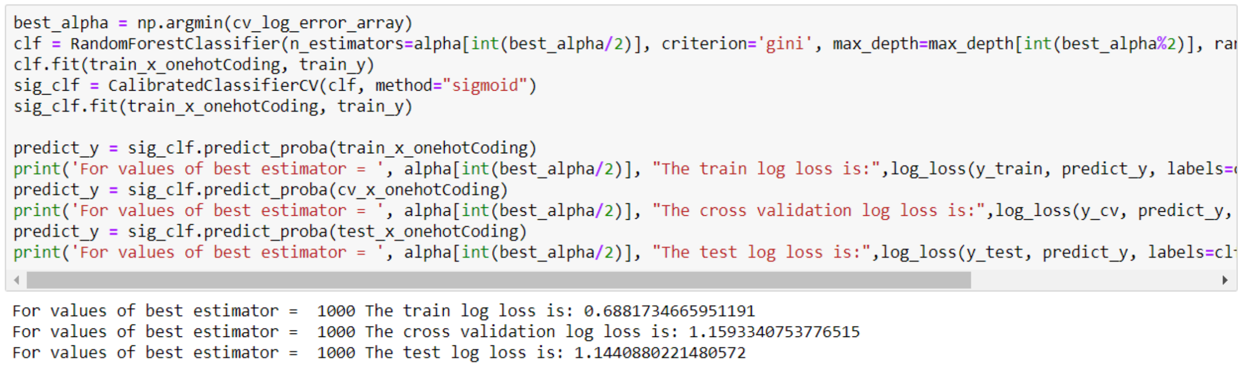
Random Forest Classifier with Maximum voting classifier

Random forests also offer a good feature selection indicator. Scikit-learn provides an extra variable with the model, which shows the relative importance or contribution of each feature in the prediction. It automatically computes the relevance score of each feature in the training phase. Then it scales the relevance down so that the sum of all scores is 1.This score will help you choose the most important features and drop the least important ones for model building.

Random forest uses gene importance or means a decrease in impurity (MDI) to calculate the importance of each feature. Gini importance is also known as the total decrease in node impurity. This is how much the model fit or accuracy decreases when you drop a variable. The larger the decrease, the more significant the variable is. Here, the mean decrease is a significant parameter for variable selection. The Gini index can describe the overall explanatory power of the variables.

The RFC is implemented using both one-hot encoder and Response encoder to get the best possible results as it prevents overfitting and is believed to lower the log loss significantly

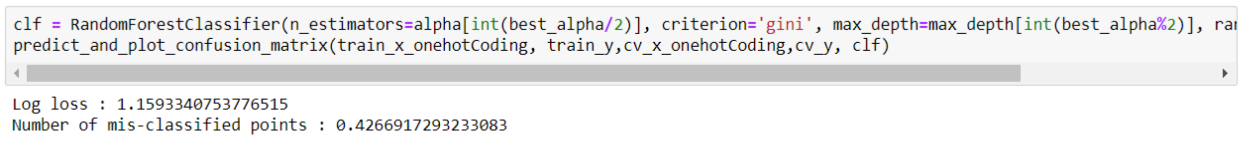
For the first case, the best n_estimators and max_depth value for this model are 1000, 10 respectively which produces the train, cross-validation, and test log-loss values as 0.68, 1.15, 1.14 respectively. When the Linear RFC Model was tested on the testing data using the best n_estimators and max_depth the log-loss for the model was generated as 1.15 and misclassified points were 42.66% which means that the model predicts 57.34% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], ra
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train, predict_y, labels=
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss is:",log_loss(y_cv, predict_y,
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(y_test, predict_y, labels=cl
```

```
For values of best estimator =  1000 The train log loss is: 0.6881734665951191
For values of best estimator =  1000 The cross validation log loss is: 1.1593340753776515
For values of best estimator =  1000 The test log loss is: 1.1440880221480572
```

Log loss for train, test, and cross-validation set using Random Forest Classifier with one hot encoder

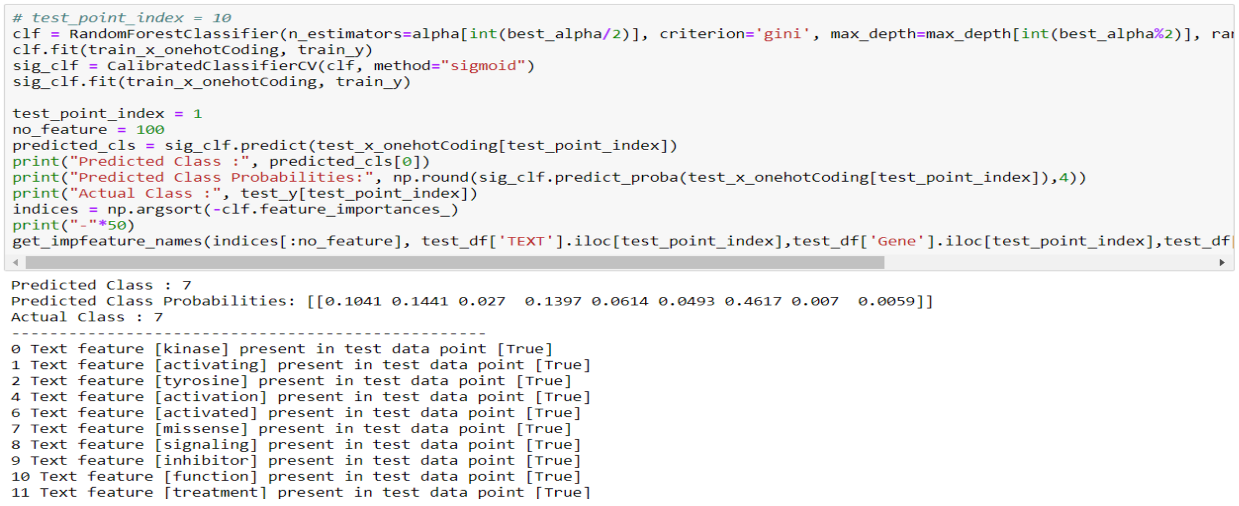


```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], ra
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1593340753776515
Number of mis-classified points : 0.4266917293233083
```

Log loss for the model and misclassified points using Random Forest Classifier with one hot encoder

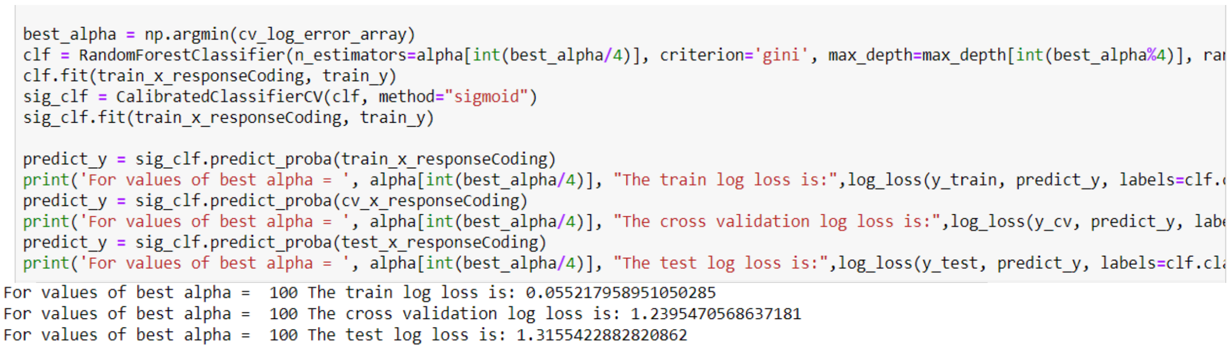
We also tested, the model using one random data points.



```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], ra
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1041 0.1441 0.027  0.1397 0.0614 0.0493 0.4617 0.007  0.0059]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [missense] present in test data point [True]
8 Text feature [signaling] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
10 Text feature [function] present in test data point [True]
11 Text feature [treatment] present in test data point [True]
```

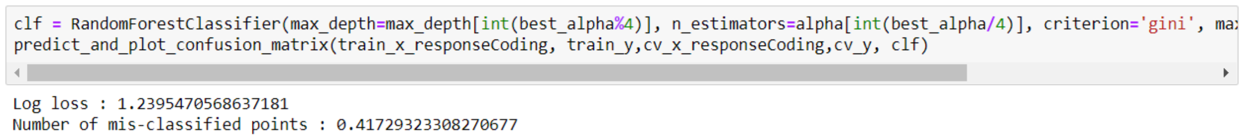
For the second case, the best n_estimators and max_depth value for this model are 100, 5 respectively which produces the train, cross-validation, and test log-loss values as 0.05, 1.23, 1.31 respectively. When the Linear RFC Model was tested on the testing data using the best n_estimators and max_depth the log-loss for the model was generated as 1.23 and misclassified points were 41.72% which means that the model predicts 58.28% points correctly. The confusion, precision and recall matrix support these results.



```
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha%4)], ra
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:",log_loss(y_cv, predict_y, lab
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.cl
```

```
For values of best alpha =  100 The train log loss is: 0.055217958951050285
For values of best alpha =  100 The cross validation log loss is: 1.2395470568637181
For values of best alpha =  100 The test log loss is: 1.3155422882820862
```

Log loss for train, test, and cross-validation set using Random Forest Classifier with response encoder

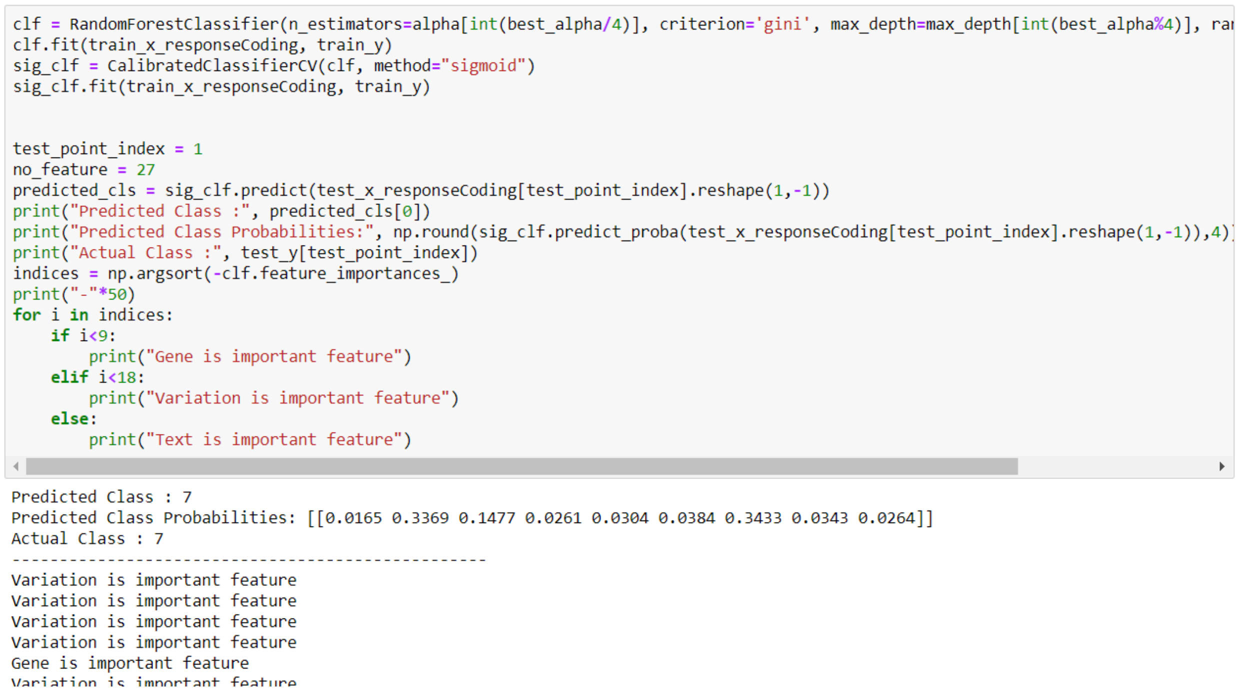


```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], criterion='gini', ma
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.2395470568637181
Number of mis-classified points : 0.41729323308270677
```

Log loss for the model and misclassified points using Random Forest Classifier with response encoder

We also tested, the model using one random data point.

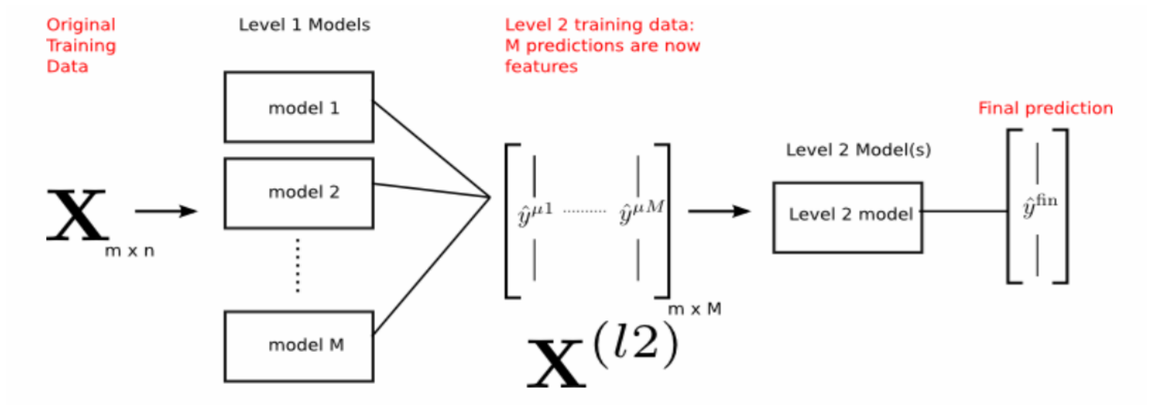


```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha%4)], ra
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4)
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0165 0.3369 0.1477 0.0261 0.0304 0.0384 0.3433 0.0343 0.0264]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
```

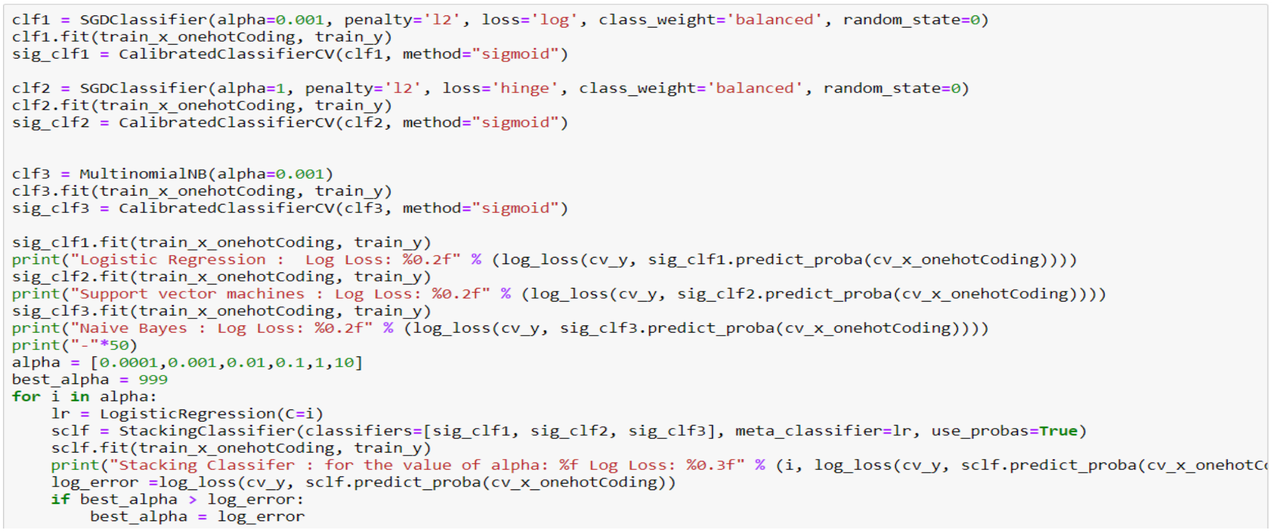
4.12.6 Stacking Model

Stacked Generalization or stacking is an ensemble algorithm where a new model is trained to combine the predictions from two or more models already trained or your dataset. The predictions from the existing models or submodels are combined using a new model, and as such stacking is often referred to as blending, as the predictions from sub-models are blended together. It is typical to use a simple linear method to combine the predictions for submodels such as simple averaging or voting, to a weighted sum using linear regression or logistic regression. Models that have their predictions combined must have skill on the problem, but do not need to be the best possible models. This means that you do not need to tune the submodels intently, as long as the model shows some advantage over a baseline prediction.



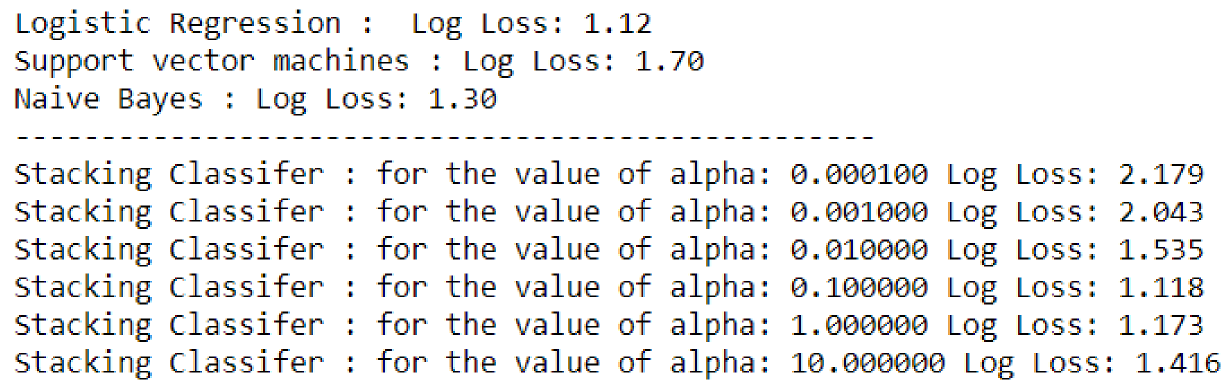
Stacking model processing methods(courtesy: www.kdnuggets.com)

In the stacking model, we are comparing the three interpretable models which are Logistic regression, Linear Support Vector Machine, and Naïve Bayes. The log-loss is 1.12, 1.70, 1.30 respectively for these models. Therefore, the stacking classifier is generated using the Logistic Regression Model and the minimum log-loss 1.11 is achieved at alpha = 0.1.



```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotC
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

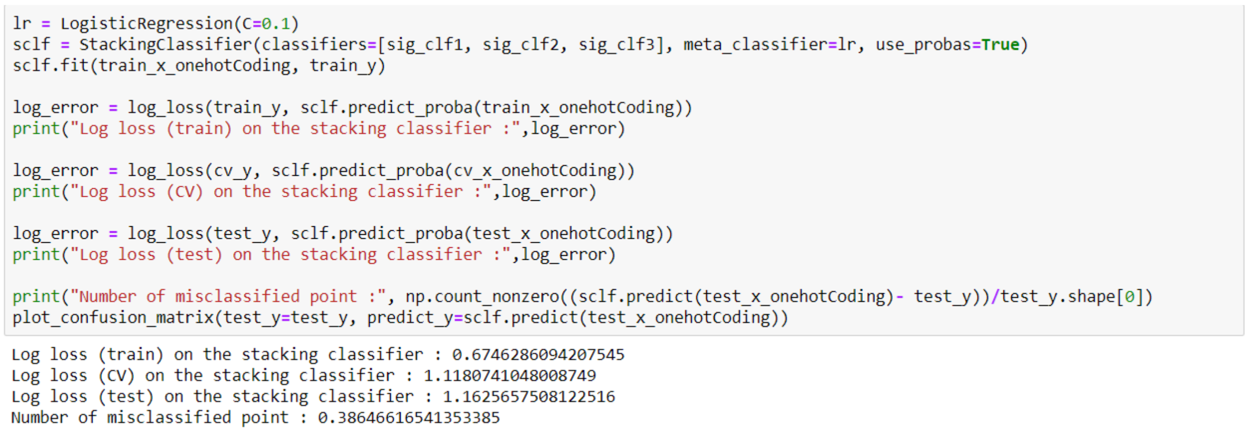
Stacking the models and generating alpha values



```
Logistic Regression :  Log Loss: 1.12
Support vector machines : Log Loss: 1.70
Naive Bayes : Log Loss: 1.30
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.043
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.535
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.118
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.173
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.416
```

Log loss results

The stacking classifier produces the train, cross-validation, and tests log-loss values as 0.67, 1.18, 1.16 respectively. The misclassified points were 38.64% which means that the model predicts 61.36% points correctly. The confusion, precision and recall matrix support these results.



```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of misclassified point :", np.count_nonzero((sclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

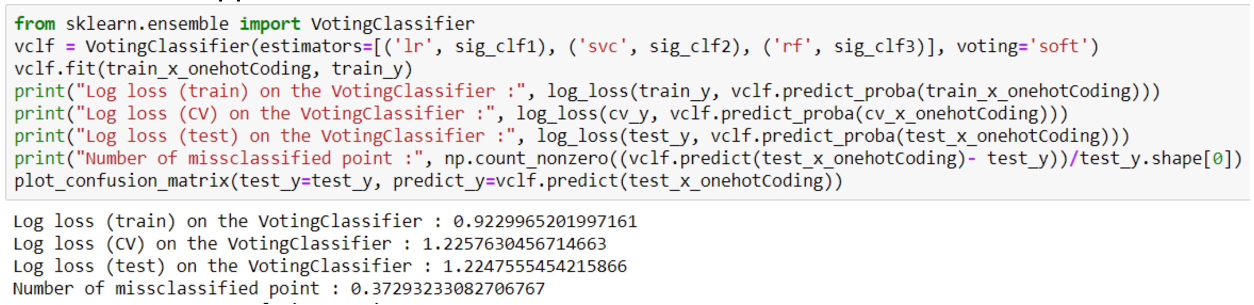
```
Log loss (train) on the stacking classifier : 0.6746286094207545
Log loss (CV) on the stacking classifier : 1.1180741048008749
Log loss (test) on the stacking classifier : 1.1625657508122516
Number of misclassified point : 0.38646616541353385
```

Final results for Stacking Classifier

4.12.7 Maximum Voting Classifier

Voting is one of the simplest ways of combining predictions from multiple machine learning algorithms. It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data. You can create a voting ensemble model for classification using the VotingClassifier class.

The maximum voting classifier uses Logistic Regression, Linear Support Vector Machine, and Random Forest Classifier and produces the train, cross-validation, and test log-loss values as 0.92, 1.22, 1.22 respectively. The misclassified points were 37.29% which means that the model predicts 62.71% points correctly. The confusion, precision and recall matrix support these results.



```
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.9229965201997161
Log loss (CV) on the VotingClassifier : 1.2257630456714663
Log loss (test) on the VotingClassifier : 1.2247555454215866
Number of missclassified point : 0.37293233082706767
```

Log Loss Results for Stacking Classifier

SPECIFICATIONS

Both training and test data sets are provided via two different files. The variants file provides information about the genetic mutations, whereas the test data file provides the clinical evidence in text format that pathologists use to classify the genetic mutations. Both the data files are linked via the ID field.

The genetic mutation with ID = 15 in the file training_variants can be classified using the clinical evidence in text format with ID = 15 in the file training_text.

File Descriptions:

- **training_variants** - a comma separated file containing the description of the genetic mutations used for training. Fields are an ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the amino acid change for this mutations), Class (1-9 the class this genetic mutation has been classified on)

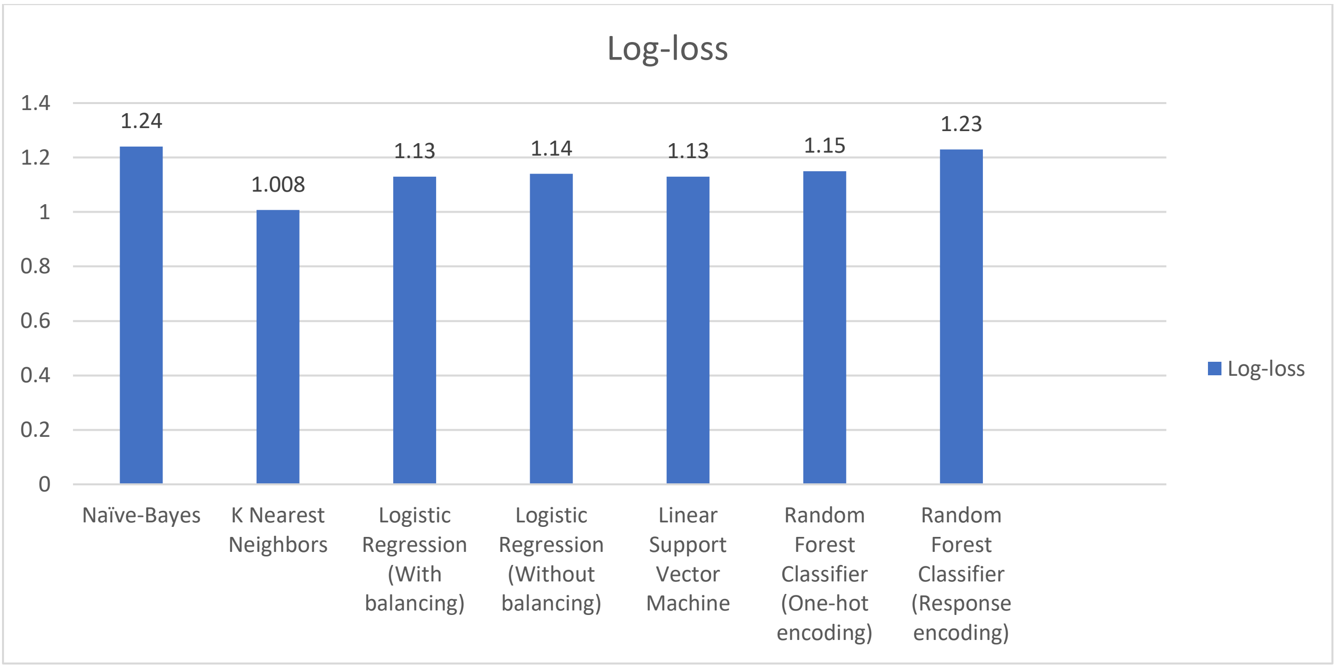
- **training_text** - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are an ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

- **test_variants** - a comma separated file containing the description of the genetic mutations used for training. Fields are an ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the amino acid change for this mutations)

- **test_text** - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are an ID(the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

RESULTS

Model	Train log-loss	Cross-Validation log-loss	Test log-loss	Miss-classified %	Log-loss
Naïve-Bayes	0.9	1.24	1.27	38.34	1.24
K Nearest Neighbors	0.48	1	1.07	31.95	1.008
Logistic Regression (With balancing)	0.63	1.13	1.11	34.2	1.13
Logistic Regression (Without balancing)	0.62	1.14	1.15	33.08	1.14
Linear Support Vector Machine	0.75	1.13	1.15	35.15	1.13
Random Forest Classifier (One-hot encoding)	0.68	1.15	1.14	42.66	1.15
Random Forest Classifier (Response encoding)	0.05	1.23	1.31	41.72	1.23



The above graph shows the trends of the Log-Loss values across the algorithms used

CONCLUSION

A well-curated characterized gene database promotes accurate phenotype-driven gene analysis. It enables classification as new studies are published, providing patients with a diagnosis and opportunities for therapeutic options, and an end to their “diagnostic journey.” Over time and with consistent literature review and clinical validity curation, we expect that many patients with candidate genetic etiologies will receive a definitive diagnosis via reclassification reports. Our machines learning modeling and analysis highlights the importance of careful literature curation and evaluation using a system of clinical validity scoring optimized for use in a diagnostic laboratory. This classification system is simple enough to be quickly implemented, and it can specifically guide reporting decisions at the important boundary of limited and moderate evidence, determining whether a gene is characterized. Further, we find that review of previous research literature while updating the clinical validity of gene-disease relationships can contribute to improved patient care, and classification reports can increase diagnostic rate.

The minimal value for the Log-Loss is obtained on implementing K Nearest Neighbors algorithm on the gene, variation, and text data value points. Although the Log-Loss for K Nearest Neighbors algorithm is less, we choose Logistic Regression (With balancing) because KNN gives an overfitted value(1.008).

As a future scope, the similar gene mutation and classification techniques can be extended to find a cure to diseases other than Cancer and can be a breakthrough technology in personalized medical space.

REFERENCE AND SOURCES

1. Kaggle Competition: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
2. Classification of Genes: Standardized Clinical Validity Assessment of Gene-Disease Associations Aids Diagnostic Exome Analysis and Reclassifications: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5655771/#humu23183-bib-0010
3. Naïve Bayes Classifier: https://en.wikipedia.org/wiki/Naive_Bayes_classifier
4. Logistic Regression: https://en.wikipedia.org/wiki/Logistic_regression
5. K-nearest neighbors Algorithm: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
6. Linear SVM: https://en.wikipedia.org/wiki/Support-vector_machine
7. Linear SVM (Medium): https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
8. Understanding the SVM algorithm for examples: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
9. Random Forest: https://en.wikipedia.org/wiki/Random_forest
10. The Random Forest Algorithm: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd