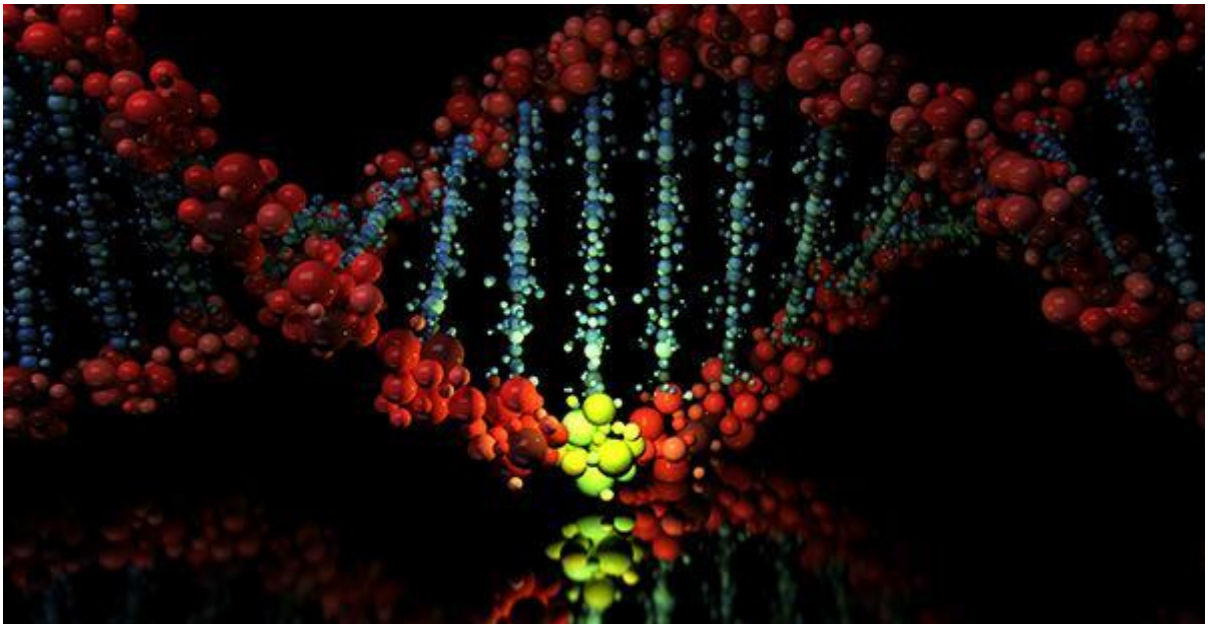




Personalized Medicine Redefining Cancer Treatment



**Guided By
Dr.Daniel Soper**

Team Members

**Arindam Roy
Gayathri Pujari**

**Cherish Reddy
Sofia Khaja**

Table of Contents

1.	Executive Summary	4
2.	Project Overview	4
3.	Problem Statement	6
4.	Research Questions and Goals	7
	4.1 Research questions.....	7
	4.2 Goals	9
5.	Methodology.....	10
6.	Exploratory Data Analysis.....	10
	6.1 Data Analysis	10
	6.2 Translating the problem into machine learning problem:	15
7.	Data Preprocessing	16
	7.1 Merging The Data Files	18
	7.2 Cleaning The Data And Handling Missing Values.....	18
	7.3 Creating Train, Test And Cross-Validation Set.....	19
	7.4 Data Distribution In Train, Test And Cross-Validation Set	20
8.	Evaluation Metrics.....	22
	8.1 Log Loss Vs. Accuracy	22
	8.2 Visualization of Log Loss.....	24
	8.3 Log Loss in Multi-class Classification.....	25
9.	Implementation Strategy	26
10.	Text Data Evaluation	29
	10.1 Text Data Handling	29
	10.2 Problem with Categorical Data:.....	29
	10.3 Methods /Techniques used with Categorical Data:.....	29
	10.3.1 One-hot Encoding:	29
	10.3.2 Bag-of-Words Model for Text Analysis	31
	10.3.4 Response Encoding.....	33
	10.3.5 Laplace Smoothing	35
	10.3.6 Calibrated Classifier.....	35
	10.3.7 Stochastic Gradient Descent (SGD)	36
11.	Evaluating The Columns	37
	11.1 Gene Column.....	38

11.2 Variation Column :	40
11.3 Text Column	42
12. Building Machine Learning Models	43
12.1 Machine Learning Framework	43
12.2 Data Preparation for ML Models	44
12.3 Combining The Features	46
12.4 Machine Learning Models Implemented	47
12.4.1 Naïve Bayes	47
12.4.2 K- Nearest Neighbours:	50
12.4.3 Logistic Regression:	52
12.4.4 Linear Support Vector Machine:	57
12.4.5 Random Forest Classifier	60
12.4.6 Stacking Model	67
12.4.7 Maximum Voting Classifier:	69
13. RESULTS:	70
14. Conclusion:	71
15. Future Scope:	73
16. Reference And Sources	74

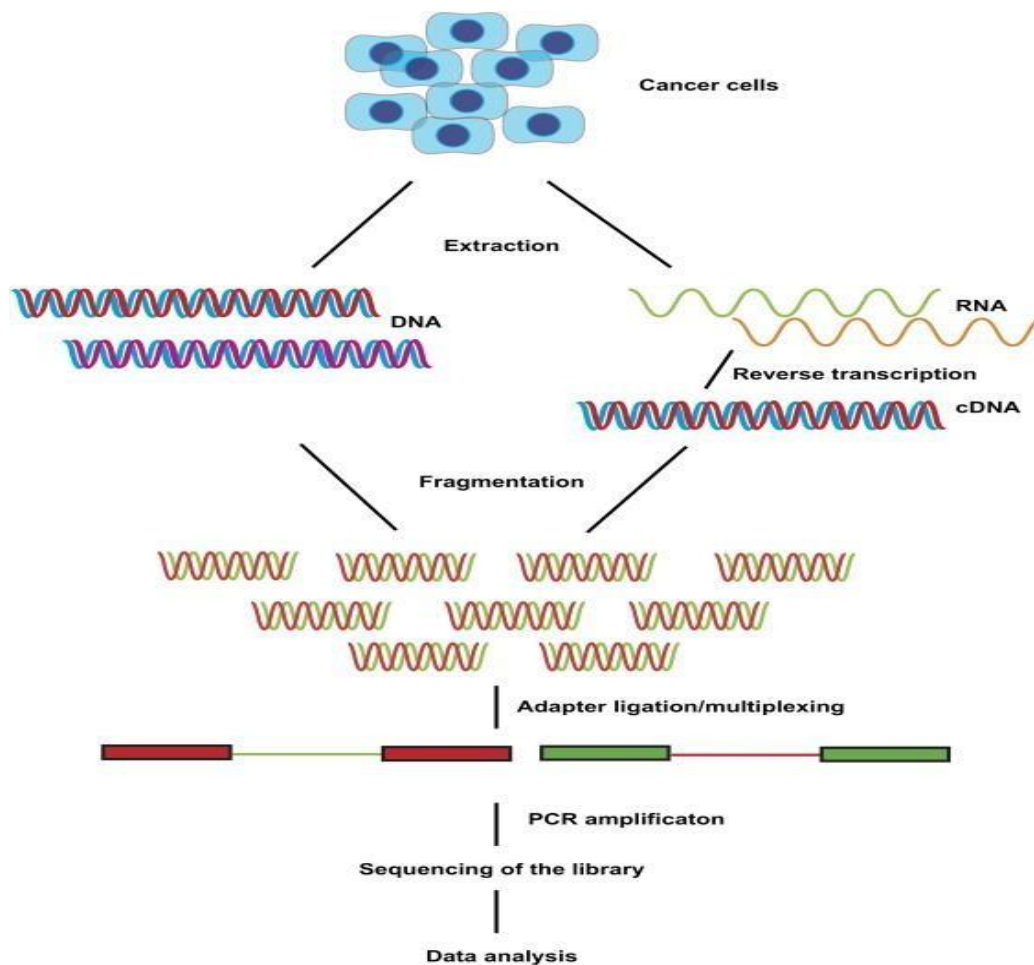
1. Executive Summary

Cancer is a disease that results when a cellular change in a mammal body causes an uncontrollable growth and cell division extensively. Some cases of cancers can be visible in the form of tumors, but there are few known as leukemia, which isn't visible. Cancer may be caused due to multiple reasons like alcohol consumption or extensive smoking or being physically inactive or even through genetics. The breakthroughs and advancement are happening at a rapid pace involving the treatment of cancer. Researchers are trying to identify as many genetic mutations possible in different kinds of cancers and the techniques which are more practical in sequence to the occurring tumors so that there can be a development in the targeted therapies. Hence, knowing the patient's genetic and molecular structure plays an important role. A lot of discussions have been going on over the past few years on how genetic testing is going to intrude on the way diseases like cancer are treated. But this is only happening partially because a vast amount of manual work is still required. A cancer tumor can have millions of genetic mutations, and distinguishing the mutations leading to cancer growth from the neutral ones is the challenge. Currently, since this task is being done manually, it is very time-consuming. The mutations are classified using text-based clinical evidence. Analyzing a patient's tumor and determining the combination of drugs that work best for them is what personalized cancer treatment is about

2. Project Overview

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 techniques 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. A lot has been said during the past several years about how precision medicine and, more concretely, how genetic testing is going to disrupt the way diseases like cancer are treated. But this is only partially happening due to the huge amount of manual work still required.



Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers). Currently, this interpretation of genetic mutations is

being made manually. This is a very time-consuming task where a clinical pathologist must manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

The purpose of the project is to develop a Machine Learning algorithm that uses the above knowledge base as a baseline, automatically classifies genetic variations. As a machine learning expert, we may not be a domain expert, but still, we will solve that domain problem using machine learning.

3. Problem Statement

In this project, we will be demonstrating a machine learning algorithm that is trained and tested with a fixed set of data obtained from Kaggle, which classifies the genetic mutations based on clinical evidence (text), which will mainly help in textual classification and analysis. In the human body, we have genes occurring in a sequence. If any variation occurs in those genes, then it might lead to cancer. It takes unnecessary manual work to go through each research paper on a different gene mutation that causes cancer, and accurate lab test results are needed to provide treatment for a specific cancer type. Our main aim of the project is to deal with what kind of gene mutation can lead to what type of cancer class. In our current problem, we take nine cancer classes ranging from 1 to 9. We have the clinical pathologist (domain expert) of the medical domain and with knowledge about genes, their variations or mutations, and which mutations result in cancer. The challenge we face here is, the possible gene mutations can be more than 1000, and we have a lot of research papers related to each mutation of each gene. So, the clinical pathologist spends an enormous amount of time reading all the research texts or documents to identify which genetic mutation belongs to which cancer type and then

suggests the appropriate treatment. Hence, as a data analyst, our goal is to replace those manual texts with our machine learning model, and thus, we save more time for more quick and precise treatment.

4. Research Questions and Goals

4.1 Research questions

Q1. How can we classify the possible genetic mutation occurring in a gene which is more likely the cancer-causing drivers in a patient?

Solution: For this research problem, we have been provided two datasets:

One dataset that includes various genes, their possible mutations, and the 9 unique cancer classes to which these genetic mutations can likely to fall under, i.e., becoming the possible cancer-causing tumors in a patient.

Another data set includes the research/ clinical evidence information in the form of text related to that genetic mutation and cancer classes.

As a data science expert, using the above information, our objective is to design a machine learning model that uses a textual analysis method for combining the gene, mutation, and clinical/research information into one feature and predicting the and probabilities for occurrence in each of those unique 9 cancer classes.

The class with the highest probability shows that the feature is more likely to fall under that cancer class. Further, this information will be used by the clinical pathologist for their cancer analysis and treatments.

Q2. Which predictive algorithm is more likely to classify the specific genetic variation into a cancer type class with a high probability occurrence rate?

Solution: For this medical problem, we have implemented 9 predictive algorithm models using the textual analysis methods that predict the accuracy for a given text feature, i.e., a combined feature of the gene, variation, and associated research text to get classified into those 9 unique cancer classes.

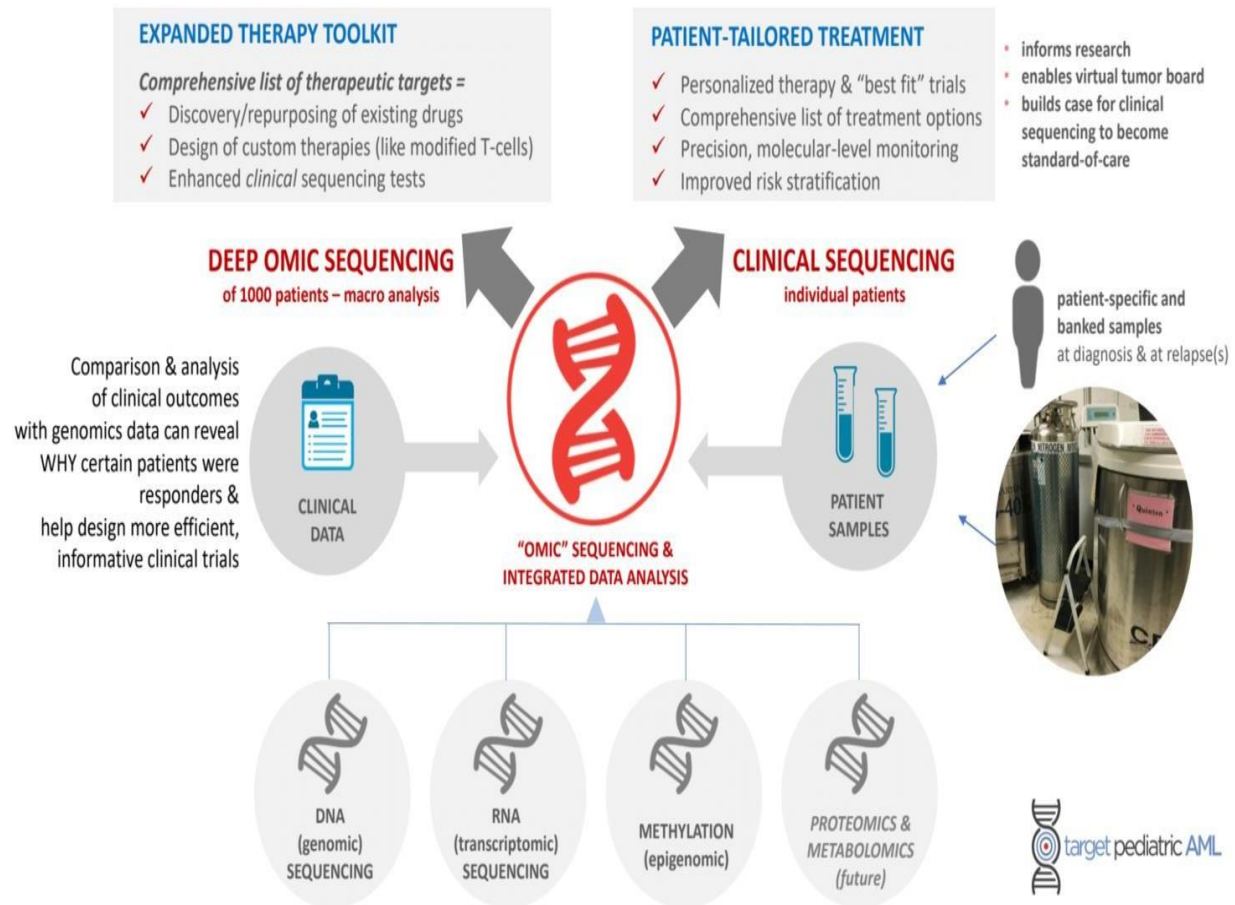
As per our model accuracy and log loss metric results, we considered logistic regression to be the better model to predict those features into those cancer classes. However, in the future scope, we would like to implement more predictive algorithms and compare results to decide the best model to be applied for actual medical treatment.

Q3. What are the probable text analysis methods to translate clinical evidence into numerical features that our machine learning model/classifiers can work with for accurate output?

Solution: For our better understanding, time restriction and implementation, we decided to use the one-hot encoding and responding technique with the Bag of Words (BoW) vectorization method and other classifiers like Calibrated Classifier and Stochastic Gradient Descent(SGD) to convert our categorical features into an appropriate format for our machine learning models.

In our future work, we want to explore and implement more advanced textual analysis methods, including TF-IDF, Word2Vec, and neural network, for our categorical data transformation in our machine learning model.

4.2 Goals



- 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.

5. 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.

6. Exploratory Data Analysis

6.1 Data Analysis

The training_text has 3321 rows, and training_variants has 3321 rows. The training_variants dataset has nine 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.

```
In [2]: # Loading training_variants. Its a comma seperated file
data_variants = pd.read_csv('training/training_variants')
# Loading training_text dataset. This is seperated by ||
data_text = pd.read_csv("training/training_text", sep="\\|\\|", engine="python", names=["ID", "TEXT"], skiprows=1)
```

```
In [3]: data_variants.head(3)
```

```
Out[3]:
```

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2

```
In [8]: data_text.head(3)
```

```
Out[8]:
```

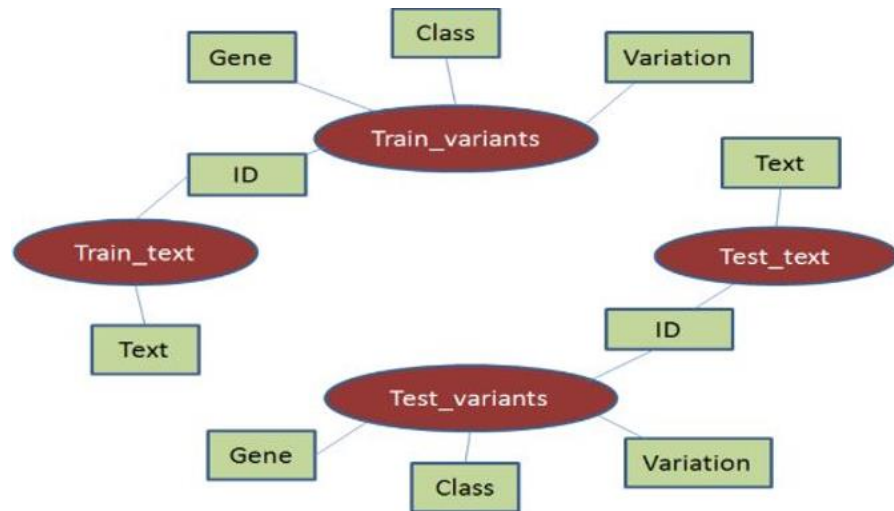
	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...

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

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

```
In [12]: # checking the dimentions
data_text.shape
```

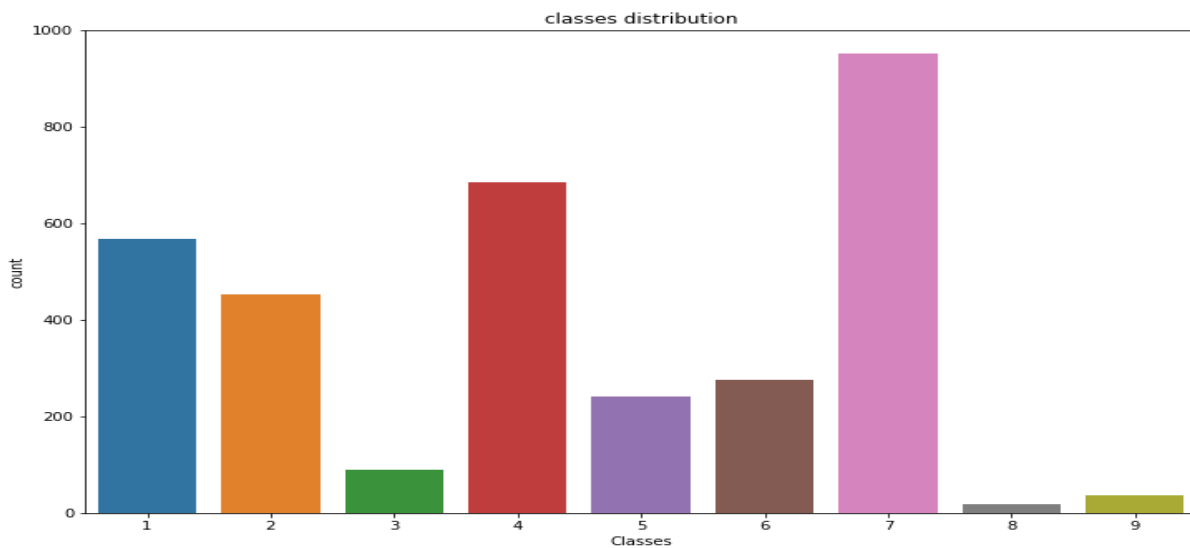
```
Out[12]: (3321, 2)
```



ER Diagram for our Dataset

Below are some of the following graphs/tables that we figured it out during our initial preliminary data analysis (EDA):

1. Class Distribution



This bar graph depicts the distribution of unique cancer classes in our dataset.

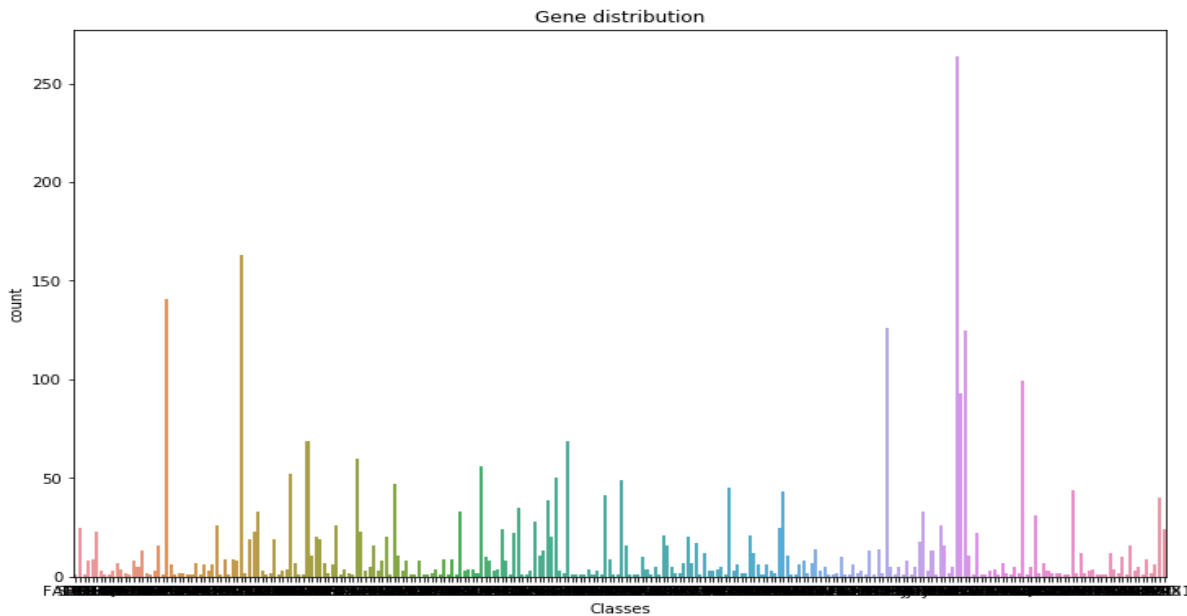
From the above, we conclude that:

- Class 7 is the most dominating class among all, i.e., have the highest data distribution.

- Class 8, Class 9, and Class 3 have the least data among all.

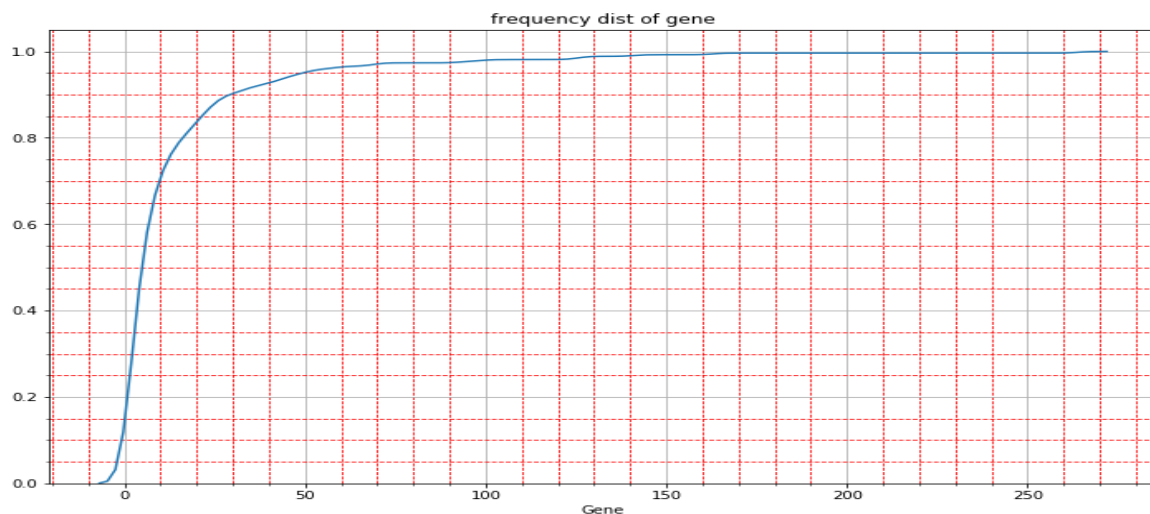
It shows that the given dataset is quite unbalanced.

2. Gene Distribution:



From the above Gene distribution plot, we can observe that few classes are super dominant, i.e., have the highest data distribution and are more likely to get predicted.

3. Genes Frequency Distribution:



From the above gene frequency distribution, we observed that:

- 20 types of gene types constitute the 80% total data
- 50 types of gene types represent 87.5% of complete data

Hence, these 20 gene types are playing a significant role in classifying genetic mutations.

4. Top 20 Genes Identification:

What are these top 20 genes?

```
In [33]: data_variants["Gene"].value_counts()[:20]
```

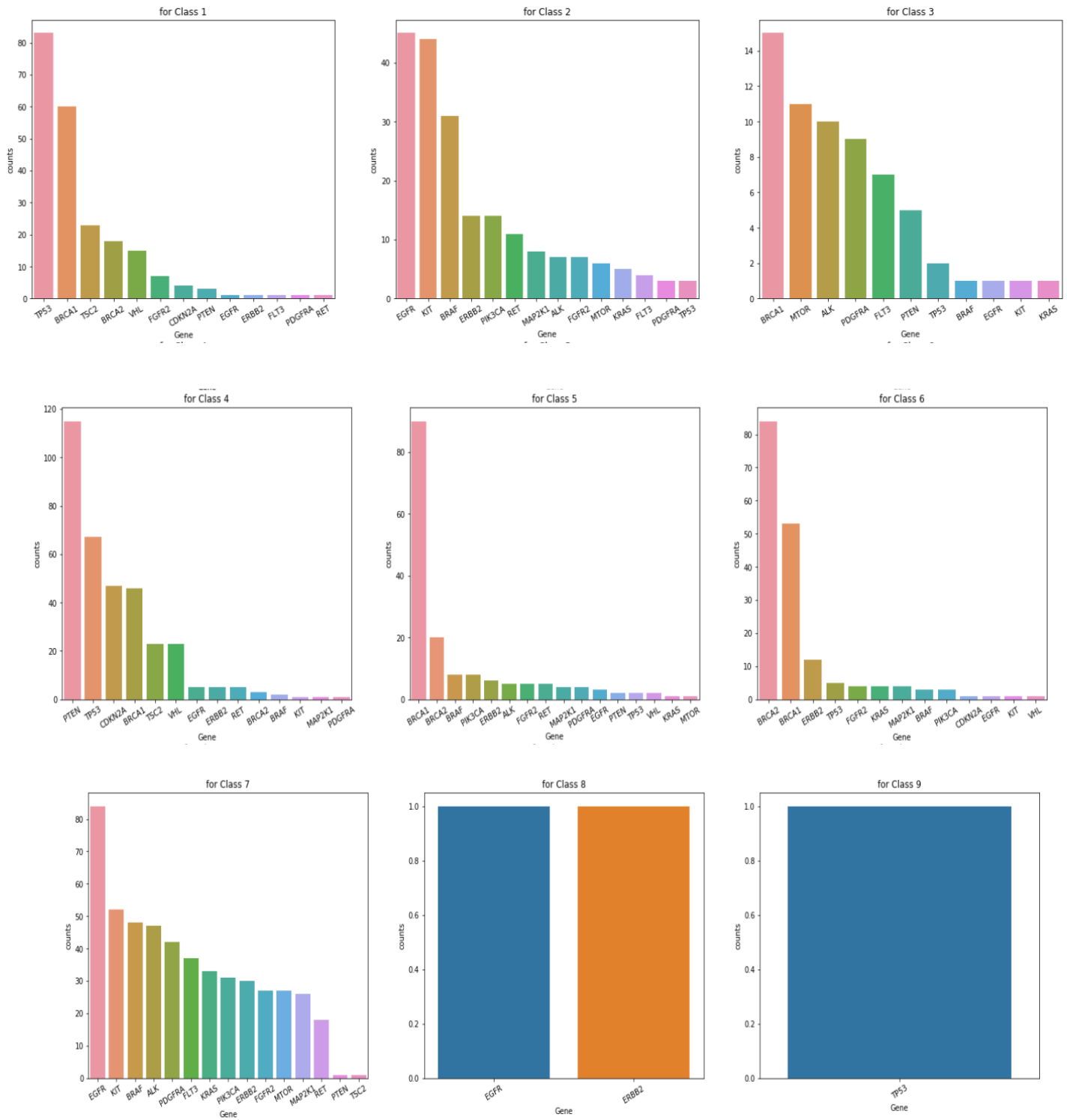
```
Out[33]: BRCA1      264
          TP53       163
          EGFR       141
          PTEN       126
          BRCA2      125
          KIT        99
          BRAF        93
          ALK         69
          ERBB2       69
          PDGFRA      60
          PIK3CA      56
          CDKN2A      52
          FGFR2       50
          FLT3        49
          TSC2        47
          MTOR        45
          KRAS        44
          MAP2K1      43
          VHL         41
          RET         40
          Name: Gene, dtype: int64
```

The above table depicts the top 20 genes, and from the above, we conclude that:

The top 5 genes named BRCA1, TP53, EGFR, PTEN, and BRCA2 are super dominant

BRCA1 is the leading gene which constitutes most of the data.

5. Top 20 Gene Distribution among 9 Classes:



From the above, we have the following conclusions:

For Class 8 and Class 9: EGFR, ERBB2, and TP53 are the most dominating genes.

EGFR is dominating gene in Class 8, 7 and 2

- TP53 is the most dominating gene in Class 1 and 9
- BRCA1 is dominating gene in class 3 and 5
- PTEN is dominating gene in class 4
- For Class 8 and 9 top 20 Genes appeared only once. It suggests class 8 and 9 are most dissimilar from others, or the total counts of classes 8 and 9 are severely underrepresented.

6.2 Translating the problem into machine learning problem:

Here given a new gene, variation, and research text associated with that gene, we want to predict which cancer class type it belongs to. The output variable class has nine unique discrete values. Hence it is a classification problem, and since there are multiple discrete outputs possible, so it is a Multi-Class classification problem. While building a model for medical problem errors needs 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.

Since it is a medical problem, the correct results are very important. The error can be costly here, so we would like to have a result for each class in terms of Probability. We might not be much bothered about the time taken by the ML algorithm as far as it is reasonable. We also want our model to be highly interpretable because a medical

practitioner wants to also give proper reasoning on why the ML algorithm is predicting any class.

E.g., Let's say the probability that a patient is more likely to have a cancer type 3 given the information about the gene, its mutation and associated research text, i.e.,

$$P(y=3|\text{Gene, Variation, Text}) = 0.99 \Rightarrow \sim 99\% \text{ confidence.}$$

Hence, the interpretation of our prediction derived from machine learning model can make our confidence stronger as it can also tell the reason as to why we are ~99% confidence that this patient is more likely to have a cancer type 3.

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 could be compromised to some extent for this prediction without increasing the error rate.

7. Data Preprocessing

Here 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 use human language data for applying in Statistical Natural Language Processing (NLP). NLP 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. We have a vast amount of text data. So, we need to pre-process it before using it in our machine learning model. The text pre-processing includes:

- Removal of the stop words such as “is”, “an”, “a”, “the” etc.
- Replacing multiple spaces with a single space.
- Replacing the special characters with a single space.
- Removing numbers like 1.1, 1.2, etc.
- Converting the whole text to lower case to bring the text to the same scale.

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

```
In [15]: > def data_text_preprocess(total_text, ind, col):  
# Remove int values from text data as that might not be imp  
if type(total_text) is not int:  
    string = ""  
    # replacing all special char 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))  
    # bring whole text to same lower-case scale.  
    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')
```

7.1 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()
```

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...

7.2 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:

- Removing the rows
- 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 the 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

We can see many rows with missing data. We do the data imputation for these null values by merging Gene and Variation column.

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

cross checking it once again if there is any missing values

```
In [20]: result[result.isnull().any(axis=1)]
```

```
Out[20]:
```

	ID	Gene	Variation	Class	TEXT
--	----	------	-----------	-------	------

7.3 Creating Train, Test And Cross-Validation Set

Before the data was split, all the spaces in the Gene and Variation column were replaced by _ (underscore symbol). The data was split into training, testing, and cross-validation because when hyperparameter tuning is done, we try to improve accuracy with respect to the Test set, which can sometimes lead to bad models. In order to do that:

- We build and train our ML model on the training data.
- We use the cross-validation data to do the hyper-parameter tuning and validation.
- Once we get the final ML model, then we validate it using the testing data.

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 Cross-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 validation 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)

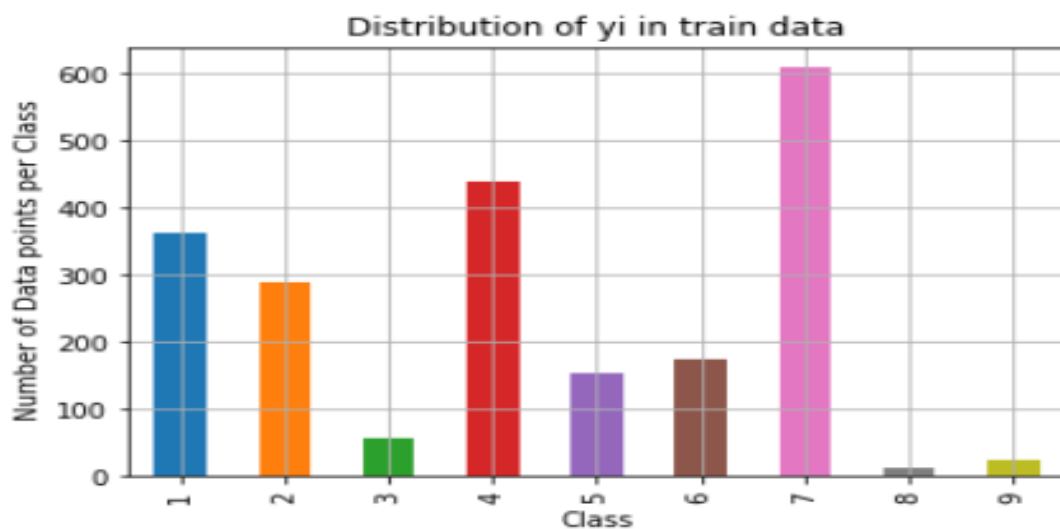
In [23]: > print('Number of data points in train data:', train_df.shape[0])
          > print('Number of data points in test data:', test_df.shape[0])
          > print('Number of data points in cross validation data:', cv_df.shape[0])

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

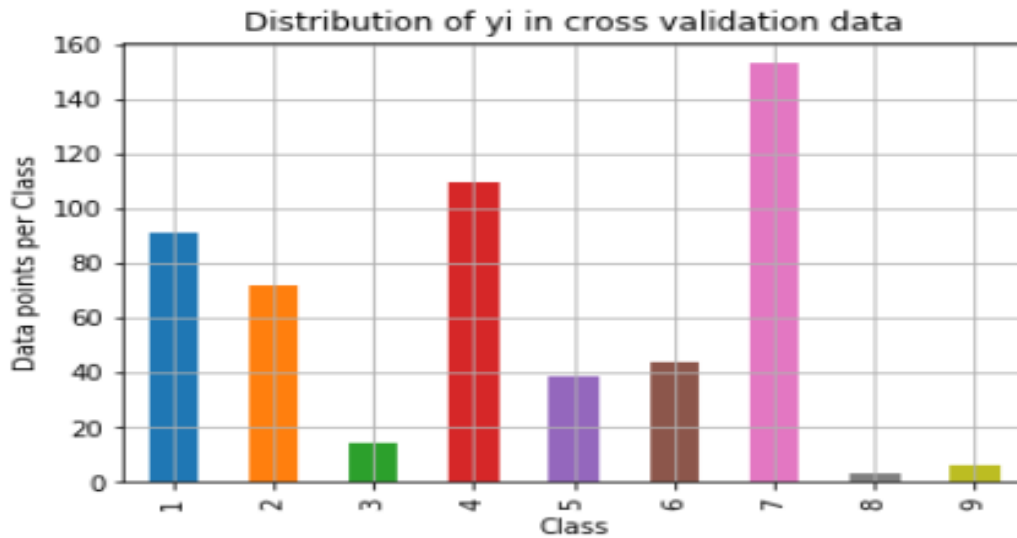
```

7.4 Data Distribution In Train, Test And Cross-Validation Set

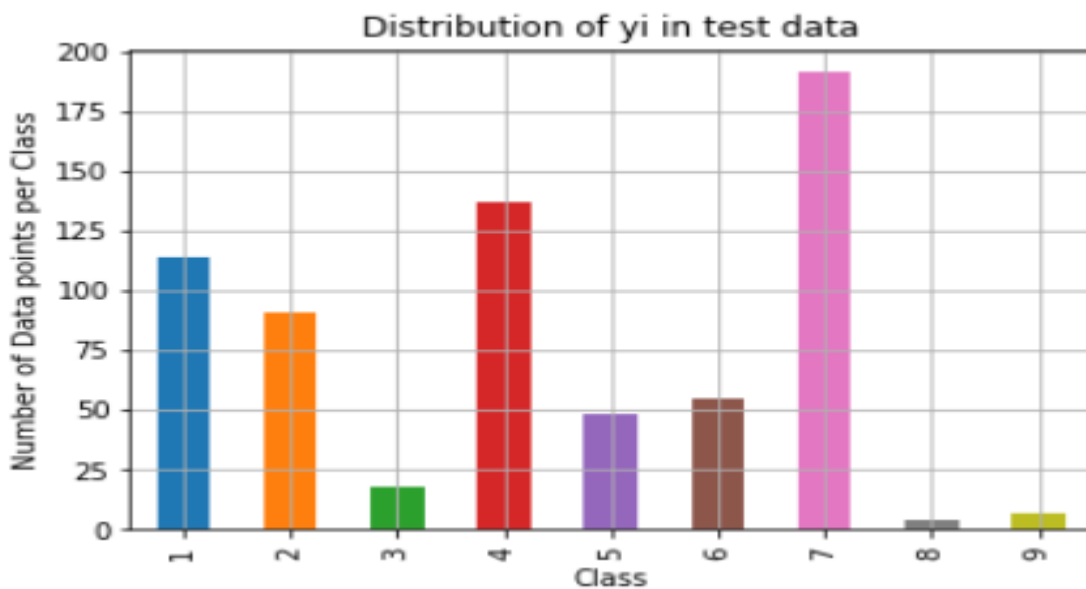
It is important that the distribution of data is similar in all three sets for good predictions. E.g., Suppose after the random split, if the training data has all the Class 4 data, but the cross-validation and testing data has no Class 4 data, then our prediction for the mutations for the Class 4 will be not accurate. Here, we are looking for the data distribution using a bar plot for all the three training data, cross-validation data, and testing data as below:



Data distribution per class in Training data



Data distribution per class in Cross-validation data



Data distribution per class for Testing data

It was verified that the distribution of all these sets was almost similar. From the distribution, we can also observe that Class 7 dominates over the other classes, while Class 8 and Class 9 have less distribution compared to other classes.

8. Evaluation Metrics

For our ML model, we need an evaluation metrics to compute and analyze how well our model is doing in predicting those cancer classes. So, here we are going to use two evaluation matrix:

- Confusion Matrix, where we will use the accurate term to compare the results.
- Log- Loss, i.e., the ML model, which has the minimum log loss value, will be better than others.

8.1 Log Loss Vs. Accuracy

- Due to its yes or no nature accuracy is not always a good indicator. Accuracy is the number of predictions where the actual value of your predicted value is equal to.
- Log Loss takes the prediction's uncertainty into account, depending on how much it differs from the actual mark. That gives us a more nuanced view of our model's success.

Below is an illustration of how to log loss is going to help us in deciding which ML model is performing better. Suppose we have x values as input, and the output variable is y where we have two classes say 1 and 0. Let's say we have the prediction values in probability terms as to what is the probability that each of those x values belongs to the actual classy. Also, the threshold we assume here is 0.5 i.e.,

If $P(x \text{ belong to Class}) \geq 0.5 \rightarrow \text{Class 1}$

If $P(x \text{ belong to Class}) < 0.5 \rightarrow \text{Class 0}$

Also, let's say we have computed the value of the log loss using the equation:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Where,

p = probability values of those predictions

yi = Original actual outputs or classes

X	Y	Probability (y'=P)	Log Loss
X1	1	0.9	$-\log(0.9) = 0.0457$
X2	1	0.6	$-\log(0.9) = 0.22$
X3	0	0.1	$-\log(0.9) = 0.0457$
X4	0	0.4	$-\log(0.9) = 0.22$

Interpretation from the above:

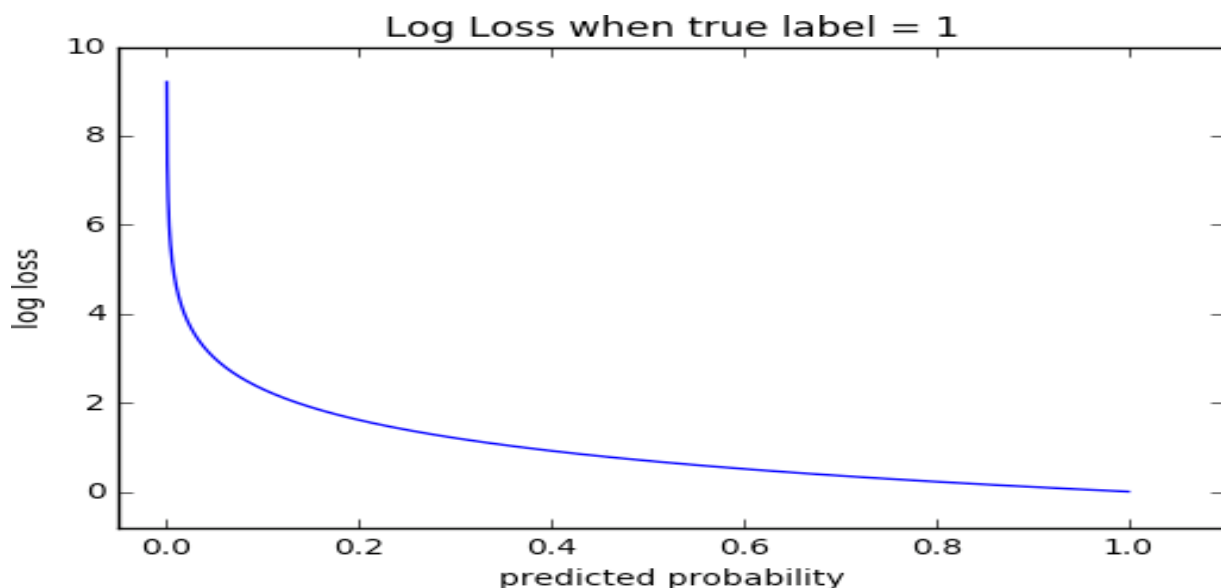
In terms of accuracy, the above-illustrated model accuracy is 100%, as the actual class is the same as predicted class.

- When our probability is 0.9, i.e., high probability to classify x into Class 1, our log loss value is 0.0457. But when the same Class 1 has probability 0.6, the log loss value is higher, i.e., 0.22.
- Similarly, when the probability is 0.1, i.e., high probability to classify as Class 0, the log loss value is low, i.e., 0.0457. But when the probability of classifying Class 0 is 0.4, the log loss value is 0.22.

Hence, the above draws our approach to evaluate our ML models, i.e., If the Log Loss value of the Prediction Model is minimum à It is a better ML Model for our purpose. Logarithmic loss tests a classification model's output where the input of prediction is a probability value between 0 and 1. Our machine learning models are aiming to reduce this value. A good model will lose 0 on log. Loss of logging increases as the expected probability diverges from the actual label.

8.2 Visualization of Log Loss

The following graph shows the spectrum of possible values for log loss given a true observation (isDog = 1). As the forecasted likelihood reaches 1, the loss of logs decreases gradually. However, as the predicted probability decreases, the loss of logs rapidly increases. Log loss penalizes all types of errors, but especially those predictions which are confident and incorrect.



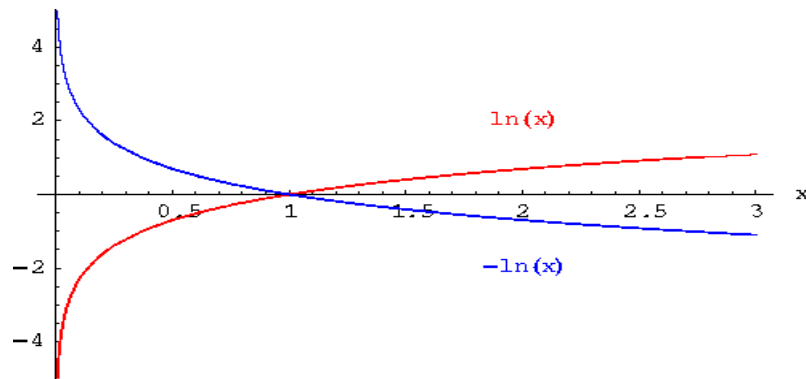
8.3 Log Loss in Multi-class Classification

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

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

Why the Negative Sign?

Log Loss uses the negative log to provide an easy metric for comparison. It takes this approach since negative values are returned by the positive log of numbers < 1 , which is difficult to deal with when comparing the output 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.

9. Implementation Strategy

As log-loss is being considered as the primary evaluation parameter, a Random Model was generated to compare all the log-losses. The log loss of the Random Model is compared against all the log losses of the other models. Any successful model would return a lower log-loss than this Random Model.

An array of zeros is generated to evaluate the log-loss of the cross-validation data for the Random Model. This array is 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' is used to get the index where the probability of getting a particular class was high. Hence, the log-loss for the cross-validation is evaluated. This value is around 2.47. Similarly, the log-loss of the Test data is evaluated and is approximately 2.51.

Building a Random model

For random model, we need to generate 9 random numbers because we have 9 class such that their sum must be equal to 1 because sum of Probability of all 9 classes must be equivalent to 1.

```
In [32]: test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]
```

```
In [33]: # we create a 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 2.47716883083
```

```
In [34]: # Test-Set error.
#we create a 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))

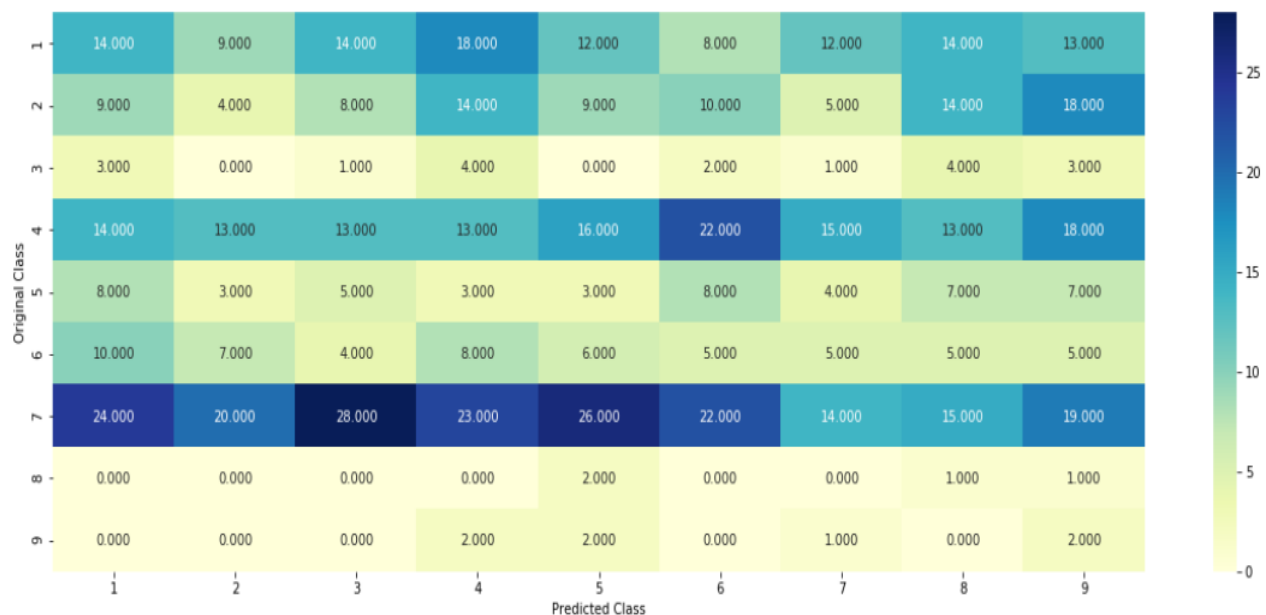
Log loss on Test Data using Random Model 2.51192909174
```

Finally, an array of predicted Class values was generated. In short, we will be computing the log loss values for both the testing data and cross-validation data for our ML model and random model and: ML Model Testing / Cross validation Log Loss > Random Model Testing / Cross validation Log Loss à ML model is better .We will also be using the confusion matrix, Recall, and precision matrix for evaluating our prediction models, i.e., we will construct a 9X9 heat map for each matrix where :

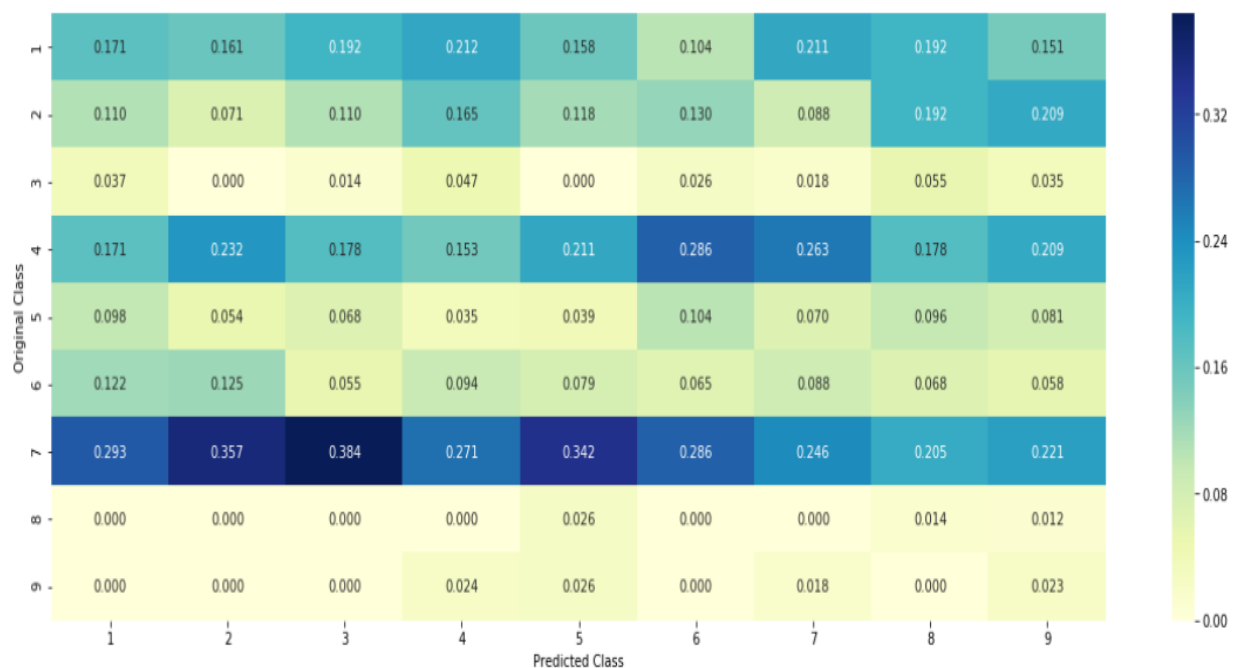
X axis → Predicted Class

Y axis → Actual Class

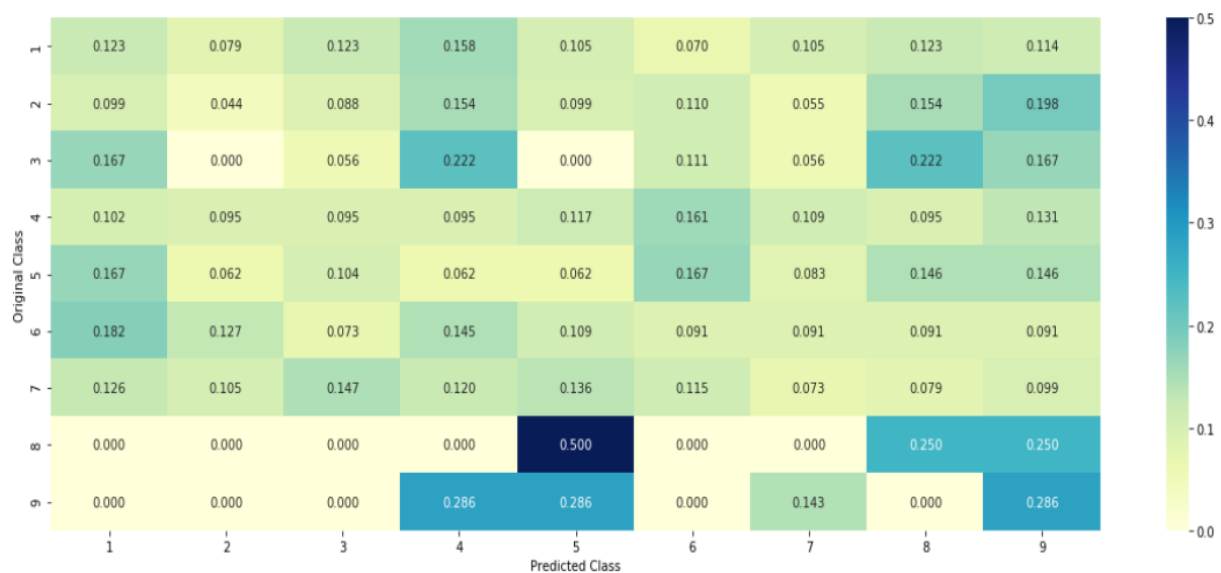
Eg. Below are the Confusion , Precision and Recall Matrix for our random model:



Random Model : Confusion Matrix



Random Model : Precision Matrix



Random Model : Recall Matrix

10. Text Data Evaluation

10.1 Text Data Handling

In our problem, we have three independent categorical variables, i.e., Gene, Variation and Text, and one dependent variable i.e., class. Now, we are going to see how much each categorical variable is impacting in predicting our class and then to convert these variables into a machine learning format. **Categorical data** are variables that contain label values rather than numeric values.

10.2 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 that a machine learning algorithm can process it.

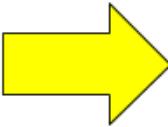
10.3 Methods /Techniques used with Categorical Data:

10.3.1 One-hot Encoding:

One hot encoding creates new (binary) columns, indicating the presence of each possible value from the original data. When we use one-hot encoding, the number of columns increases because all the unique values of the independent variable are made into columns, their presence is determined by one, whereas absence is determined by zero.

For example, consider a given categorical column Color with values:

Color			
Red			
Red			
Yellow			
Green			
Yellow			



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

The values in the original data are Red, Yellow, and Green. We create a separate column for each possible value. Wherever the original value was Red, we put a 1 in the Red column. Similarly, using one-hot encoding, for all the unique categorical values that we have in the gene, variation, and text column, we will create several individual columns and assigning them numbers. The drawback of one-hot encoding: if we have, say a huge number of unique categorical values, for example, 1000, that means one-hot encoding will generate 1000 new unique columns, which will ultimately increase the dimensionality of the data leading to the sparse data.

In other words, when we have large scale data, then it is possible that some of the expected values are missing, which further impacts the prediction accuracy of the machine learning algorithms. Also, algorithms like Random Forest or KNN will not perform well with one-hot encoding due to the high dimensionality of the data. But algorithms like Linear logistic regression and Linear SVM (Support Vector machines) work well with high dimensionality data.

10.3.2 Bag-of-Words Model for Text Analysis

For our better understanding and reducing the complexity of text analysis, we have implemented the Bag-of-Words (BoW), which is one of the methods of extracting the features from the text data for the use in our machine learning algorithms. A bag-of-words is a representation of text data that describes the occurrence of words within a document. In this approach, we look at the histogram of the words within the text, i.e., considering each word count as a feature. It involves two things:

1. A vocabulary of known words.
2. A measure of the presence of known words.

The steps which are involved in the Bag-of-Words model are:

- a) Collect the text data
- b) Design the vocabulary, i.e., making the list of all the unique words while ignoring the case and punctuations.
- c) Creating the document vectors, i.e., turning each text data (or document) into a vector that we can use as input or output for a machine learning model.
- d) Using scoring methods like
 - counting the number of times each word appears in a document
 - calculating the frequency that each word appears in a document out of all the words in the document.

For our project, we are implementing the Bag-of-words for a one-hot encoding method using CountVectorizer. CountVectorizer is used to convert the text data into the numerical format by transforming the text data into vectors using BoW.

It counts the total number of textual data like genes, variations and research text words in each categorical columns and makes them as columns in that vector. Then it counts the frequency of each word (or tokens) and places those values. For example, for our gene column in training data, using one-hot encoding with BoW, we have created a vector matrix of 2124X235 dimension as shown below:

```
In [54]: ▶ train_gene_feature_onehotCoding.shape
Out[54]: (2124, 235)
```

```
In [55]: ▶ train_gene_feature_onehotCoding.toarray()
Out[55]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

After performing the one-hot encoding for the gene , variation and text column for train data as below :

Features	Train Data Set
Gene	238
Variation	1953
Text	52816

Similarly, we compute the same for the cross-validation and testing data for the respective gene, variation and text column.

10.3.4 Response Encoding

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. In our case, for each of the rows that we have in the gene, variation, and text column, we will create nine unique columns corresponding to the nine Cancer classes. For each row/categorical data, we will compute the probability of that predicted class for that row using the Naïve Bayes theorem. For example,

Let's say we have a Gene column and also the actual classes that they belong to. Suppose we have a total of 100 values of genes and their corresponding Cancer classes as below:

	Gene	Class
Row 1	gi	2
Row 2	gi	2
Row 3	gi	3
	gi	3
	gi	3
	⋮	⋮
	⋮	⋮
	⋮	⋮

Let's say out of 100 rows, 30 rows belongs to Class 2 and 70 rows belongs to Class 3. So using response encoding, we will create 9 columns corresponding to each gene row and will compute the probability for each row for those 9 classes in each of those 9 columns as shown below:

	P(y=1 gi)	P(y=2 gi)	P(y=3 gi)						P(y=9 gi)	
	1	2	3	4	5	6	7	8	9	9 Classes
Naïve Bayes	$P(y=3 gi) = P(y=3) * P (gi)/P(gi) = 70/100 = 0.7$ $P(y=2 gi) = P(y=2) * P (gi)/P(gi) = 30/100 = 0.3$									

Finally we impute those probability values in each of the columns and thus replace each row of the gene column with 9 column values as shown below:

P(y=1 gi)	P(y=2 gi)	P(y=3 gi)								
0	0.3	0.7	0	0	0	0	0	0		Sum of all probabillites should be 1
1	2	3	4	5	6	7	8	9		

For response encoding, we have created custom functions named `get_gv_feature` and `get_text_responsecoding`, which takes the training, cross-validation, and testing data set and returns nine columned arrays of probability values for each of the textual data for respective cancer classes. After performing the response encoding for the gene, variation and text column for train data as below:

Features	Train Data Set
Gene	9
Variation	9
Text	9

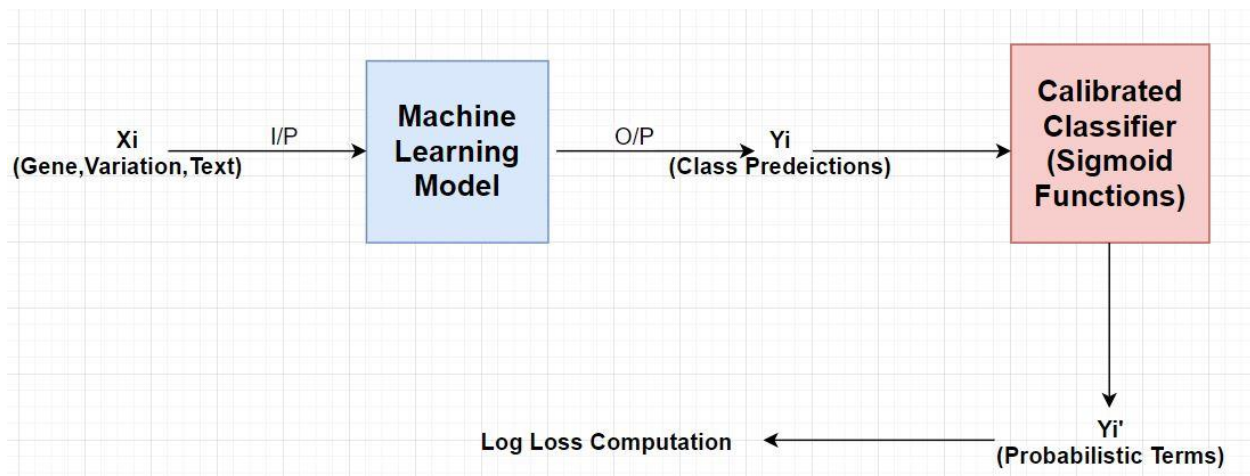
Similarly we compute the same for the cross-validation and testing data for the respective gene, variation and text column.

10.3.5 Laplace Smoothing

This is also called Additive smoothing. It is a statistical technique that will be used to smooth the categorical data. In our case, we are using it to solve the problems of zero probability, i.e., during the testing phase, there may be some words that might be present in the training data set but not a part of our testing data set or vice versa. Hence it will help us in shifting some probability from seen words to obscure words; in other words, assigning obscure words/phrases some likelihood of occurring. As we know that the splitting of the data happens randomly, and it is possible that some of the text data which might be present in the training data might be missed in our cross-validation or testing data or vice versa. Therefore, we are incorporating this technique while implementing the response encoding since we try to compute the probabilities of each of the text data of the gene, variation, and text column for their respective classes. We can avoid the zero probability value while computing for any missing term in either of the training, cross-validation, and testing data.

10.3.6 Calibrated Classifier

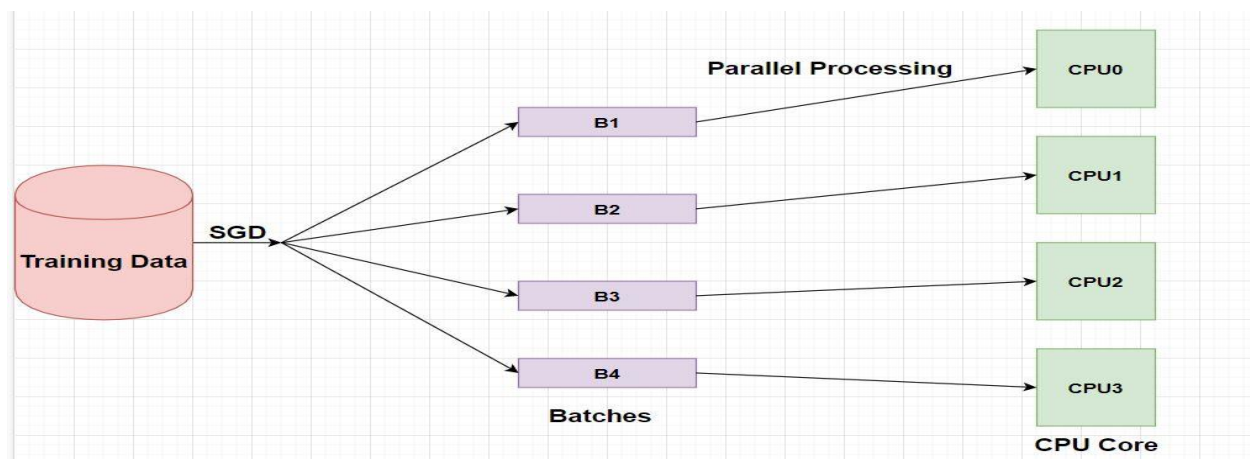
We use the Calibrated Classifier to make sure our machine learning model results come out in the form of probabilistic terms. Since we are using log loss as an evaluation metric, it is recommended that you use a calibrated classifier to get the output in probability format. We can also use calibration to improve our ML model such that the distribution and behavior of the probability predicted are similar to the distribution and behavior of probability observed in training data.



We remodel our ML model to incorporate the Calibrated Classifier using a function called Sigmoid to ensure our prediction comes in probability terms. We perform logistic regression on the output of the model to the actual label.

10.3.7 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an efficient optimization algorithm that is used to find the parameters/coefficients values of functions that minimize a cost function. In other words, we use it for the discriminative understanding of linear classifiers within a convex loss function like SVM & Logistic regression (LR). Results have shown its successful application to large-scale datasets. Here an update is performed to coefficients for each of the training instances rather than at the end of instances.



It is a simple and efficient approach for discriminative learning of linear classifiers under convex loss functions such as (direct) Support Vector Machines and Logistic Regression. We are implementing the SGD classifier provided by the Scikit-learn library in python to implement SGD classification. It helps us to optimize the performance of our machine learning algorithm. It uses the concept of batches, i.e., splitting our training data into mini-batches and then iterate over each batch to process. Processing each batch in parallel and gives a speed boost to our algorithm to deal with the text data.

11. Evaluating The Columns

The columns were evaluated to check whether they are impactful for predicting the Class column, which is the dependent variable. For this, we will build a simple model like logistic regression and will use every single categorical variable as a single input and try to predict the cancer classes. We will compute the log loss for each of those ML models and will compare it with our random model log loss. In addition to that, we will also compute the

- cumulative distribution for each of the categorical columns
- overlapping % between the training, test, and cross-validation data sets.

As we know that the data is splitting randomly, so we need to see the overlapping of each categorical data between the data-set, which ensures the stability. If the overlapping for a variable between the train, cross-validation, and testing data is low, that means we might not get accurate results. It is possible that some genes could be trained using our training data, but if they are not present in the testing data, then it will impact our

predictions. Therefore, the overlapping between sets → More stability in terms of predictions.

11.1 Gene Column

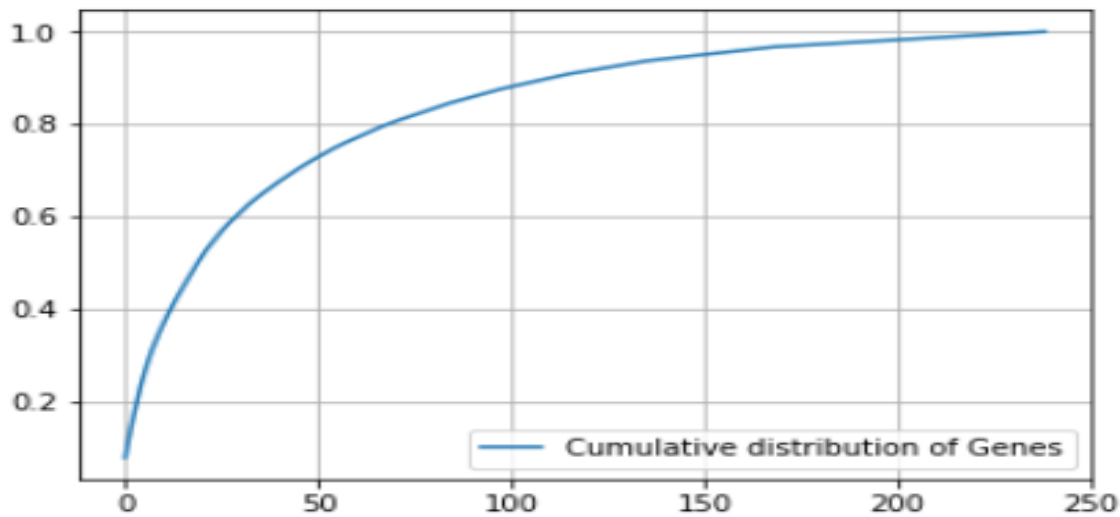
Here, as we know that the data is splitting randomly, so we need to see the overlapping of each categorical data between the data-set, which ensures the stability. If the overlapping for a variable between the train, cross-validation, and testing data is low, that means we might not get accurate results. It is possible that some genes might be trained using our training data, but if they are not present in the testing data, then it will impact our predictions.

```
In [44]: ▶ unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occurred most
print(unique_genes.head(10))
```

Number of Unique Genes : 239	
BRCA1	163
TP53	96
EGFR	84
PTEN	76
BRCA2	74
KIT	62
BRAF	56
ERBB2	51
ALK	43
PDGFRA	42

Name: Gene, dtype: int64

Number of Unique Genes



Cumulative Distribution of Genes

```
In [56]: # Lets use best alpha value as we can see from above graph and 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=
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
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=

For values of best alpha = 0.0001 The train log loss is: 1.03428893664
For values of best alpha = 0.0001 The cross validation log loss is: 1.1538252363
For values of best alpha = 0.0001 The test log loss is: 1.23969521463
```

Now lets check how many values are overlapping between train, test or between CV and train

```
In [57]: 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 [58]: 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 651 out of 665 : 97.89473684210527
2. In cross validation data 519 out of 532 : 97.55639097744361
```

Evaluation of log loss and overlap for gene column

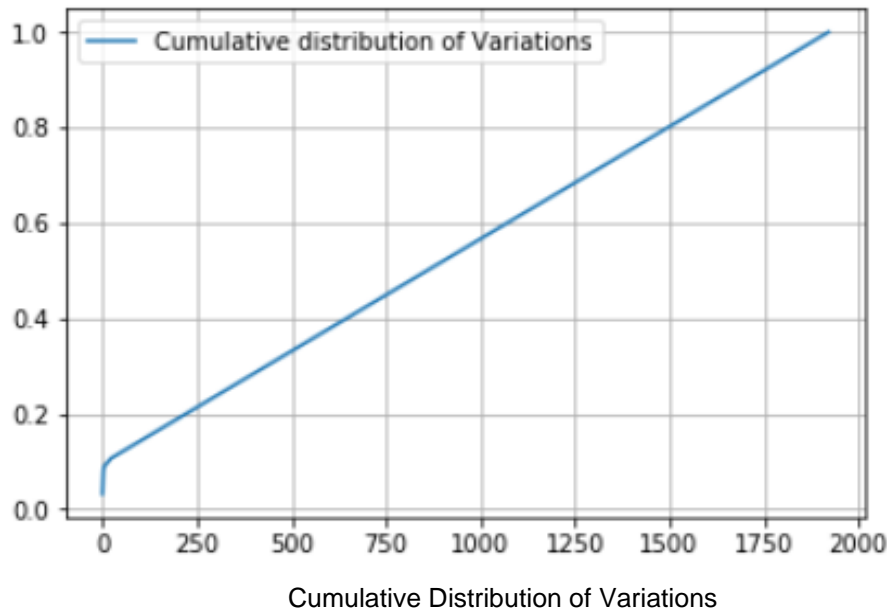
11.2 Variation Column :

From the Cumulative graph of Variation column, it is observed that the first 1500 unique variations, contributes almost 80% of the total values. The Laplace Smoothing is performed on the Variation column, and it is evaluated that the train, cross-validation, and test log-loss were approximately 0.74, 1.70, 1.71 for $\alpha = 0.0001$. Also, the overlap between train- test and train cross-validation is found to be 10.53% and 9.02%, respectively. It is therefore observed that the Variation column is a useful 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 [59]: ▶ unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occurred most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1922
Truncating_Mutations      67
Deletion                   53
Amplification              45
Fusions                    19
Overexpression             4
T58I                       3
Y42C                       2
G12V                       2
T286A                      2
E17K                       2
Name: Variation, dtype: int64
```

Number of Unique Variation



```
In [68]: 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=
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_
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=
4
```

For values of best alpha = 0.0001 The train log loss is: 0.747743968009
For values of best alpha = 0.0001 The cross validation log loss is: 1.70909981795
For values of best alpha = 0.0001 The test log loss is: 1.71064410395

```
In [69]: 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]
```

```
In [70]: 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 70 out of 665 : 10.526315789473683
2. In cross validation data 48 out of 532 : 9.022556390977442

Evaluation of log loss and overlap for variations column

11.3 Text Column

For the Text column, the unique words are stored in a dictionary, and the corresponding count value is incremented every time the word is encountered. A count vectorizer is built with all the words that occurred a minimum three times in the data. The total unique words are found to be 52816. Values in each row are converted such that the sum is one using the method of Response Encoding. Every feature is normalized and is trained using the same Vectorizer as the train data. It is evaluated that the train, cross-validation, and test log-loss are approximately 0.76, 1.15, 1.14 for $\alpha = 0.001$. Also, the overlap between train-test and train - cross-validation is found to be 96.54% and 97.09%, respectively. It is therefore observed that the Text column is a good prediction variable as it has a very low log-loss value compared to Random Model and high stability as the overlap is high.

```
In [73]: # 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 occurred
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 : 52816

Number of Unique words in Text

```
In [82]: > 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, 1))
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, 1))
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, 1))
```

For values of best alpha = 0.001 The train log loss is: 0.758074401927
For values of best alpha = 0.001 The cross validation log loss is: 1.1529733302
For values of best alpha = 0.001 The test log loss is: 1.13995955716

Evaluation of log loss for Text

```
In [84]: > 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")
```

96.549 % of word of test data appeared in train data
97.099 % of word of Cross Validation appeared in train data

Overlap of Text column

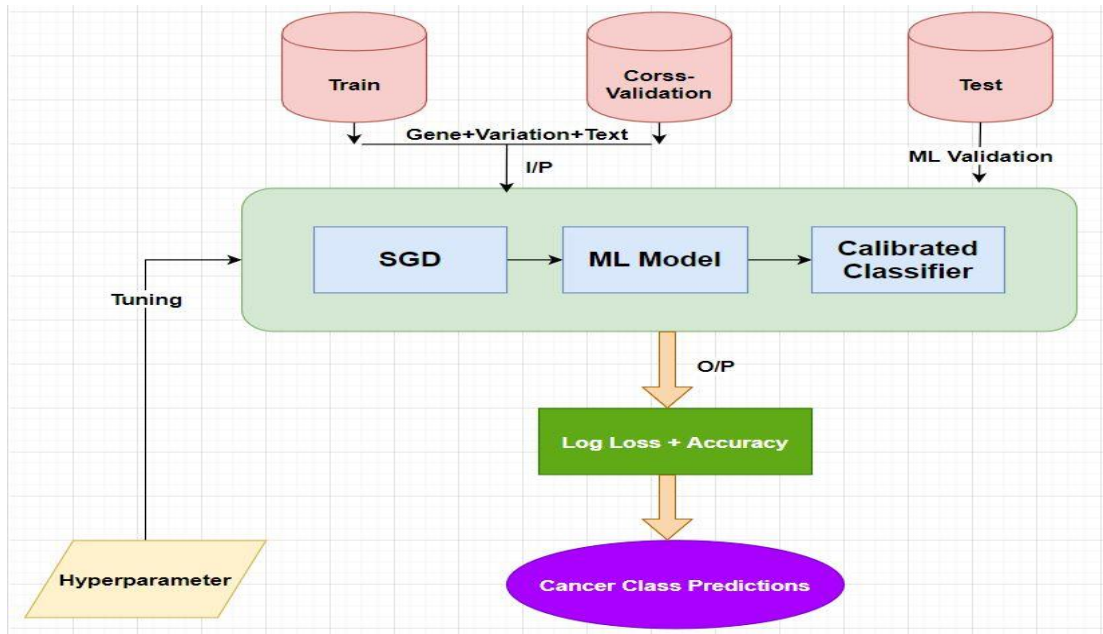
12. Building Machine Learning Models

12.1 Machine Learning Framework

For our project, for every ML model that we are going to implement, the following is the common framework that is used:

1. Using hyperparameters for tuning the algorithms.
2. Implementing Stochastic Gradient Descent (SGD) and Calibrated Classifier, along with our ML model for predicting the classes.

Below is the visual diagram to illustrate the general framework for each model being implemented and the descriptions of each classifier:



12.2 Data Preparation for ML Models

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

```
In [85]: ► 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

```
In [86]: # This function plots the confusion matrices given y_i, y_i_hat.
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')
```

Confusion Matrix Function

```
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 we will provide the array of probabilities 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

```
In [87]: # this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check 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'])
```

Some part of feature function to be used for Naïve Bayes

12.3 Combining The Features

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

According to one-hot encoding, the features are:

1. Train (2124, 55007)
2. Test (665, 55007)
3. Cross-Validation (532, 55007)

```
In [89]: > 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, 55007)
(number of data points * number of features) in test data = (665, 55007)
(number of data points * number of features) in cross validation data = (532, 55007)

One-hot encoding features

According to Response encoding, the features are:

4. Train (2124, 27)
5. Test (665, 27)
6. Cross-Validation (532, 27)

```
In [90]: > 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.

12.4 Machine Learning Models Implemented

12.4.1 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.

The formula for Multinomial Naïve Bayes:

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

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

The 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.

In our case, the Naïve Bayes algorithm works well with high dimensionality data. So we have implemented Naïve Bayes with one-hot encoding features and computed the train, cross-validation, and test log loss along with the accuracy.

The best alpha value for Naïve Bayes Model is 0.0001, which produces the train, cross-validation, and test log-loss values as 0.84, 1.24, 1.25, 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 37.78%, which means that the model predicts 62.22% points correctly. The confusion, precision, and recall matrix support these results. From the output, we can observe that Naïve Bayes with a one-hot encoding model performed well as the log loss computed were less than those of the random model. The difference between the train, cross-validation, and test log loss is not much, which shows that overfitting is not happening.

```
In [93]: 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=c
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,
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

For values of best alpha = 0.0001 The train log loss is: 0.840306988767
For values of best alpha = 0.0001 The cross validation log loss is: 1.2403382721
For values of best alpha = 0.0001 The test log loss is: 1.25731298028
```

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


```
In [94]: 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 probabilities we use log-probability estimates
print("Log Loss :", log_loss(cv_y, sig_clf_probs))
print("Number of misclassified point :", 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 : 1.2403382721
Number of misclassified point : 0.37781954887218044
----- Confusion matrix -----
```

Log Loss for the model using test data and misclassified points using Multinomial Naïve Bayes.

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

```
In [95]: 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], tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0863  0.0802  0.0108  0.1088  0.0418  0.0393  0.6242  0.0048  0.0038]]
Actual Class : 7
```

```
-----
15 Text feature [kinase] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
18 Text feature [inhibitor] present in test data point [True]
19 Text feature [activating] present in test data point [True]
20 Text feature [activation] present in test data point [True]
21 Text feature [presence] present in test data point [True]
22 Text feature [well] present in test data point [True]
23 Text feature [potential] present in test data point [True]
```

```
In [96]: 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], tes
```

```
Predicted Class : 6
Predicted Class Probabilities: [[ 0.0818  0.0756  0.0102  0.2831  0.04  0.4004  0.1006  0.0044  0.0038]]
Actual Class : 4
```

```
-----
10 Text feature [brca] present in test data point [True]
13 Text feature [history] present in test data point [True]
Out of the top 100 features 2 are present in query point
```

12.4.2 K- Nearest Neighbours:

It is one of the simplest (in understanding) algorithms used in Machine Learning for regression and classification problems. KNN algorithms uses the input data and later classifies new data points through similarity measures. In K-NN the classification is done by a majority vote to its neighbors. When KNN model is used in a classification problem, the output can be known from the class that is having highest frequency from the K-most neighbouring instances. Each instance, it votes for their class, and the class with the maximum votes is taken as the k nearest prediction. Since KNN does not work well with high dimensionality data, we are going to implement KNN with response encoding features for our classification.

The reason is the order of complexity for running this algorithm is :

$KNN = O(nd)$ Where, N = Total number of the data points

D = Total number of dimensions in the data-set

So if we go using KNN with one-hot encoding, then the dimensionality will be huge will, in turn, increases the running time complexity and also the space required to store in the memory. Here the hyperparameter that we are passing is the values for K, i.e., 5, 11, 15, etc. what should be the best nearest neighbor for which it gives us the optimal outcome. After executing the ML model, we found that the optimal value for K comes out to be 11.

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

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 11, which produces the train, cross-validation, and test log-loss values as 0.65, 1.01, 1.07, respectively. When the KNN Model is tested on the testing data using the best alpha, the log-loss for the model was generated as 1.01, and misclassified points were 33.83%, which means that the model predicts 66.17% points correctly. The confusion, precision, and recall matrix support these results.

```
In [99]: > 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.
> 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, lab
> 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.cl

For values of best alpha = 11 The train log loss is: 0.654238319024
For values of best alpha = 11 The cross validation log loss is: 1.0116193704
For values of best alpha = 11 The test log loss is: 1.07839089152
```

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

```
In [100]: > 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 : 1.0116193704

Number of mis-classified points : 0.3383458646616541

----- Confusion matrix -----

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

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

```
In [101]: # Lets Look at few test 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 = 1
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("Frequency of nearest points :", Counter(train_y[neighbors[1][0]]))

Predicted Class : 2
Actual Class : 7
The 11 nearest neighbours of the test points belongs to classes [7 2 7 7 7 7 7 7 7 7]
Frequency of nearest points : Counter({7: 10, 2: 1})
```

```
In [102]: 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 = 100

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[neighbors[1][0]])
print("Frequency of nearest points :", Counter(train_y[neighbors[1][0]]))

Predicted Class : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [6 4 4 1 4 4 4 4 1 4]
Frequency of nearest points : Counter({4: 8, 1: 2, 6: 1})
```

12.4.3 Logistic Regression:

Logistic Regression is a Machine Learning algorithm that is used to predict the probability of a categorical dependent variable or target variable given in a problem. In logistic regression, the dependent or the target variable is a binary variable that contains data value as 1 (yes, success, etc.) or 0 (no, failure, etc.). The LR 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 LR 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 for 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 0.61, 1.10, 1.06, 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 35.34%, which means that the model predicts 64.66% points correctly. The confusion, precision, and recall matrix support these results.

```
In [105]: 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))
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))
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))
```

For values of best alpha = 0.001 The train log loss is: 0.617073292064
For values of best alpha = 0.001 The cross validation log loss is: 1.10013828147
For values of best alpha = 0.001 The test log loss is: 1.06706518612

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

```
In [106]: 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 : 1.10013828147
Number of mis-classified points : 0.3533834586466165
----- Confusion matrix -----

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

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

```
In [108]: # from tabulate import tabulate
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_

Predicted Class : 7
Predicted Class Probabilities: [[ 0.019  0.0289  0.0021  0.0486  0.005  0.0018  0.8762  0.0159  0.0026]]
Actual Class : 7
-----
35 Text feature [activated] present in test data point [True]
49 Text feature [transforming] present in test data point [True]
61 Text feature [oncogene] present in test data point [True]
66 Text feature [cylinders] present in test data point [True]
76 Text feature [transformation] present in test data point [True]
83 Text feature [extracellular] present in test data point [True]
90 Text feature [downstream] present in test data point [True]
149 Text feature [resnitzky] present in test data point [True]
151 Text feature [activation] present in test data point [True]
157 Text feature [infect] present in test data point [True]

In [109]: 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_

Predicted Class : 6
Predicted Class Probabilities: [[ 0.0307  0.0112  0.0023  0.4229  0.0082  0.4827  0.0314  0.0082  0.0024]]
Actual Class : 4
-----
114 Text feature [motors] present in test data point [True]
139 Text feature [unwound] present in test data point [True]
161 Text feature [unwind] present in test data point [True]
167 Text feature [m299i] present in test data point [True]
187 Text feature [duplex] present in test data point [True]
216 Text feature [zealand] present in test data point [True]
243 Text feature [hospitals] present in test data point [True]
413 Text feature [unwinding] present in test data point [True]
443 Text feature [unwinds] present in test data point [True]
Out of the top 500 features 9 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 0.61, 1.09, 1.07, 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 34.77%, which means that the model predicts 65.23% points correctly. The confusion, precision, and recall matrix support these results.

```
In [114]: > 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))
           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))
           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))

For values of best alpha = 0.001 The train log loss is: 0.612225474329
For values of best alpha = 0.001 The cross validation log loss is: 1.09218683253
For values of best alpha = 0.001 The test log loss is: 1.07362382408
```

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

```
In [115]: > 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 : 1.09218683253
Number of mis-classified points : 0.34774436090225563
----- Confusion matrix -----
```

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

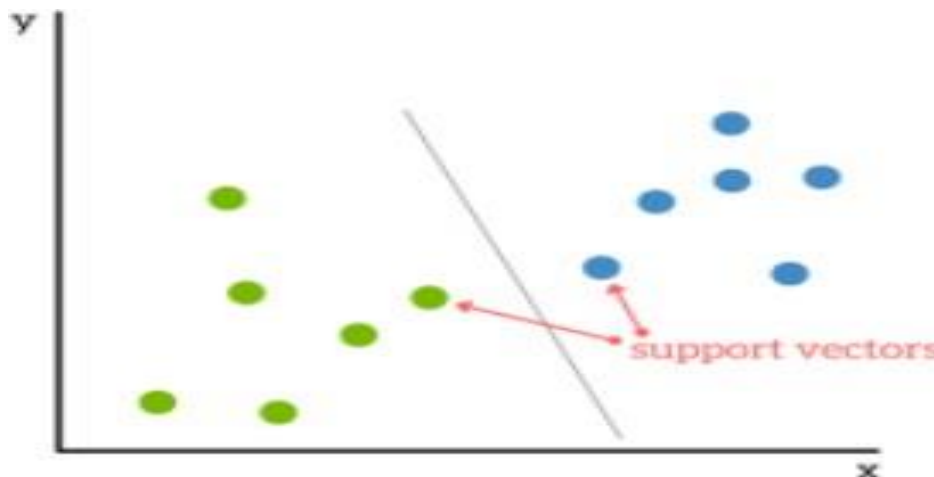
We also tested, the model using one random data points.

```
In [117]: 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],te

Predicted Class : 7
Predicted Class Probabilities: [[ 3.28000000e-02  2.65000000e-02  4.00000000e-04  5.59000000e-02
 2.40000000e-03  1.30000000e-03  8.64200000e-01  1.65000000e-02
 0.00000000e+00]]
Actual Class : 7
-----
75 Text feature [ht] present in test data point [True]
127 Text feature [transforming] present in test data point [True]
139 Text feature [transformation] present in test data point [True]
145 Text feature [activated] present in test data point [True]
173 Text feature [extracellular] present in test data point [True]
187 Text feature [oncogene] present in test data point [True]
207 Text feature [downstream] present in test data point [True]
234 Text feature [cylinders] present in test data point [True]
245 Text feature [transformed] present in test data point [True]
```

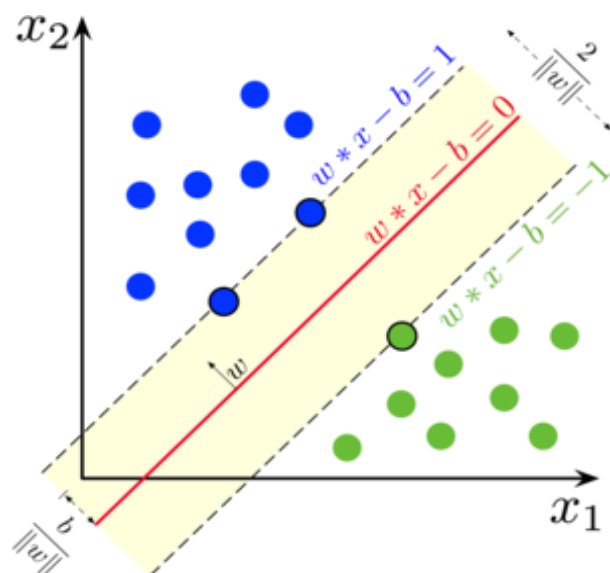
12.4.4 Linear Support Vector Machine:

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



Support vectors formed are the data points that are nearest to the hyperplane, i.e., the points of the data set, and if they are removed, they would alter the position of the dividing or split hyperplane. Hence due to 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.

We are not going to implement RBF SVM as we don't know the kernel here in this machine learning problem, and also RBF is not that interpretable compared to the Linear

SVM. Since Linear SVM is similar to logistic regression, for our project, we are going to implement it using one-hot encoding since it performs well with the high dimensionality of data.

The best C/alpha value for Linear SVM Model is 0.01, which produces the train, cross-validation, and test log-loss values as 0.74, 1.14, 1.14, 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 34.21%, which means that the model predicts 65.79% points correctly. The confusion, precision, and recall matrix support these results.

```
In [119]: alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
    # clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
    clf = SGDClassifier(class_weight='balanced', alpha=i, 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)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))
```

For values of best alpha = 0.01 The train log loss is: 0.74082256438

For values of best alpha = 0.01 The cross validation log loss is: 1.13584927473

For values of best alpha = 0.01 The test log loss is: 1.14042661431

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

```
In [120]: 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 : 1.13584927473
Number of mis-classified points : 0.34210526315789475
----- Confusion matrix -----
```

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

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

```
In [121]: 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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 6.19000000e-02  2.07000000e-02  2.10000000e-03  1.06700000e-01
  6.90000000e-03  1.06000000e-02  7.83800000e-01  6.60000000e-03
  7.00000000e-04]]
Actual Class : 7
```

12.4.5 Random Forest Classifier

Random forests are a supervised machine learning algorithm that can be implemented both for classification and regression problems. Random forest comprised of trees, and the more trees it has, the more robust it is. Random forests create decision trees on randomly selected data samples, get a prediction from each of the trees, and selects the best solution using voting. It also provides a pretty good indicator of the feature importance. Random Forest Classifier is an ensemble algorithm. Ensemble algorithms are those which combine more than one algorithms of the 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 critical term Bagging directs to the ensemble, which means that a group of things viewed as a whole. 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,

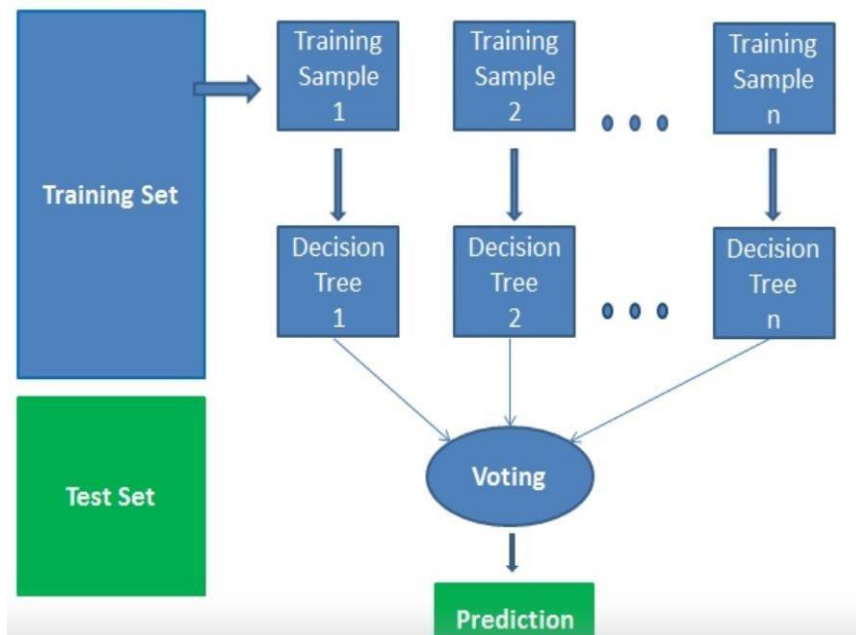
[X1, X2, X3]

[X1, X2, X4]

[X2, X3, X4]

So finally, it tries to make predictions 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. A tree with a high error rate is given low weight value and vis-a-vis. This would increase the decision impact of trees with a flat error rate. It is technically considered as 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 created 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 more straightforward and more powerful compared to the other non-linear classification algorithms.



Random Forest Classifier with Maximum voting classifier

Random forest uses gene importance or means a decrease in impurity (MDI) to calculate the significance of each of the distinct features. Gini index 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 more substantial the reduction, the more significant the variable is. Here, the mean decrease is a vital parameter for variable selection. The Gini index can describe the overall explanatory power of the variables.

The RFC is implemented using both a one-hot encoder and a 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 200, 10, respectively, which produces the train, cross-validation, and test log-loss values as 0.71, 1.14, 1.12, 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 1.14 and misclassified points were 38.35%, which means that the model predicts 61.65% points correctly. The confusion, precision, and recall matrix support these results.

```
In [123]: 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)])
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))
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))
```

```
For values of best estimator = 200 The train log loss is: 0.710284280638
For values of best estimator = 200 The cross validation log loss is: 1.14070205202
For values of best estimator = 200 The test log loss is: 1.1236273535
```

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

```
In [124]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha/2)])
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.14070205202
Number of mis-classified points : 0.38345864661654133
----- Confusion matrix -----
```

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.

```
In [126]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_a
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_in

Predicted Class : 7
Predicted Class Probabilities: [[ 0.1015  0.0948  0.0148  0.0939  0.0418  0.0355  0.6048  0.0069  0.0061]]
Actual Class : 7
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activated] present in test data point [True]
2 Text feature [activating] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [kinases] 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, 1.28, 1.30, 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 1.28, and misclassified points were 48.68% which means that the model predicts 51.32% points correctly. The confusion, precision, and recall matrix support these results.


```
In [127]: alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

For values of best alpha = 100 The train log loss is: 0.051190149414
For values of best alpha = 100 The cross validation log loss is: 1.28392384765
For values of best alpha = 100 The test log loss is: 1.30082160359
```

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

response encoder

```
In [128]: clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha/4)], n_estimators=alpha[int(best_alpha/4)], criterion='gini',
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)

Log loss : 1.28392384765
Number of mis-classified points : 0.4868421052631579
----- Confusion matrix -----
```

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

encoder

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

```
In [129]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha/4])
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")

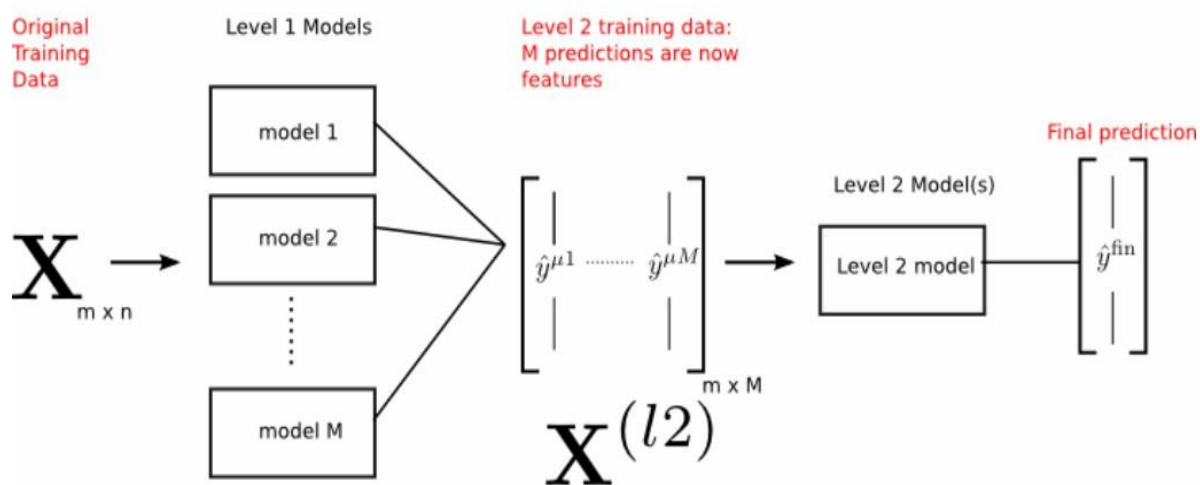
Predicted Class : 7
Predicted Class Probabilities: [[ 2.00000000e-03  2.44000000e-02  1.70000000e-03  3.00000000e-03
  9.00000000e-04  3.10000000e-03  9.61800000e-01  1.40000000e-03
  1.70000000e-03]]
Actual Class : 7
-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
```

From the result , we can see that since there is a huge difference between the training log loss and cross-validation and testing log loss , this indicates that there is a huge overfitting happening and we can reject this ML model for our analysis. This above shows that random forest is not performing good for our analysis for both one-hot encoding and response encoding.

12.4.6 Stacking Model

A stacking method is one of the ensemble algorithms where a new model is trained in such a way that it combines the predictions from two or more models that are already trained on our dataset. The predictions calculated from those existing models or submodels are combined using a new model as blending.

It is typically 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 connected must-have skill on the problem, but do not need to be the best possible models. This ensures we don't 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)

The stacking model works well when we have a massive amount of data, but since we don't have that broad data, it may not perform well as expected. 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 1.07, 1.69, 1.24, respectively, for these models. Therefore, the stacking classifier is generated using the Logistic Regression Model, and the minimum log-loss 1.07 is obtained at $\alpha = 0.1$.

```
In [130]: 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: %.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: %.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: %.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=1)
              sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
              sclf.fit(train_x_onehotCoding, train_y)
              print("Stacking Classifier : for the value of alpha: %f Log Loss: %.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
              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: 1.07
Support vector machines : Log Loss: 1.69
Naive Bayes : Log Loss: 1.24
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.041
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.527
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.114
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.172
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.404
```

Log loss results

```

In [131]: lr = LogisticRegression(C=0.1)
          scf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probab=True)
          scf.fit(train_x_onehotCoding, train_y)

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

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

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

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

          Log loss (train) on the stacking classifier : 0.662940243923
          Log loss (CV) on the stacking classifier : 1.11432017542
          Log loss (test) on the stacking classifier : 1.11594191877
          Number of missclassified point : 0.35789473684210527
          ----- Confusion matrix -----

```

Final results for Stacking Classifier

12.4.7 Maximum Voting Classifier:

Voting Classifier is one of the techniques of combining the predictions obtained from multiple machine learning algorithms and computing the best one for the problem.

It operates by first forming two or more standalone models of our training dataset. A Voting Classifier can then be used to wrap our models and equalize the predictions of the sub-models when required to make predictions for new data. We can devise 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, 1.19, 1.20 respectively. The misclassified points were 35.64%, which means that the model predicts 64.36% points correctly. The confusion, precision, and recall matrix support these results.

Log Loss Results for Stacking Classifier

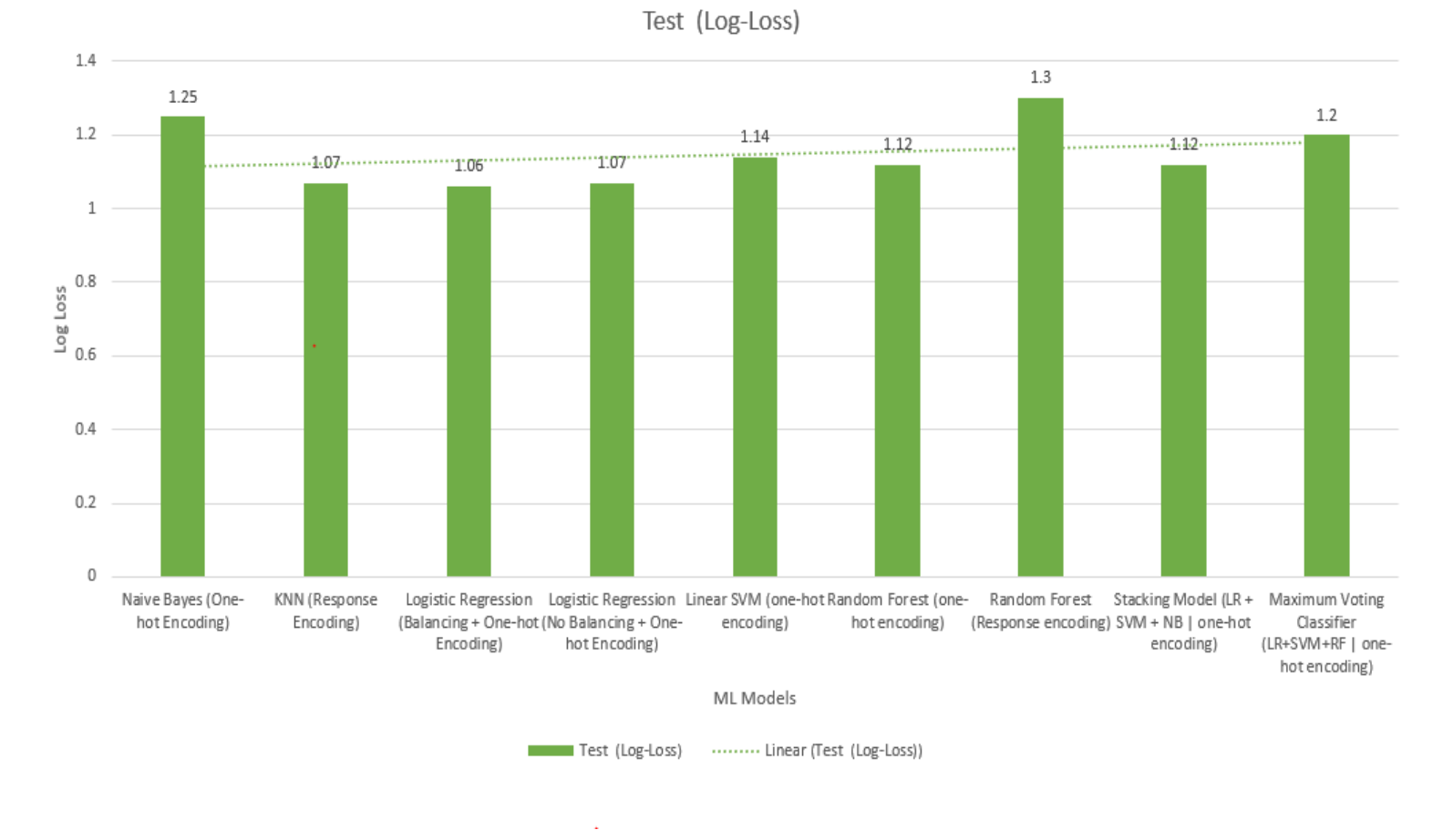
```
In [132]: #http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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 misclassified 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 : 0.916449056762
Log loss (CV) on the VotingClassifier : 1.1858758572
Log loss (test) on the VotingClassifier : 1.20235899773
Number of misclassified point : 0.35639097744360904
----- Confusion matrix -----
```

13. RESULTS:

S.NO	ML Model	Train (Log-Loss)	Cross- Validation (Log-Loss)	Test (Log-Loss)	Misclassification %	Accuracy%
1	Naive Bayes (One-hot Encoding)	0.84	1.24	1.25	37.78%	62.22%
2	KNN (Response Encoding)	0.65	1.01	1.07	33.83%	66.17%
3	Logistic Regression (Balancing + One-hot Encoding)	0.61	1.1	1.06	35.34%	64.66%
4	Logistic Regression (No Balancing + One-hot Encoding)	0.61	1.09	1.07	34.77%	65.23%
5	Linear SVM (one-hot encoding)	0.74	1.14	1.14	34.21%	65.79%
6	Random Forest (one-hot encoding)	0.71	1.14	1.12	38.35%	61.65%

7	Random Forest (Response encoding)	0.05	1.28	1.3	48.68%	51.32%
8	Stacking Model (LR + SVM + NB one-hot encoding)	0.66	1.11	1.12	35.79%	64.21%
9	Maximum Voting Classifier (LR+SVM+RF one-hot encoding)	0.92	1.19	1.2	35.64%	64.36%



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

14. Conclusion:

It is found that a well-curated characterized gene database can promote accurate phenotype-driven gene analysis in the medical domain. It enables classification as to what

the new studies explored, giving patients with a diagnosis and opportunities for therapeutic benefits, and an end to their “diagnostic odyssey.” Across time and with regular literature review and clinical legality curation, we expect that many patients with candidate genetic etiologies will obtain a definitive diagnosis via reclassification reports. Our machine 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 accurately guide reporting decisions at the critical 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 the diagnostic rate.

In this project, we have implemented various machine learning models using the combination of one-hot encoding and response encoding and computed the log loss and accuracy on the training, cross-validation, and testing data. For model validation, we will consider the testing data, and based on our evaluation criteria, i.e., log loss and accuracy. We have to decide which model performs excellently and which we should recommend for our machine learning problem. If our business requirement states log loss to be the evaluation criteria, then we are going for logistic regression with balancing – one hot encoding since it gives the lowest testing log loss, i.e., 1.06 compared to the other ML models.

Similarly, if the requirement is based in terms of model accuracy, we will go with KNN with response encoding since it gives minimal misclassification, i.e., approx. 33.83 % ~ approx.66.17% accuracy compared to other ML models.

So for our case, we would recommend going with logistic regression since it works well with high data dimensionality compared to KNN(which gives the overfitted results) and for the problem where we have textual data, it is recommended to use such ML models which are capable to handling enormous data and compute the predictions based on the conditions.

15. Future Scope:

We have only used a text analysis method like Bag of words to further convert the categorical data into an appropriate format for our machine learning algorithm. In the future, we would like to explore more on the text analysis methods like TF-IDF, Word2Vec, and other advanced NLP methods, including neural networks.

We would also like to implement other models, including XGBoost and RBF SVM. However, our statistical results on the datasets reveal the problem of an unbalanced distribution of each class in both training set and testing set. Hence, we need to experiment with a variety of classification models along with text analysis methods to ensure which combination works well and produce the optimal result.

Further work also relies on finding more effective links between the genes, variation, and text proof as well as use more powerful methods to extract the information from the text database.

16. Reference And Sources

- [1] Personalized Medicine: Redefining Cancer Treatment. (n.d.). Retrieved from <https://www.kaggle.com/c/msk-redefining-cancer-treatment/overview>
- [2] Critelli, K. (2018, March 5). Redefining Cancer Treatment: Predicting Gene Mutations to Advance Personalized Medicine. Retrieved March 8, 2020, from <https://nycdatascience.com/blog/student-works/redefining-cancer-treatment-predicting-gene-mutations-advance-personalized-medicine/>
- [3] Mendes, E. (2015, April 3). Personalized Medicine: Redefining Cancer and Its Treatment. Retrieved March 8, 2020, from <https://www.cancer.org/latest-news/personalized-medicine-redefining-cancer-and-its-treatment.html>
- [4] Personalized Medicine. (n.d.). Retrieved March 8, 2020, from <https://moffitt.org/treatments/personalized-medicine/>
- [5] Malik, U. (2019, February 17). Text Classification with Python and Scikit-Learn. Retrieved from <https://stackabuse.com/text-classification-with-python-and-scikit-learn/>
- [6] Shaikh, J. (2017, June 23). Machine Learning, NLP: Text Classification using scikit-learn, python and NLTK. Retrieved from <https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a>
- [7] Python 3 Tutorial - Tutorialspoint. (n.d.). Retrieved March 8, 2020, from <https://www.tutorialspoint.com/python3/index.htm>
- [8] M. Verma, "Personalized Medicine and Cancer," Journal of Personalized Medicine, vol. 2, no. 1, pp. 1-14, 2012.

- [9] A. Holzinger, "Trends in interactive knowledge discovery for personalized medicine: cognitive science meets machine learning," IEEE Intell Inform Bull, vol. 15, no.1, pp. 6-14, 2014.
- [10] M. W. Libbrecht and W. S. Noble. (2015, Mar.). Machine learning applications in genetics and genomics. Nature Reviews Genetics. [Online]. 16(6), pp. 321-332. Available: <https://doi.org/10.1038/nrg3920>
- [11] A. Bhola and A. K. Tiwari, "Machine Learning Based Approaches for Cancer Classification Using Gene Expression Data", Machine Learning and Applications: An International Journal (MLAIJ), Vol. 2, No. 3/4, Dec. 2015.
- [12] Samuel, Arthur (1959). "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development. 3 (3): 210–229. doi:10.1147/rd.33.0210.
- [13] "Machine Learning: What it is and why it matters". www.sas.com.
- [14] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer. (2002, June). "Smote: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research. [Online]. 16, pp. 321-357. Available: <https://doi.org/10.1613/jair.953>
- [15] Patel, Savan. "Chapter 2: SVM (Support Vector Machine) — Theory." [Medium, Machine Learning 101, 3 May 2017, medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72](https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72).
- [16] "Personalized Medicine: Redefining Cancer Treatment." Kaggle.Com, www.kaggle.com/c/msk-redefining-cancer-treatment/data. Accessed 10 May 2020.

- [17] Wikipedia Contributors. "K-Nearest Neighbors Algorithm." Wikipedia, Wikimedia Foundation, 19 Mar. 2019, en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.
- [18]"Logistic Regression." Wikipedia, Wikimedia Foundation, 12 Apr. 2019, en.wikipedia.org/wiki/Logistic_regression.
- [19]"Naive Bayes Classifier." Wikipedia, Wikimedia Foundation, 17 June 2019, en.wikipedia.org/wiki/Naive_Bayes_classifier.
- [20]"Random Forest." Wikipedia, Wikimedia Foundation, 9 Apr. 2019, en.wikipedia.org/wiki/Random_forest.
- [21]"Support-Vector Machine." Wikipedia, Wikimedia Foundation, 14 May 2019, en.wikipedia.org/wiki/Support-vector_machine.
- [22]Yaghoobzadeh, Yadollah, and Hinrich Schütze. "Intrinsic Subspace Evaluation of Word Embedding Representations." ArXiv:1606.07902 [Cs], 25 June 2016, arxiv.org/abs/1606.07902. Accessed 10 May 2020.
- [23]Baker, Simon, et al. Cancer Hallmark Text Classification Using Convolutional Neural Networks Introduction and Motivation.
- [24]Mikolov, Tomas, et al. "Efficient Estimation of Word Representations in Vector Space." ArXiv.Org, 2013, arxiv.org/abs/1301.3781.