

Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/>

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462>

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>
2. <https://www.youtube.com/watch?v=UwbuW7oK8rk>
3. <https://www.youtube.com/watch?v=qxXRKVompI8>

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/data>
- We have two data files: one contains the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files have a common column called ID
- Data file's information:
 - training_variants (ID, Gene, Variations, Class)
 - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text

0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome. Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation>

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilities => Metric is Log-loss.
- No latency constraints.

2.3. Train, CV and Test Datasets

Split the dataset randomly into three parts train, cross validation and test with 64%, 16%, 20% of data respectively

3. Exploratory Data Analysis

In [6]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [7]:

```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

Number of data points : 3321
Number of features : 4
Features : ['ID' 'Gene' 'Variation' 'Class']

Out[7]:

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

1	ID	Gene	W802*	Variation	Class
2	2	CBL	Q249E		2
3	3	CBL	N454D		3
4	4	CBL	L399V		4

training/training_variants is a comma separated file containing the description of the genetic mutations used for training.
Fields are

- **ID** : the id of the row used to link the mutation to the clinical evidence
- **Gene** : the gene where this genetic mutation is located
- **Variation** : the aminoacid change for this mutations
- **Class** : 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

In [8]:

```
# note the separator in this file
data_text = pd.read_csv("training_text", sep="\t", engine="python", names=["ID", "TEXT"], skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

Number of data points : 3321
Number of features : 2
Features : ['ID' 'TEXT']

Out[8]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3. Preprocessing of text

In [10]:

```
import nltk
nltk.download('stopwords')

[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ashis\AppData\Roaming\nltk_data...
[nltk_data]     Unzipping corpora\stopwords.zip.
```

Out[10]:

True

In [12]:

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))
```

```

def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+', ' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

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

        data_text[column][index] = string

```

In [13]:

```

#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:", index)
print('Time took for preprocessing the text :', time.clock() - start_time, "seconds")

```

there is no text description for id: 1109
 there is no text description for id: 1277
 there is no text description for id: 1407
 there is no text description for id: 1639
 there is no text description for id: 2755
 Time took for preprocessing the text : 231.6373561 seconds

In [14]:

```

#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text, on='ID', how='left')
result.head()

```

Out[14]:

	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 lineage...

In [15]:

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

Out[15]:

	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

In [16]:

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

In [17]:

```
result[result['ID']==1109]
```

Out[17]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

In [18]:

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

# split the data into test and train by maintaining same distribution of output variable 'y_true'
# [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2
, random_state=0)
# split the train data into train and cross validation by maintaining same distribution of output
# variable 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
, random_state=0)
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [19]:

```
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
```

3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [21]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
```

```

print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')

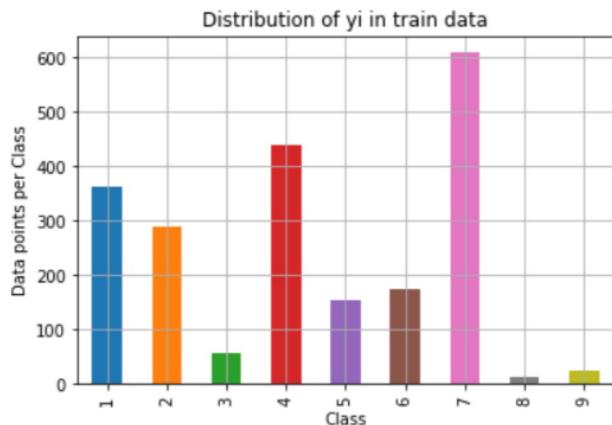
print('*'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

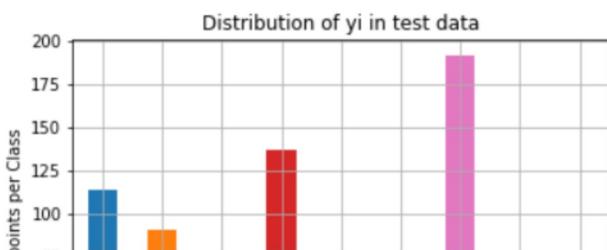
print('*'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

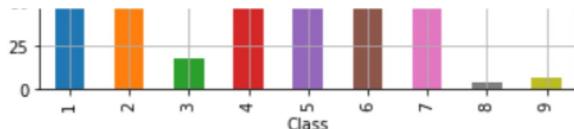
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')

```



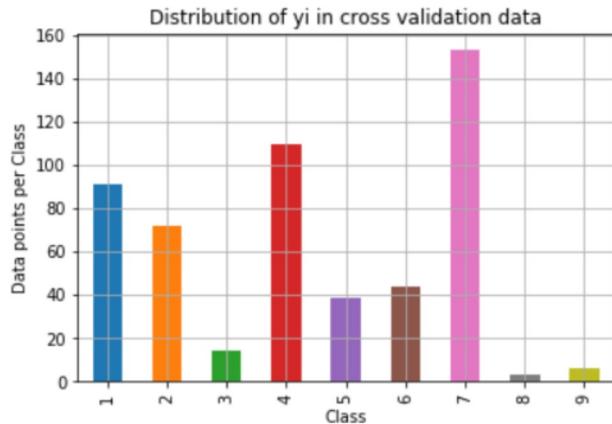
Number of data points in class 7 : 609 (28.672 %)
 Number of data points in class 4 : 439 (20.669 %)
 Number of data points in class 1 : 363 (17.09 %)
 Number of data points in class 2 : 289 (13.606 %)
 Number of data points in class 6 : 176 (8.286 %)
 Number of data points in class 5 : 155 (7.298 %)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





```

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



```

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

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilities randomly such that they sum to 1.

In [22]:

```

# 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)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A = ((C.T)/(C.sum(axis=1))).T
    # divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #      [2, 4]]
    # C.sum(axis = 1)  axis=0 corresponds to columns and axis=1 corresponds to rows in two
    # dimensional array
    # C.sum(axis =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
```

```

B = (C/C.sum(axis=0))
#divid each element of the confusion matrix with the sum of elements in that row
# C = [[1, 2],
#       [3, 4]]
# C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
#diamensional array
# C.sum(axis =0) = [[4, 6]
# (C/C.sum(axis=0)) = [[1/4, 2/6],
#                      [3/4, 4/6]]

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

print("-"*20, "Precision matrix (Column Sum=1)", "-"*20)
plt.figure(figsize=(20,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

# representing B in heatmap format
print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
plt.figure(figsize=(20,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

```

In [27]:

```

# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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))

# 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))

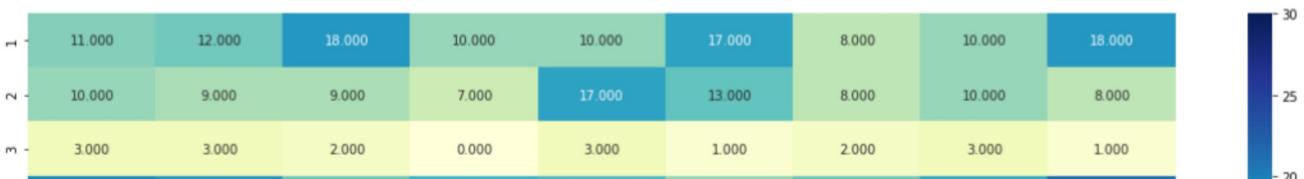
predicted_y = np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

Log loss on Cross Validation Data using Random Model 2.4352964192635302

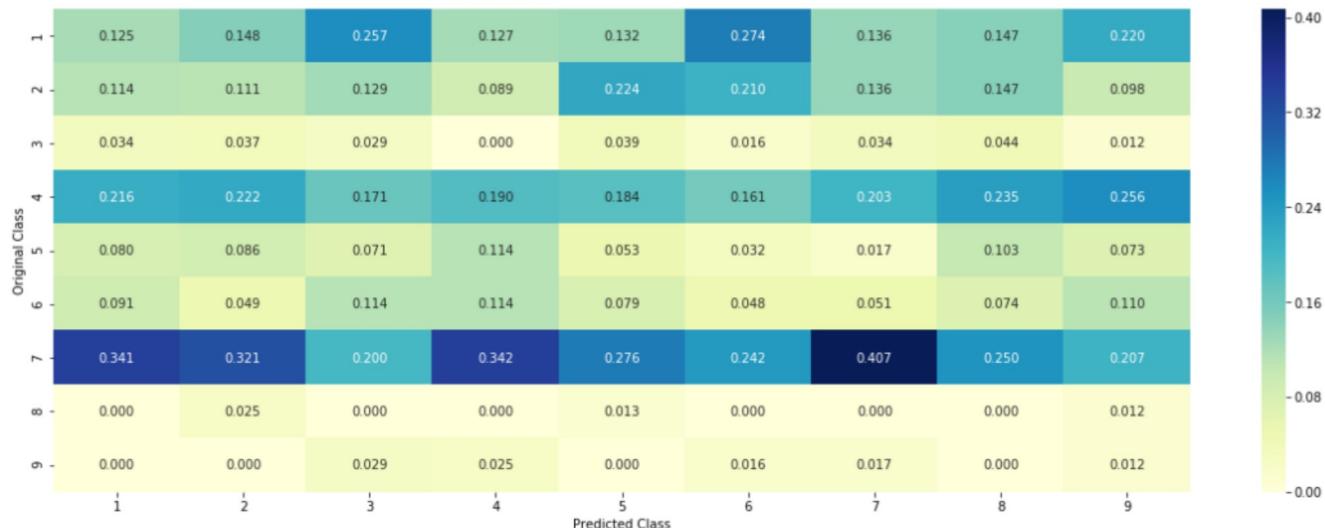
Log loss on Test Data using Random Model 2.500015188432678

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

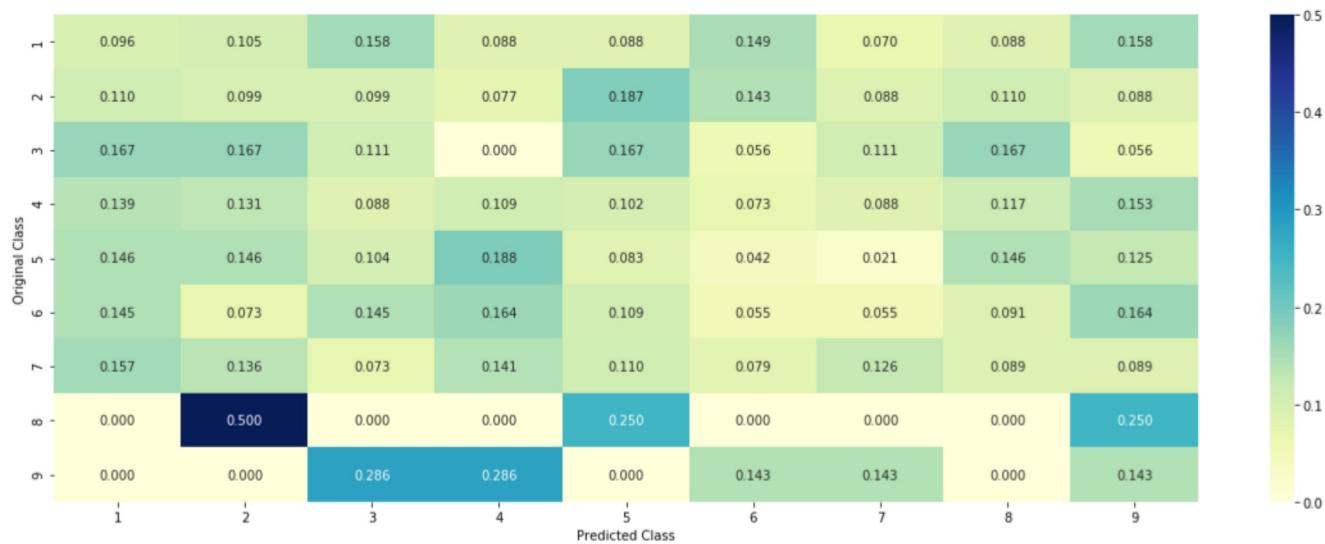




----- Precision matrix (Column Sum=1) -----



----- Recall matrix (Row sum=1) -----



3.3 Univariate Analysis

In [28]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
#
# Consider all unique values and the number of occurrences of given feature in train data dataframe
# build a vector (1*9) - the first element = (number of times it occurred in class1 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class2 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class3 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class4 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class5 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class6 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class7 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class8 + 10*alpha) / n
# build a vector (1*9) - the first element = (number of times it occurred in class9 + 10*alpha) / n
```

```

# gv_dict is like a look up table, for every gene it stores a 10 dimensional vector
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----

# get_gv_fea_dict: Get Gene variation Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #       {BRCA1      174
    #        TP53      106
    #        EGFR      86
    #        BRCA2      75
    #        PTEN      69
    #        KIT       61
    #        BRAF      60
    #        ERBB2      47
    #        PDGFRA     46
    #        ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations      63
    # Deletion                  43
    # Amplification             43
    # Fusions                   22
    # Overexpression            3
    # E17K                      3
    # Q61L                      3
    # S222D                     2
    # P130S                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occurred in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
        # vec is 9 dimensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #       ID   Gene           Variation  Class
            # 2470  2470  BRCA1          S1715C    1
            # 2486  2486  BRCA1          S1841R    1
            # 2614  2614  BRCA1          M1R       1
            # 2432  2432  BRCA1          L1657P    1
            # 2567  2567  BRCA1          T1685A    1
            # 2583  2583  BRCA1          E1660G    1
            # 2634  2634  BRCA1          W1718L    1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]

            # cls_cnt.shape[0] (numerator) will contain the number of time that particular feature occurred in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

            # we are adding the gene/variation to the dict as key and vec as value
            gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.200757575757575, 0.037878787878788, 0.06818181818177,
    0.136363636363635, 0.25, 0.193181818181818, 0.037878787878788, 0.037878787878788,
    0.0378787878787881]

```

```

0.270100100200001, 0.001221100100000, 0.00002000001100, 0.001020100100000, 0.001020100
163265307, 0.056122448979591837],
    # 'EGFR': [0.0568181818181816, 0.21590909090909091, 0.0625, 0.0681818181818177,
0.0681818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.0568181818181816],
    # 'BRCA2': [0.1333333333333333, 0.060606060606060608, 0.060606060606060608,
0.0787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    # 'BRAF': [0.06666666666666666, 0.17999999999999999, 0.07333333333333334,
0.0733333333333334, 0.0933333333333338, 0.080000000000000002, 0.29999999999999999,
0.06666666666666666, 0.06666666666666666],
    #
    ...
    #
    ]
gv_dict = get_gv_fea_dict(alpha, feature, df)
# value_count is similar in get_gv_fea_dict
value_count = train_df[feature].value_counts()

# gv_fea: Gene_variation feature, it will contain the feature for each feature value in the da
ta
gv_fea = []
# for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
# if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
for index, row in df.iterrows():
    if row[feature] in dict(value_count).keys():
        gv_fea.append(gv_dict[row[feature]])
    else:
        gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#
    gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1])
return gv_fea

```

when we calculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- $(\text{numerator} + 10\alpha) / (\text{denominator} + 90\alpha)$

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

In [29]:

```

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 : 226
BRCA1    169
TP53     101
EGFR     94
PTEN     85
BRCA2     77
KIT      59
BRAF     58
ERBB2     46
ALK      41
FGFR2     40
Name: Gene, dtype: int64

```

In [30]:

```

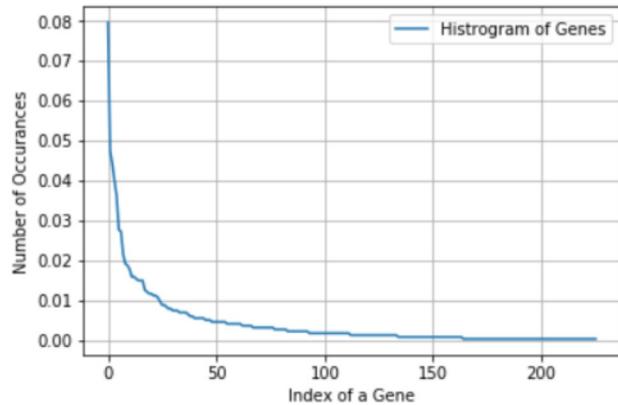
print("Ans: There are", unique_genes.shape[0], "different categories of genes in the train data. an

```

Ans: There are 226 different categories of genes in the train data, and they are distributed as follows

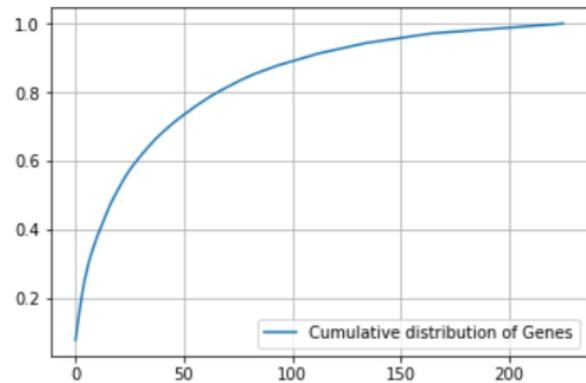
In [33]:

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurrences')
plt.legend()
plt.grid()
plt.show()
```



In [32]:

```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans. there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

In [34]:

```
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [36]:

```
print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature: (2124, 9)
```

In [37]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [38]:

```
train_df['Gene'].head()
```

Out[38]:

```
1703      PMS2
1284      HRAS
3208      RB1
2125      CCND1
476       TP53
Name: Gene, dtype: object
```

In [39]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 225)
```

Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i . One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i .

In [40]:

```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.
```

```

# video link:
#-----



cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, 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(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

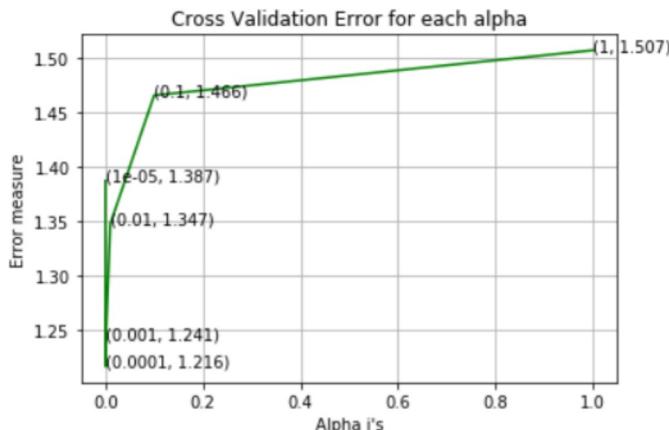
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.3870468447618116
 For values of alpha = 0.0001 The log loss is: 1.2159741721811521
 For values of alpha = 0.001 The log loss is: 1.2409499117374159
 For values of alpha = 0.01 The log loss is: 1.3471077827397941
 For values of alpha = 0.1 The log loss is: 1.4655295368185326
 For values of alpha = 1 The log loss is: 1.5070498610818235



For values of best alpha = 0.0001 The train log loss is: 1.0505172190473846
 For values of best alpha = 0.0001 The cross validation log loss is: 1.2159741721811521
 For values of best alpha = 0.0001 The test log loss is: 1.1824833977922369

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

In [41]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in train dataset?")  
  
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]  
  
print('Ans\n1. 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)
```

Q6. How many data points in Test and CV datasets are covered by the 226 genes in train dataset?
Ans

1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 508 out of 532 : 95.48872180451127

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [42]:

```
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 : 1929  
Truncating_Mutations      55  
Amplification            48  
Deletion                 44  
Fusions                  27  
G12V                     3  
T73I                     2  
S308A                     2  
Y42C                     2  
F384L                     2  
S222D                     2  
Name: Variation, dtype: int64
```

In [43]:

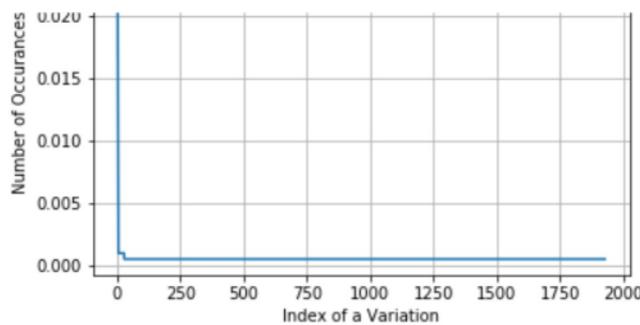
```
print("Ans: There are", unique_variations.shape[0], "different categories of variations in the  
train data, and they are distributed as follows")
```

Ans: There are 1929 different categories of variations in the train data, and they are distributed as follows

In [44]:

```
s = sum(unique_variations.values);  
h = unique_variations.values/s;  
plt.plot(h, label="Histogram of Variations")  
plt.xlabel('Index of a Variation')  
plt.ylabel('Number of Occurrences')  
plt.legend()  
plt.grid()  
plt.show()
```

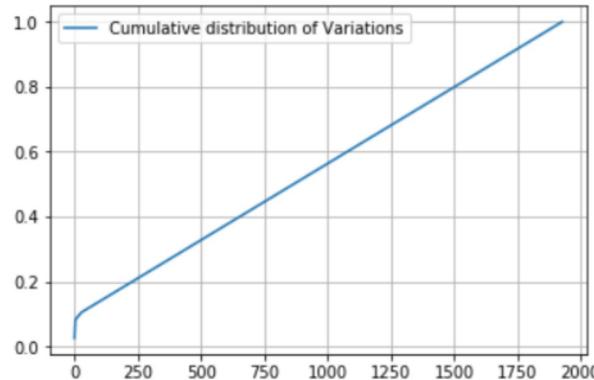




In [45]:

```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

[0.02589454 0.04849341 0.06920904 ... 0.99905838 0.99952919 1.]



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

We will be using both these methods to featurize the Variation Feature

In [46]:

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [47]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

train_variation_feature_responseCoding is a converted feature using the response coding method. Th
e shape of Variation feature: (2124, 9)

In [48]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [49]:

```
print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 1956)
```

Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [51]:

```
alpha = [10 ** x for x in range(-5, 1)]
```

read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

```
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
```

some of methods

```
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.
```

#-----
video link:
#-----

```
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, 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(cv_variation_feature_onehotCoding)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
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")
```

```

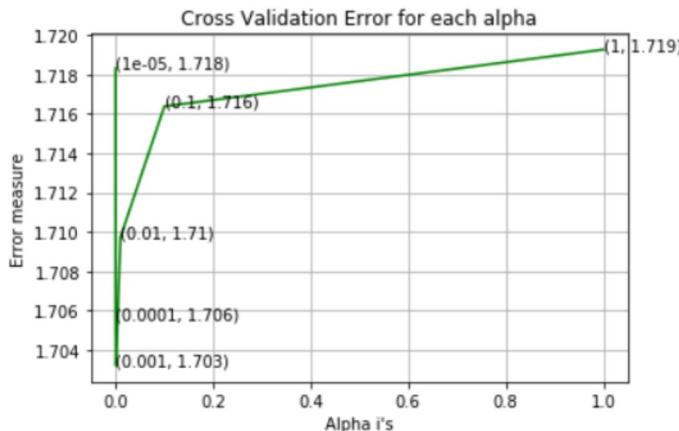
predict_y = sig_cir.predict_proba(train_variation_feature_onenotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
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, p
redict_y, labels=clf.classes_, eps=1e-15))

```

```

For values of alpha =  1e-05 The log loss is: 1.7183183219047198
For values of alpha =  0.0001 The log loss is: 1.7055971679720234
For values of alpha =  0.001 The log loss is: 1.7031780233150433
For values of alpha =  0.01 The log loss is: 1.7097248043870965
For values of alpha =  0.1 The log loss is: 1.7163869593445384
For values of alpha =  1 The log loss is: 1.7192762227537035

```



```

For values of best alpha =  0.001 The train log loss is: 1.0688038793149857
For values of best alpha =  0.001 The cross validation log loss is: 1.7031780233150433
For values of best alpha =  0.001 The test log loss is: 1.708692905946978

```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Not sure! But lets be very sure using the below analysis.

In [54]:

```

print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
st and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. 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.s
hape[0])*100)

```

Q12. How many data points are covered by total 1929 genes in test and cross validation data sets?

Ans

1. In test data 76 out of 665 : 11.428571428571429
2. In cross validation data 51 out of 532 : 9.586466165413533

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicting y_i ?
5. Is the text feature stable across train, test and CV datasets?

```

# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary

```

In [56]:

```

import math
#https://stackoverflow.com/a/1602964
def get_text_responseCoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding

```

In [57]:

```

# 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_textfea_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_textfea_counts))

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

```

Total number of unique words in train data : 52865

In [58]:

```

dict_list = []
# dict_list =[] contains 9 dictionaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th class text data
# total_dict is build on whole training text data
total_dict = extract_dictionary_paddle(train_df)

confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)

```

In [59]:

```
#response coding of text features
train_text_feature_responseCoding = get_text_responsecoding(train_df)
test_text_feature_responseCoding = get_text_responsecoding(test_df)
cv_text_feature_responseCoding = get_text_responsecoding(cv_df)
```

In [60]:

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding =
(test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [61]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

In [62]:

```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [63]:

```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 4883, 4: 3770, 5: 3157, 6: 3015, 9: 2120, 8: 1806, 7: 1682, 10: 1473, 12: 1390, 11: 108, 15: 979, 14: 859, 13: 810, 18: 795, 16: 689, 17: 627, 20: 559, 22: 528, 24: 484, 19: 440, 21: 438, 27: 430, 30: 396, 25: 370, 23: 363, 28: 346, 38: 342, 29: 307, 31: 306, 26: 306, 35: 293, 36: 289, 34: 268, 46: 265, 32: 257, 33: 236, 40: 228, 42: 217, 37: 213, 39: 200, 45: 198, 48: 190, 47: 176, 41: 171, 44: 169, 51: 165, 56: 164, 60: 161, 43: 160, 54: 158, 52: 153, 49: 152, 50: 149, 57: 136, 53: 131, 55: 129, 63: 127, 59: 121, 68: 116, 66: 116, 58: 113, 61: 112, 70: 110, 69: 108, 64: 104, 67: 100, 84: 99, 72: 92, 62: 88, 76: 87, 65: 87, 78: 86, 92: 85, 75: 83, 87: 82, 80: 82, 77: 80, 71: 78, 107: 74, 79: 73, 73: 73, 90: 72, 89: 72, 91: 67, 82: 66, 81: 65, 74: 65, 106: 62, 103: 62, 83: 62, 105: 61, 100: 61, 85: 61, 98: 60, 102: 58, 88: 56, 86: 55, 120: 53, 113: 53, 95: 53, 93: 53, 110: 52, 97: 52, 112: 51, 96: 51, 108: 50, 115: 48, 94: 48, 121: 47, 126: 46, 109: 45, 150: 44, 130: 44, 124: 44, 118: 44, 101: 43, 111: 42, 99: 42, 138: 41, 149: 40, 122: 40, 128: 39, 104: 39, 123: 38, 119: 38, 116: 38, 152: 37, 160: 36, 142: 36, 133: 36, 129: 36, 127: 36, 148: 35, 146: 34, 135: 34, 132: 34, 163: 33, 144: 33, 141: 33, 166: 32, 157: 32, 143: 32, 139: 32, 134: 32, 114: 32, 170: 31, 154: 31, 131: 31, 168: 30, 136: 30, 177: 29, 176: 29, 151: 29, 117: 29, 153: 28, 137: 28, 192: 27, 162: 27, 147: 27, 186: 26, 156: 26, 155: 26, 145: 26, 140: 26, 216: 25, 212: 25, 208: 25, 188: 25, 164: 25, 158: 25, 125: 25, 193: 24, 191: 24, 179: 24, 171: 24, 167: 24, 200: 23, 196: 23, 161: 23, 159: 23, 273: 22, 202: 22, 197: 22, 195: 22, 183: 22, 178: 22, 175: 22, 346: 21, 263: 21, 220: 21, 182: 21, 276: 20, 247: 20, 224: 20, 222: 20, 221: 20, 201: 20, 198: 20, 181: 20, 180: 20, 240: 19, 217: 19, 215: 19, 210: 19, 185: 19, 174: 19, 165: 19, 365: 18, 258: 18, 251: 18, 239: 18, 229: 18, 211: 18, 207: 18, 184: 18, 172: 18, 288: 17, 257: 17, 242: 17, 238: 17, 235: 17, 234: 17, 232: 17, 206: 17, 203: 17, 190: 17, 173: 17, 349: 16, 325: 16, 309: 16, 293: 16, 278: 16, 272: 16, 269: 16, 265: 16, 254: 16, 246: 16, 245: 16, 231: 16, 227: 16, 226: 16, 225: 16, 189: 16, 169: 16, 390: 15, 355: 15, 294: 15, 252: 15, 249: 15, 223: 15, 218: 15, 214: 15, 204: 15, 199: 15, 333: 14, 318: 14, 286: 14, 281: 14, 268: 14, 261: 14, 244: 14, 243: 14, 219: 14, 209: 14, 205: 14, 402: 13, 307: 13, 303: 13, 295: 13, 292: 13, 287: 13, 280: 13, 274: 13, 267: 13, 260: 13, 248: 13, 213: 13, 187: 13, 465: 12, 386: 12, 331: 12, 304: 12, 302: 12, 300: 12, 299: 12, 289: 12, 284:
```

10, 368: 10, 363: 10, 359: 10, 352: 10, 345: 10, 332: 10, 330: 10, 326: 10, 315: 10, 311: 10, 290: 10, 279: 10, 270: 10, 510: 9, 471: 9, 451: 9, 444: 9, 425: 9, 421: 9, 405: 9, 381: 9, 364: 9, 362: 9, 358: 9, 354: 9, 341: 9, 329: 9, 324: 9, 323: 9, 321: 9, 317: 9, 310: 9, 301: 9, 297: 9, 283: 9, 264: 9, 262: 9, 255: 9, 230: 9, 228: 9, 701: 8, 632: 8, 588: 8, 542: 8, 538: 8, 481: 8, 462: 8, 458: 8, 449: 8, 445: 8, 443: 8, 438: 8, 428: 8, 416: 8, 406: 8, 404: 8, 395: 8, 374: 8, 373: 8, 369: 8, 361: 8, 360: 8, 356: 8, 347: 8, 344: 8, 342: 8, 338: 8, 328: 8, 320: 8, 316: 8, 314: 8, 306: 8, 277: 8, 275: 8, 259: 8, 250: 8, 237: 8, 754: 7, 714: 7, 654: 7, 635: 7, 610: 7, 576: 7, 534: 7, 527: 7, 501: 7, 499: 7, 486: 7, 460: 7, 455: 7, 452: 7, 447: 7, 437: 7, 431: 7, 419: 7, 417: 7, 415: 7, 412: 7, 410: 7, 408: 7, 401: 7, 371: 7, 367: 7, 366: 7, 339: 7, 319: 7, 313: 7, 308: 7, 296: 7, 881: 6, 855: 6, 843: 6, 764: 6, 731: 6, 694: 6, 664: 6, 661: 6, 652: 6, 624: 6, 609: 6, 596: 6, 571: 6, 564: 6, 547: 6, 545: 6, 521: 6, 517: 6, 505: 6, 490: 6, 487: 6, 475: 6, 463: 6, 456: 6, 454: 6, 446: 6, 423: 6, 414: 6, 409: 6, 403: 6, 398: 6, 391: 6, 388: 6, 387: 6, 385: 6, 353: 6, 350: 6, 322: 6, 305: 6, 291: 6, 256: 6, 2120: 5, 1439: 5, 1308: 5, 1056: 5, 963: 5, 940: 5, 904: 5, 886: 5, 816: 5, 771: 5, 769: 5, 761: 5, 748: 5, 723: 5, 720: 5, 707: 5, 683: 5, 681: 5, 670: 5, 668: 5, 659: 5, 656: 5, 653: 5, 647: 5, 631: 5, 627: 5, 614: 5, 611: 5, 602: 5, 586: 5, 585: 5, 583: 5, 582: 5, 578: 5, 568: 5, 566: 5, 559: 5, 555: 5, 549: 5, 544: 5, 532: 5, 530: 5, 526: 5, 520: 5, 515: 5, 507: 5, 502: 5, 497: 5, 492: 5, 485: 5, 480: 5, 470: 5, 466: 5, 461: 5, 459: 5, 453: 5, 442: 5, 440: 5, 439: 5, 434: 5, 429: 5, 427: 5, 426: 5, 399: 5, 397: 5, 393: 5, 389: 5, 383: 5, 377: 5, 370: 5, 343: 5, 337: 5, 336: 5, 335: 5, 298: 5, 1676: 4, 1617: 4, 1602: 4, 1541: 4, 1370: 4, 1347: 4, 1321: 4, 1306: 4, 1304: 4, 1216: 4, 1196: 4, 1187: 4, 1160: 4, 1109: 4, 1105: 4, 1067: 4, 1065: 4, 1036: 4, 1016: 4, 1007: 4, 1006: 4, 961: 4, 954: 4, 927: 4, 924: 4, 894: 4, 880: 4, 879: 4, 876: 4, 865: 4, 857: 4, 845: 4, 844: 4, 828: 4, 823: 4, 821: 4, 808: 4, 806: 4, 778: 4, 772: 4, 768: 4, 766: 4, 765: 4, 762: 4, 758: 4, 756: 4, 751: 4, 749: 4, 746: 4, 734: 4, 719: 4, 712: 4, 709: 4, 692: 4, 678: 4, 669: 4, 666: 4, 665: 4, 658: 4, 655: 4, 650: 4, 645: 4, 641: 4, 640: 4, 639: 4, 637: 4, 629: 4, 628: 4, 625: 4, 618: 4, 604: 4, 598: 4, 597: 4, 593: 4, 590: 4, 587: 4, 577: 4, 575: 4, 574: 4, 573: 4, 569: 4, 565: 4, 563: 4, 560: 4, 552: 4, 548: 4, 543: 4, 540: 4, 539: 4, 537: 4, 529: 4, 516: 4, 512: 4, 500: 4, 484: 4, 483: 4, 469: 4, 468: 4, 467: 4, 450: 4, 433: 4, 432: 4, 430: 4, 422: 4, 418: 4, 413: 4, 396: 4, 394: 4, 378: 4, 357: 4, 3507: 3, 2754: 3, 2377: 3, 2223: 3, 2126: 3, 2109: 3, 2063: 3, 2004: 3, 1966: 3, 1962: 3, 1939: 3, 1889: 3, 1841: 3, 1808: 3, 1795: 3, 1788: 3, 1763: 3, 1752: 3, 1749: 3, 1745: 3, 1628: 3, 1614: 3, 1598: 3, 1577: 3, 1547: 3, 1520: 3, 1516: 3, 1515: 3, 1513: 3, 1505: 3, 1472: 3, 1455: 3, 1450: 3, 1426: 3, 1382: 3, 1381: 3, 1368: 3, 1341: 3, 1325: 3, 1298: 3, 1297: 3, 1296: 3, 1286: 3, 1282: 3, 1275: 3, 1236: 3, 1224: 3, 1218: 3, 1201: 3, 1199: 3, 1198: 3, 1190: 3, 1168: 3, 1166: 3, 1163: 3, 1156: 3, 1148: 3, 1141: 3, 1129: 3, 1116: 3, 1115: 3, 1101: 3, 1097: 3, 1095: 3, 1092: 3, 1084: 3, 1078: 3, 1074: 3, 1073: 3, 1054: 3, 1049: 3, 1038: 3, 1009: 3, 1001: 3, 995: 3, 991: 3, 980: 3, 962: 3, 951: 3, 949: 3, 947: 3, 935: 3, 934: 3, 931: 3, 928: 3, 920: 3, 897: 3, 888: 3, 864: 3, 861: 3, 853: 3, 852: 3, 846: 3, 830: 3, 827: 3, 822: 3, 817: 3, 814: 3, 807: 3, 801: 3, 798: 3, 795: 3, 786: 3, 785: 3, 777: 3, 775: 3, 774: 3, 773: 3, 753: 3, 750: 3, 747: 3, 744: 3, 739: 3, 738: 3, 737: 3, 727: 3, 717: 3, 715: 3, 710: 3, 700: 3, 687: 3, 685: 3, 674: 3, 673: 3, 667: 3, 663: 3, 660: 3, 657: 3, 651: 3, 649: 3, 644: 3, 643: 3, 642: 3, 638: 3, 626: 3, 623: 3, 622: 3, 615: 3, 613: 3, 612: 3, 607: 3, 605: 3, 599: 3, 592: 3, 591: 3, 589: 3, 581: 3, 567: 3, 562: 3, 557: 3, 556: 3, 553: 3, 551: 3, 550: 3, 546: 3, 531: 3, 528: 3, 525: 3, 518: 3, 513: 3, 509: 3, 508: 3, 506: 3, 504: 3, 495: 3, 493: 3, 477: 3, 476: 3, 474: 3, 473: 3, 457: 3, 441: 3, 436: 3, 435: 3, 424: 3, 420: 3, 375: 3, 351: 3, 7557: 2, 7485: 2, 7211: 2, 7078: 2, 6711: 2, 6636: 2, 5920: 2, 5915: 2, 5741: 2, 5495: 2, 4879: 2, 4841: 2, 4679: 2, 4653: 2, 4428: 2, 4405: 2, 4359: 2, 4232: 2, 4201: 2, 4078: 2, 4044: 2, 4035: 2, 4026: 2, 4021: 2, 4003: 2, 3856: 2, 3755: 2, 3737: 2, 3682: 2, 3652: 2, 3595: 2, 3584: 2, 3534: 2, 3493: 2, 3467: 2, 3450: 2, 3404: 2, 3400: 2, 3347: 2, 3327: 2, 3202: 2, 3178: 2, 3156: 2, 3126: 2, 3101: 2, 3071: 2, 2986: 2, 2971: 2, 2969: 2, 2922: 2, 2840: 2, 2824: 2, 2779: 2, 2701: 2, 2699: 2, 2657: 2, 2633: 2, 2631: 2, 2614: 2, 2612: 2, 2602: 2, 2586: 2, 2548: 2, 2498: 2, 2489: 2, 2486: 2, 2466: 2, 2462: 2, 2457: 2, 2452: 2, 2440: 2, 2438: 2, 2435: 2, 2407: 2, 2404: 2, 2397: 2, 2369: 2, 2366: 2, 2361: 2, 2323: 2, 2320: 2, 2302: 2, 2238: 2, 2206: 2, 2191: 2, 2146: 2, 2135: 2, 2117: 2, 2076: 2, 2068: 2, 2021: 2, 2020: 2, 2002: 2, 2001: 2, 1992: 2, 1984: 2, 1969: 2, 1965: 2, 1961: 2, 1938: 2, 1928: 2, 1923: 2, 1919: 2, 1888: 2, 1868: 2, 1865: 2, 1859: 2, 1850: 2, 1849: 2, 1838: 2, 1817: 2, 1806: 2, 1803: 2, 1797: 2, 1790: 2, 1786: 2, 1758: 2, 1755: 2, 1754: 2, 1753: 2, 1739: 2, 1738: 2, 1728: 2, 1724: 2, 1714: 2, 1704: 2, 1698: 2, 1696: 2, 1694: 2, 1686: 2, 1661: 2, 1660: 2, 1657: 2, 1649: 2, 1643: 2, 1633: 2, 1629: 2, 1624: 2, 1622: 2, 1620: 2, 1597: 2, 1592: 2, 1589: 2, 1580: 2, 1579: 2, 1574: 2, 1571: 2, 1570: 2, 1562: 2, 1553: 2, 1537: 2, 1534: 2, 1529: 2, 1527: 2, 1523: 2, 1508: 2, 1499: 2, 1492: 2, 1491: 2, 1489: 2, 1487: 2, 1484: 2, 1480: 2, 1476: 2, 1471: 2, 1470: 2, 1468: 2, 1460: 2, 1451: 2, 1446: 2, 1442: 2, 1437: 2, 1420: 2, 1414: 2, 1412: 2, 1411: 2, 1399: 2, 1392: 2, 1391: 2, 1383: 2, 1377: 2, 1372: 2, 1364: 2, 1363: 2, 1353: 2, 1352: 2, 1345: 2, 1343: 2, 1339: 2, 1338: 2, 1332: 2, 1330: 2, 1324: 2, 1322: 2, 1320: 2, 1315: 2, 1303: 2, 1302: 2, 1295: 2, 1291: 2, 1289: 2, 1281: 2, 1279: 2, 1271: 2, 1269: 2, 1266: 2, 1260: 2, 1259: 2, 1254: 2, 1253: 2, 1249: 2, 1244: 2, 1241: 2, 1229: 2, 1222: 2, 1219: 2, 1215: 2, 1214: 2, 1211: 2, 1210: 2, 1208: 2, 1205: 2, 1202: 2, 1197: 2, 1192: 2, 1188: 2, 1185: 2, 1182: 2, 1181: 2, 1178: 2, 1175: 2, 1171: 2, 1162: 2, 1161: 2, 1158: 2, 1155: 2, 1147: 2, 1125: 2, 1123: 2, 1114: 2, 1111: 2, 1106: 2, 1104: 2, 1100: 2, 1098: 2, 1090: 2, 1089: 2, 1087: 2, 1083: 2, 1076: 2, 1075: 2, 1071: 2, 1070: 2, 1069: 2, 1068: 2, 1059: 2, 1050: 2, 1040: 2, 1031: 2, 1030: 2, 1029: 2, 1028: 2, 1025: 2, 1022: 2, 1021: 2, 1017: 2, 1015: 2, 1013: 2, 1004: 2, 1003: 2, 1000: 2, 996: 2, 994: 2, 990: 2, 989: 2, 988: 2, 987: 2, 985: 2, 978: 2, 975: 2, 972: 2, 960: 2, 958: 2, 956: 2, 955: 2, 953: 2, 952: 2, 950: 2, 945: 2, 944: 2, 942: 2, 941: 2, 938: 2, 936: 2, 933: 2, 930: 2, 926: 2, 925: 2, 918: 2, 913: 2, 911: 2, 910: 2, 907: 2, 903: 2, 895: 2, 893: 2, 891: 2, 890: 2, 889: 2, 884: 2, 883: 2, 882: 2, 877: 2, 874: 2, 867: 2, 863: 2, 862: 2, 860: 2, 856: 2, 848: 2, 847: 2, 842: 2, 841: 2, 839: 2, 838: 2, 836: 2, 835: 2, 834: 2, 832: 2, 831: 2, 829: 2, 813: 2, 811: 2, 810: 2, 800: 2, 797: 2, 796: 2, 794: 2, 792: 2, 791: 2, 787: 2, 784: 2, 783: 2, 781: 2, 779: 2, 776: 2, 770: 2, 767: 2, 760: 2, 759: 2, 757: 2, 755: 2, 742: 2, 741: 2,


```

32: 1, 2917: 1, 2914: 1, 2910: 1, 2909: 1, 2903: 1, 2902: 1, 2901: 1, 2891: 1, 2890: 1, 2888: 1, 28
78: 1, 2877: 1, 2876: 1, 2875: 1, 2867: 1, 2857: 1, 2852: 1, 2842: 1, 2833: 1, 2829: 1, 2828: 1, 28
15: 1, 2804: 1, 2799: 1, 2790: 1, 2785: 1, 2781: 1, 2766: 1, 2756: 1, 2752: 1, 2749: 1, 2746: 1, 27
45: 1, 2744: 1, 2738: 1, 2731: 1, 2730: 1, 2728: 1, 2727: 1, 2725: 1, 2721: 1, 2717: 1, 2714: 1, 27
06: 1, 2700: 1, 2697: 1, 2694: 1, 2688: 1, 2678: 1, 2675: 1, 2673: 1, 2665: 1, 2664: 1, 2658: 1, 26
56: 1, 2655: 1, 2653: 1, 2652: 1, 2651: 1, 2648: 1, 2647: 1, 2645: 1, 2641: 1, 2638: 1, 2634: 1, 26
28: 1, 2627: 1, 2626: 1, 2625: 1, 2618: 1, 2609: 1, 2608: 1, 2604: 1, 2601: 1, 2600: 1, 2596: 1, 25
85: 1, 2581: 1, 2580: 1, 2578: 1, 2577: 1, 2575: 1, 2574: 1, 2571: 1, 2569: 1, 2566: 1, 2565: 1, 25
63: 1, 2559: 1, 2555: 1, 2553: 1, 2547: 1, 2545: 1, 2537: 1, 2534: 1, 2533: 1, 2531: 1, 2529: 1, 25
10: 1, 2509: 1, 2505: 1, 2503: 1, 2497: 1, 2487: 1, 2483: 1, 2476: 1, 2475: 1, 2464: 1, 2461: 1, 24
59: 1, 2448: 1, 2444: 1, 2442: 1, 2422: 1, 2420: 1, 2419: 1, 2418: 1, 2416: 1, 2412: 1, 2406: 1, 24
05: 1, 2398: 1, 2396: 1, 2394: 1, 2392: 1, 2389: 1, 2380: 1, 2378: 1, 2376: 1, 2375: 1, 2374: 1, 23
65: 1, 2363: 1, 2358: 1, 2357: 1, 2355: 1, 2353: 1, 2351: 1, 2350: 1, 2349: 1, 2346: 1, 2342: 1, 23
41: 1, 2338: 1, 2336: 1, 2335: 1, 2327: 1, 2325: 1, 2315: 1, 2313: 1, 2312: 1, 2308: 1, 2306: 1, 23
00: 1, 2299: 1, 2295: 1, 2294: 1, 2289: 1, 2285: 1, 2278: 1, 2277: 1, 2275: 1, 2273: 1, 2271: 1, 22
69: 1, 2265: 1, 2262: 1, 2260: 1, 2258: 1, 2255: 1, 2253: 1, 2251: 1, 2249: 1, 2248: 1, 2247: 1, 22
45: 1, 2244: 1, 2236: 1, 2235: 1, 2232: 1, 2228: 1, 2225: 1, 2222: 1, 2220: 1, 2216: 1, 2210: 1, 22
09: 1, 2202: 1, 2200: 1, 2198: 1, 2196: 1, 2192: 1, 2189: 1, 2188: 1, 2184: 1, 2183: 1, 2179: 1, 21
73: 1, 2170: 1, 2159: 1, 2157: 1, 2152: 1, 2151: 1, 2150: 1, 2145: 1, 2143: 1, 2138: 1, 2134: 1, 21
33: 1, 2132: 1, 2131: 1, 2121: 1, 2116: 1, 2110: 1, 2104: 1, 2102: 1, 2101: 1, 2098: 1, 2095: 1, 20
89: 1, 2086: 1, 2085: 1, 2081: 1, 2078: 1, 2077: 1, 2074: 1, 2067: 1, 2066: 1, 2060: 1, 2057: 1, 20
55: 1, 2050: 1, 2045: 1, 2039: 1, 2035: 1, 2033: 1, 2024: 1, 2022: 1, 2018: 1, 2016: 1, 2015: 1, 20
14: 1, 2013: 1, 2007: 1, 2005: 1, 2000: 1, 1999: 1, 1997: 1, 1995: 1, 1994: 1, 1993: 1, 1990: 1, 19
88: 1, 1986: 1, 1983: 1, 1982: 1, 1979: 1, 1977: 1, 1973: 1, 1972: 1, 1968: 1, 1963: 1, 1960: 1, 19
59: 1, 1956: 1, 1954: 1, 1953: 1, 1951: 1, 1950: 1, 1948: 1, 1946: 1, 1945: 1, 1944: 1, 1936: 1, 19
34: 1, 1932: 1, 1927: 1, 1925: 1, 1924: 1, 1922: 1, 1917: 1, 1913: 1, 1912: 1, 1909: 1, 1907: 1, 19
05: 1, 1902: 1, 1900: 1, 1897: 1, 1893: 1, 1892: 1, 1887: 1, 1886: 1, 1885: 1, 1883: 1, 1882: 1, 18
81: 1, 1880: 1, 1879: 1, 1875: 1, 1873: 1, 1872: 1, 1870: 1, 1869: 1, 1867: 1, 1866: 1, 1863: 1, 18
58: 1, 1852: 1, 1847: 1, 1845: 1, 1844: 1, 1843: 1, 1840: 1, 1833: 1, 1832: 1, 1828: 1, 1827: 1, 18
25: 1, 1822: 1, 1820: 1, 1819: 1, 1818: 1, 1816: 1, 1815: 1, 1814: 1, 1813: 1, 1810: 1, 1809: 1, 18
07: 1, 1804: 1, 1802: 1, 1801: 1, 1800: 1, 1798: 1, 1792: 1, 1789: 1, 1785: 1, 1784: 1, 1783: 1, 17
82: 1, 1781: 1, 1780: 1, 1779: 1, 1768: 1, 1767: 1, 1764: 1, 1762: 1, 1761: 1, 1757: 1, 1744: 1, 17
43: 1, 1740: 1, 1737: 1, 1733: 1, 1732: 1, 1727: 1, 1719: 1, 1717: 1, 1713: 1, 1706: 1, 1701: 1, 17
99: 1, 1695: 1, 1691: 1, 1685: 1, 1680: 1, 1679: 1, 1673: 1, 1672: 1, 1670: 1, 1669: 1, 1667: 1, 16
66: 1, 1665: 1, 1664: 1, 1663: 1, 1662: 1, 1659: 1, 1653: 1, 1652: 1, 1651: 1, 1647: 1, 1645: 1, 16
41: 1, 1638: 1, 1637: 1, 1635: 1, 1634: 1, 1631: 1, 1630: 1, 1623: 1, 1621: 1, 1616: 1, 1613: 1, 16
12: 1, 1609: 1, 1608: 1, 1604: 1, 1601: 1, 1594: 1, 1593: 1, 1588: 1, 1587: 1, 1586: 1, 1585: 1, 15
84: 1, 1583: 1, 1581: 1, 1578: 1, 1572: 1, 1569: 1, 1567: 1, 1566: 1, 1565: 1, 1564: 1, 1563: 1, 15
60: 1, 1559: 1, 1558: 1, 1557: 1, 1556: 1, 1554: 1, 1552: 1, 1551: 1, 1543: 1, 1538: 1, 1536: 1, 15
32: 1, 1528: 1, 1526: 1, 1525: 1, 1524: 1, 1518: 1, 1517: 1, 1514: 1, 1512: 1, 1511: 1, 1504: 1, 15
02: 1, 1498: 1, 1496: 1, 1495: 1, 1490: 1, 1488: 1, 1485: 1, 1478: 1, 1474: 1, 1473: 1, 1469: 1, 14
67: 1, 1459: 1, 1458: 1, 1457: 1, 1448: 1, 1447: 1, 1445: 1, 1443: 1, 1440: 1, 1438: 1, 1434: 1, 14
31: 1, 1429: 1, 1425: 1, 1423: 1, 1419: 1, 1416: 1, 1413: 1, 1409: 1, 1406: 1, 1405: 1, 1404: 1, 14
03: 1, 1398: 1, 1396: 1, 1394: 1, 1388: 1, 1387: 1, 1386: 1, 1385: 1, 1384: 1, 1379: 1, 1378: 1, 13
75: 1, 1374: 1, 1371: 1, 1369: 1, 1367: 1, 1366: 1, 1362: 1, 1361: 1, 1360: 1, 1358: 1, 1356: 1, 13
54: 1, 1342: 1, 1337: 1, 1336: 1, 1335: 1, 1334: 1, 1331: 1, 1328: 1, 1327: 1, 1323: 1, 1319: 1, 13
16: 1, 1314: 1, 1313: 1, 1312: 1, 1311: 1, 1310: 1, 1309: 1, 1307: 1, 1305: 1, 1300: 1, 1293: 1, 12
90: 1, 1288: 1, 1285: 1, 1283: 1, 1276: 1, 1274: 1, 1272: 1, 1268: 1, 1267: 1, 1265: 1, 1264: 1, 12
63: 1, 1262: 1, 1261: 1, 1258: 1, 1257: 1, 1251: 1, 1247: 1, 1246: 1, 1243: 1, 1242: 1, 1237: 1, 12
35: 1, 1232: 1, 1230: 1, 1227: 1, 1226: 1, 1223: 1, 1213: 1, 1206: 1, 1200: 1, 1195: 1, 1191: 1, 11
86: 1, 1183: 1, 1179: 1, 1177: 1, 1176: 1, 1174: 1, 1173: 1, 1170: 1, 1169: 1, 1164: 1, 1151: 1, 11
46: 1, 1143: 1, 1142: 1, 1138: 1, 1137: 1, 1133: 1, 1132: 1, 1131: 1, 1130: 1, 1128: 1, 1127: 1, 11
26: 1, 1121: 1, 1120: 1, 1117: 1, 1112: 1, 1108: 1, 1107: 1, 1103: 1, 1102: 1, 1099: 1, 1094: 1, 10
91: 1, 1086: 1, 1080: 1, 1079: 1, 1072: 1, 1066: 1, 1063: 1, 1062: 1, 1061: 1, 1058: 1, 1055: 1, 10
53: 1, 1052: 1, 1051: 1, 1048: 1, 1047: 1, 1046: 1, 1044: 1, 1042: 1, 1041: 1, 1039: 1, 1037: 1, 10
34: 1, 1033: 1, 1032: 1, 1027: 1, 1026: 1, 1024: 1, 1023: 1, 1019: 1, 1018: 1, 1014: 1, 1011: 1, 10
05: 1, 1002: 1, 999: 1, 998: 1, 997: 1, 993: 1, 992: 1, 984: 1, 983: 1, 981: 1, 979: 1, 974: 1,
971: 1, 970: 1, 968: 1, 964: 1, 957: 1, 948: 1, 946: 1, 943: 1, 939: 1, 932: 1, 929: 1, 923: 1,
921: 1, 917: 1, 916: 1, 915: 1, 914: 1, 912: 1, 909: 1, 908: 1, 905: 1, 902: 1, 901: 1, 900: 1,
899: 1, 898: 1, 896: 1, 892: 1, 887: 1, 878: 1, 872: 1, 869: 1, 868: 1, 866: 1, 859: 1, 858: 1,
850: 1, 849: 1, 840: 1, 837: 1, 833: 1, 825: 1, 820: 1, 818: 1, 815: 1, 809: 1, 804: 1, 803: 1,
802: 1, 799: 1, 793: 1, 790: 1, 788: 1, 780: 1, 752: 1, 743: 1, 740: 1, 736: 1, 733: 1, 732: 1,
729: 1, 726: 1, 708: 1, 705: 1, 702: 1, 691: 1, 690: 1, 686: 1, 684: 1, 680: 1, 662: 1, 620: 1,
608: 1, 601: 1, 595: 1, 572: 1, 570: 1, 558: 1, 554: 1, 536: 1, 533: 1, 523: 1, 519: 1, 494: 1,
464: 1}

```

In [64]:

```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
```

```
alpha = [10 ** x for x in range(-5, 1)]
```

```
# read more about SGDClassifier() at http://scikit-
```

```
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
```

```
# -----
```

```

ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

#-----
# video link:
#-----


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, 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(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

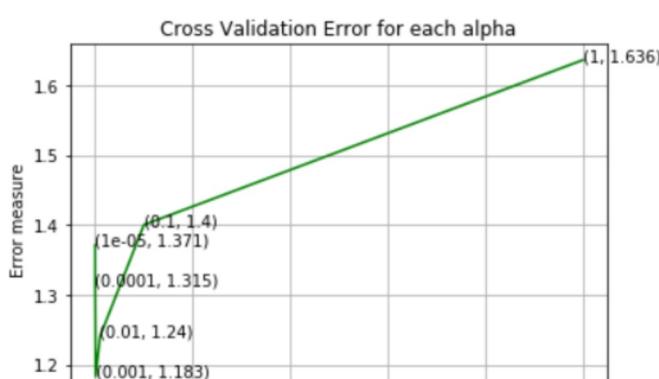
predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
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, p
redict_y, labels=clf.classes_, eps=1e-15))

```

```

For values of alpha =  1e-05 The log loss is: 1.3709434761923998
For values of alpha =  0.0001 The log loss is: 1.3146645671674828
For values of alpha =  0.001 The log loss is: 1.1830761835400556
For values of alpha =  0.01 The log loss is: 1.2402678498175872
For values of alpha =  0.1 The log loss is: 1.3999248278444862
For values of alpha =  1 The log loss is: 1.6356823134476974

```



```
For values of best alpha = 0.001 The train log loss is: 0.7584207407557678
For values of best alpha = 0.001 The cross validation log loss is: 1.1830761835400556
For values of best alpha = 0.001 The test log loss is: 1.1963633403579697
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it seems like!

In [65]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_textfea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_textfea_counts = df_textfea.sum(axis=0).A1
    df_textfea_dict = dict(zip(list(df_text_features), df_textfea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1, len2
```

In [66]:

```
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")
```

```
96.307 % of word of test data appeared in train data
98.811 % of word of Cross Validation appeared in train data
```

In [75]:

```
result.head(1)
```

Out[75]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating_Mutations	1	cyclin dependent kinases cdks regulate variety...

In [76]:

```
result = result[['ID', 'Gene', 'Variation', 'TEXT', 'Class']]
```

In [79]:

```
result.head(1)
```

Out[79]:

	ID	Gene	Variation	TEXT	Class
0	0	FAM58A	Truncating_Mutations	cyclin dependent kinases cdks regulate variety...	1

In [80]:

```
result.to_csv('cancer_data', index=False)
```

In [1]:

```
import plotly.plotly as py
```

```
offline.init_notebook_mode()
```

In [2]:

```
values_d = [[['1', '2', '3', '4', '5'],
["What type of feature it is ?",
"How many categories or unique words are there ?",
"How to featurize this feature ?",
"How good is this feature in predicting y_i ?",
"Is the feature stable across all the data sets (Train, Test and CV) ?"
],
["categorical variable",
"226 categories",
"one hot encoding or response coding",
"Yes (obtained average log loss of 1.1)",
"Yes (average overlap of data b/w train, test and CV =95%)"
],
[
    "categorical variable",
    "1929 categories",
    "one hot encoding or response coding",
    "Yes (obtained average log loss of 1.6)",
    "No (average overlap of data b/w train, test and CV =10%)"
],
[
    "Text",
    "52865 (unique words in train data)",
    "one hot encoding or response coding",
    "Yes (obtained average log loss of 1.1)",
    "Yes (average overlap of data b/w train, test and CV =97%)"
]
]
```

In [3]:

```
headerColor = 'grey'
rowEvenColor = 'lightgrey'
rowOddColor = 'white'

trace0 = go.Table(
    header = dict(
        values = ['<b></b>',
                  '<b>Questions ?</b>',
                  '<b>Gene</b>',
                  '<b>Variation</b>',
                  '<b>Text'
                ],
        line = dict(color = '#506784'),
        fill = dict(color = headerColor),
        align = ['right'],
        font = dict(color = 'white', size = 10),
        height=40
    ),
    cells = dict(
        values = values_d,
        line = dict(color = '#506784'),
        fill = dict(color = [rowOddColor, rowEvenColor, rowOddColor, rowEvenColor]),
        align = [ 'center'],
        font = dict(color = '#506784', size = 11),
        height=40
    )
)
layout = dict(width=5000, height=1000)
data = [trace0]
```

Summary of EDA

2. Total no of features for each data points - 4
3. (a) ID : the id of the row used to link the mutation to the clinical evidence
 4. (b) Gene : the gene where this genetic mutation is located
 5. (c) Variation : the aminoacid change for this mutations
 6. (d) Class : 1-9 the class this genetic mutation has been classified on
7. The distribution of Classes is not uniform and some classes occur more frequently than the other classes, therefore balancing the class weights before applying any ML model is important.
8. Log loss obtained by applying random model was 2.5 therefore our objective will be to obtain a Log loss less than 2.5 while applying ML models.

In [4]:

```
offline.iplot(data, layout)
```