cancer-project

January 2, 2024

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')

import os
folder_path = '/content/drive/MyDrive/DS&ML'
os.chdir(folder_path)
```

```
Mounted at /content/drive
```

1 Libraries

[ ]:
```
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 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
```

[ ]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]: True

2 Reading and PreProcessing the Data

[ ]:
```
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']
```

[ ]:

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

[ ]:
```
print(type(data))
print(type(data.columns))
print(type(data.columns.values))
print(data.columns)
print(data.columns.values)
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.indexes.base.Index'>
<class 'numpy.ndarray'>
Index(['ID', 'Gene', 'Variation', 'Class'], dtype='object')
['ID' 'Gene' 'Variation' 'Class']
```

[ ]:
```
data_text = pd.
  ↪read_csv("training_text",sep="\|\|",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)
print(data_text.values[0,1])
data_text.head()
```

```
Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']
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).Here, we deorphanize CDK10 by identifying cyclin M, the product of FAM58A, as a binding partner. Mutations in this gene that predict absence or truncation of cyclin M are associated with STAR syndrome, whose features include toe syndactyly, telecanthus, and anogenital and renal malformations in heterozygous females (10). However, both the functions of cyclin M and the pathogenesis of STAR syndrome remain unknown. We show that a recombinant CDK10/cyclin M heterodimer is an active protein kinase that phosphorylates ETS2 in vitro. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and phospho-ERK expression levels and in inducing tamoxifen resistance in estrogen receptor (ER)+ breast cancer cells. We show that CDK10/cyclin M positively controls ETS2 degradation by the proteasome, through the phosphorylation of two neighboring serines. Finally, we detect an increased ETS2 expression level in cells derived from a STAR patient, and we demonstrate that it is attributable to the decreased cyclin M expression level observed in these cells.Previous SectionNext SectionResultsA yeast two-hybrid (Y2H) screen unveiled an interaction signal between CDK10 and a mouse protein whose C-terminal half presents a strong sequence homology with the human FAM58A gene product [whose proposed name is cyclin M (11)]. We thus performed Y2H mating assays to determine whether human CDK10 interacts with human cyclin M (Fig. 1 A-C). The longest CDK10 isoform (P1) expressed as a bait protein produced a strong interaction phenotype with full-length cyclin M (expressed as a prey protein) but no detectable phenotype with cyclin D1, p21 (CIP1), and Cdi1 (KAP), which are known binding partners of other CDKs (Fig. 1B). CDK1 and CDK3 also produced Y2H signals with cyclin M, albeit notably weaker than that observed with CDK10 (Fig. 1B). An interaction phenotype was also observed between full-length cyclin M and CDK10 proteins expressed as bait and prey, respectively (Fig. S1A). We then tested different isoforms of CDK10 and cyclin M originating from alternative gene splicing, and two truncated cyclin M proteins corresponding to the hypothetical products of two mutated FAM58A genes found in STAR syndrome patients (10). None of these shorter isoforms produced interaction phenotypes (Fig. 1 A and C and Fig. S1A).Fig. 1.In a new window Download PPTFig. 1.CDK10 and cyclin M form an interaction complex. (A) Schematic representation of the different protein isoforms analyzed by Y2H assays. Amino acid numbers are indicated. Black boxes indicate internal deletions. The red box indicates a differing amino acid sequence compared with CDK10 P1. (B) Y2H assay between a set of CDK proteins expressed as baits (in fusion to the LexA DNA binding domain) and CDK interacting proteins expressed as preys (in fusion to the B42 transcriptional activator). pEG202 and pJG4-5 are the empty bait and prey plasmids expressing LexA and B42, respectively. lacZ was used as a reporter gene, and blue yeast are indicative of a Y2H interaction phenotype. (C) Y2H assay between the different CDK10 and cyclin M isoforms. The amino-terminal region of ETS2, known to interact with CDK10 (9), was also assayed. (D) Western blot analysis of Myc-CDK10 (wt or kd) and CycM-V5-6His expression levels in transfected HEK293 cells. (E) Western blot analysis of Myc-CDK10 (wt or kd) immunoprecipitates obtained using the anti-Myc antibody. "Inputs" correspond to 10 g total lysates obtained from HEK293 cells coexpressing Myc-CDK10 (wt or kd) and CycM-V5-6His. (F) Western blot analysis of immunoprecipitates obtained using the anti-CDK10 antibody or a control goat antibody, from human breast cancer

MCF7 cells. "Input" corresponds to 30 g MCF7 total cell lysates. The lower band of the doublet observed on the upper panel comigrates with the exogenously expressed untagged CDK10 and thus corresponds to endogenous CDK10. The upper band of the doublet corresponds to a nonspecific signal, as demonstrated by it insensitivity to either overexpression of CDK10 (as seen on the left lane) or silencing of CDK10 (Fig. S2B). Another experiment with a longer gel migration is shown in Fig. S1D.Next we examined the ability of CDK10 and cyclin M to interact when expressed in human cells (Fig. 1 D and E). We tested wild-type CDK10 (wt) and a kinase dead (kd) mutant bearing a D181A amino acid substitution that abolishes ATP binding (12). We expressed cyclin M-V5-6His and/or Myc-CDK10 (wt or kd) in a human embryonic kidney cell line (HEK293). The expression level of cyclin M-V5-6His was significantly increased upon coexpression with Myc-CDK10 (wt or kd) and, to a lesser extent, that of Myc-CDK10 (wt or kd) was increased upon coexpression with cyclin M-V5-6His (Fig. 1D). We then immunoprecipitated Myc-CDK10 proteins and detected the presence of cyclin M in the CDK10 (wt) and (kd) immunoprecipitates only when these proteins were coexpressed pair-wise (Fig. 1E). We confirmed these observations by detecting the presence of Myc-CDK10 in cyclin M-V5-6His immunoprecipitates (Fig. S1B). These experiments confirmed the lack of robust interaction between the CDK10.P2 isoform and cyclin M (Fig. S1C). To detect the interaction between endogenous proteins, we performed immunoprecipitations on nontransfected MCF7 cells derived from a human breast cancer. CDK10 and cyclin M antibodies detected their cognate endogenous proteins by Western blotting. We readily detected cyclin M in immunoprecipitates obtained with the CDK10 antibody but not with a control antibody (Fig. 1F). These results confirm the physical interaction between CDK10 and cyclin M in human cells.To unveil a hypothesized CDK10/cyclin M protein kinase activity, we produced GST-CDK10 and StrepII-cyclin M fusion proteins in insect cells, either individually or in combination. We observed that GST-CDK10 and StrepII-cyclin M copurified, thus confirming their interaction in yet another cellular model (Fig. 2A). We then performed in vitro kinase assays with purified proteins, using histone H1 as a generic substrate. Histone H1 phosphorylation was detected only from lysates of cells coexpressing GST-CDK10 and StrepII-cyclin M. No phosphorylation was detected when GST-CDK10 or StrepII-cyclin M were expressed alone, or when StrepII-cyclin M was coexpressed with GST-CDK10(kd) (Fig. 2A). Next we investigated whether ETS2, which is known to interact with CDK10 (9) (Fig. 1C), is a phosphorylation substrate of CDK10/cyclin M. We detected strong phosphorylation of ETS2 by the GST-CDK10/StrepII-cyclin M purified heterodimer, whereas no phosphorylation was detected using GST-CDK10 alone or GST-CDK10(kd)/StrepII-cyclin M heterodimer (Fig. 2B).Fig. 2.In a new window Download PPTFig. 2.CDK10 is a cyclin M-dependent protein kinase. (A) In vitro protein kinase assay on histone H1. Lysates from insect cells expressing different proteins were purified on a glutathione Sepharose matrix to capture GST-CDK10(wt or kd) fusion proteins alone, or in complex with STR-CycM fusion protein. Purified protein expression levels were analyzed by Western blots (Top and Upper Middle). The kinase activity was determined by autoradiography of histone H1, whose added amounts were visualized by Coomassie staining (Lower Middle and Bottom). (B) Same as in A, using purified recombinant 6His-ETS2 as a substrate.CDK10 silencing has been shown to increase ETS2-driven c-RAF

transcription and to activate the MAPK pathway (6). We investigated whether cyclin M is also involved in this regulatory pathway. To aim at a highly specific silencing, we used siRNA pools (mix of four different siRNAs) at low final concentration (10 nM). Both CDK10 and cyclin M siRNA pools silenced the expression of their cognate targets (Fig. 3 A and C and Fig. S2) and, interestingly, the cyclin M siRNA pool also caused a marked decrease in CDK10 protein level (Fig. 3A and Fig. S2B). These results, and those shown in Fig. 1D, suggest that cyclin M binding stabilizes CDK10. Cyclin M silencing induced an increase in c-Raf protein and mRNA levels (Fig. 3 B and C) and in phosphorylated ERK1 and ERK2 protein levels (Fig. S3B), similarly to CDK10 silencing. As expected from these effects (6), CDK10 and cyclin M silencing both decreased the sensitivity of ER+ MCF7 cells to tamoxifen, to a similar extent. The combined silencing of both genes did not result in a higher resistance to the drug (Fig. S3C). Altogether, these observations demonstrate a functional interaction between cyclin M and CDK10, which negatively controls ETS2.Fig. 3.In a new window Download PPTFig. 3.Cyclin M silencing up-regulates c-Raf expression. (A) Western blot analysis of endogenous CDK10 and cyclin M expression levels in MCF7 cells, in response to siRNA-mediated gene silencing. (B) Western blot analysis of endogenous c-Raf expression levels in MCF7 cells, in response to CDK10 or cyclin M silencing. A quantification is shown in Fig. S3A. (C) Quantitative RT-PCR analysis of CDK10, cyclin M, and c-Raf mRNA levels, in response to CDK10 (Upper) or cyclin M (Lower) silencing. **P 0.01; ***P 0.001.We then wished to explore the mechanism by which CDK10/cyclin M controls ETS2. ETS2 is a short-lived protein degraded by the proteasome (13). A straightforward hypothesis is that CDK10/cyclin M positively controls ETS2 degradation. We thus examined the impact of CDK10 or cyclin M silencing on ETS2 expression levels. The silencing of CDK10 and that of cyclin M caused an increase in the expression levels of an exogenously expressed Flag-ETS2 protein (Fig. S4A), as well as of the endogenous ETS2 protein (Fig. 4A). This increase is not attributable to increased ETS2 mRNA levels, which marginally fluctuated in response to CDK10 or cyclin M silencing (Fig. S4B). We then examined the expression levels of the Flag-tagged ETS2 protein when expressed alone or in combination with Myc-CDK10 or -CDK10(kd), with or without cyclin M-V5-6His. Flag-ETS2 was readily detected when expressed alone or, to a lesser extent, when coexpressed with CDK10(kd). However, its expression level was dramatically decreased when coexpressed with CDK10 alone, or with CDK10 and cyclin M (Fig. 4B). These observations suggest that endogenous cyclin M levels are in excess compared with those of CDK10 in MCF7 cells, and they show that the major decrease in ETS2 levels observed upon CDK10 coexpression involves CDK10 kinase activity. Treatment of cells coexpressing Flag-ETS2, CDK10, and cyclin M with the proteasome inhibitor MG132 largely rescued Flag-ETS2 expression levels (Fig. 4B).Fig. 4.In a new window Download PPTFig. 4.CDK10/cyclin M controls ETS2 stability in human cancer derived cells. (A) Western blot analysis of endogenous ETS2 expression levels in MCF7 cells, in response to siRNA-mediated CDK10 and/or cyclin M silencing. A quantification is shown in Fig. S4B. (B) Western blot analysis of exogenously expressed Flag-ETS2 protein levels in MCF7 cells cotransfected with empty vectors or coexpressing Myc-CDK10 (wt or kd), or Myc-CDK10/CycM-V5-6His. The latter cells were treated for 16 h with the MG132 proteasome inhibitor. Proper expression of CDK10 and

cyclin M tagged proteins was verified by Western blot analysis. (C and D) Western blot analysis of expression levels of exogenously expressed Flag-ETS2 wild-type or mutant proteins in MCF7 cells, in the absence of (C) or in response to (D) Myc-CDK10/CycM-V5-6His expression. Quantifications are shown in Fig. S4 C and D.A mass spectrometry analysis of recombinant ETS2 phosphorylated by CDK10/cyclin M in vitro revealed the existence of multiple phosphorylated residues, among which are two neighboring phospho-serines (at positions 220 and 225) that may form a phosphodegron (14) (Figs. S5-S8). To confirm this finding, we compared the phosphorylation level of recombinant ETS2wt with that of ETS2SASA protein, a mutant bearing alanine substitutions of these two serines. As expected from the existence of multiple phosphorylation sites, we detected a small but reproducible, significant decrease of phosphorylation level of ETS2SASA compared with ETS2wt (Fig. S9), thus confirming that Ser220/Ser225 are phosphorylated by CDK10/cyclin M. To establish a direct link between ETS2 phosphorylation by CDK10/cyclin M and degradation, we examined the expression levels of Flag-ETS2SASA. In the absence of CDK10/cyclin M coexpression, it did not differ significantly from that of Flag-ETS2. This is contrary to that of Flag-ETS2DBM, bearing a deletion of the N-terminal destruction (D-) box that was previously shown to be involved in APC-Cdh1-mediated degradation of ETS2 (13) (Fig. 4C). However, contrary to Flag-ETS2 wild type, the expression level of Flag-ETS2SASA remained insensitive to CDK10/cyclin M coexpression (Fig. 4D). Altogether, these results suggest that CDK10/cyclin M directly controls ETS2 degradation through the phosphorylation of these two serines.Finally, we studied a lymphoblastoid cell line derived from a patient with STAR syndrome, bearing FAM58A mutation c.555+1G>A, predicted to result in aberrant splicing (10). In accordance with incomplete skewing of X chromosome inactivation previously found in this patient, we detected a decreased expression level of cyclin M protein in the STAR cell line, compared with a control lymphoblastoid cell line. In line with our preceding observations, we detected an increased expression level of ETS2 protein in the STAR cell line compared with the control (Fig. 5A and Fig. S10A). We then examined by quantitative RT-PCR the mRNA expression levels of the corresponding genes. The STAR cell line showed a decreased expression level of cyclin M mRNA but an expression level of ETS2 mRNA similar to that of the control cell line (Fig. 5B). To demonstrate that the increase in ETS2 protein expression is indeed a result of the decreased cyclin M expression observed in the STAR patient-derived cell line, we expressed cyclin M-V5-6His in this cell line. This expression caused a decrease in ETS2 protein levels (Fig. 5C).Fig. 5.In a new window Download PPTFig. 5.Decreased cyclin M expression in STAR patient-derived cells results in increased ETS2 protein level. (A) Western blot analysis of cyclin M and ETS2 protein levels in a STAR patient-derived lymphoblastoid cell line and in a control lymphoblastoid cell line, derived from a healthy individual. A quantification is shown in Fig. S10A. (B) Quantitative RT-PCR analysis of cyclin M and ETS2 mRNA levels in the same cells. ***P 0.001. (C) Western blot analysis of ETS2 protein levels in the STAR patient-derived lymphoblastoid cell line transfected with an empty vector or a vector directing the expression of cyclin M-V5-6His. Another Western blot revealing endogenously and exogenously expressed cyclin M levels is shown in Fig. S10B. A quantification of ETS2 protein levels is shown in Fig. S10C.Previous SectionNext

SectionDiscussionIn this work, we unveil the interaction between CDK10, the last orphan CDK discovered in the pregenomic era (2), and cyclin M, the only cyclin associated with a human genetic disease so far, and whose functions remain unknown (10). The closest paralogs of CDK10 within the CDK family are the CDK11 proteins, which interact with L-type cyclins (15). Interestingly, the closest paralog of these cyclins within the cyclin family is cyclin M (Fig. S11). The fact that none of the shorter CDK10 isoforms interact robustly with cyclin M suggests that alternative splicing of the CDK10 gene (16, 17) plays an important role in regulating CDK10 functions.The functional relevance of the interaction between CDK10 and cyclin M is supported by different observations. Both proteins seem to enhance each other's stability, as judged from their increased expression levels when their partner is exogenously coexpressed (Fig. 1D) and from the much reduced endogenous CDK10 expression level observed in response to cyclin M silencing (Fig. 3A and Fig. S2B). CDK10 is subject to ubiquitin-mediated degradation (18). Our observations suggest that cyclin M protects CDK10 from such degradation and that it is the only cyclin partner of CDK10, at least in MCF7 cells. They also suggest that cyclin M stability is enhanced upon binding to CDK10, independently from its kinase activity, as seen for cyclin C and CDK8 (19). We uncover a cyclin M-dependent CDK10 protein kinase activity in vitro, thus demonstrating that this protein, which was named a CDK on the sole basis of its amino acid sequence, is indeed a genuine cyclin-dependent kinase. Our Y2H assays reveal that truncated cyclin M proteins corresponding to the hypothetical products of two STAR syndrome-associated FAM58A mutations do not produce an interaction phenotype with CDK10. Hence, regardless of whether these mutated mRNAs undergo nonsense-mediated decay (as suggested from the decreased cyclin M mRNA levels in STAR cells, shown in Fig. 5B) or give rise to truncated cyclin M proteins, females affected by the STAR syndrome must exhibit compromised CDK10/cyclin M kinase activity at least in some tissues and during specific developmental stages.We show that ETS2, a known interactor of CDK10, is a phosphorylation substrate of CDK10/cyclin M in vitro and that CDK10/cyclin M kinase activity positively controls ETS2 degradation by the proteasome. This control seems to be exerted through a very fine mechanism, as judged from the sensitivity of ETS2 levels to partially decreased CDK10 and cyclin M levels, achieved in MCF7 cells and observed in STAR cells, respectively. These findings offer a straightforward explanation for the already reported up-regulation of ETS2-driven transcription of c-RAF in response to CDK10 silencing (6). We bring evidence that CDK10/cyclin M directly controls ETS2 degradation through the phosphorylation of two neighboring serines, which may form a noncanonical -TRCP phosphodegron (DSMCPAS) (14). Because none of these two serines precede a proline, they do not conform to usual CDK phosphorylation sites. However, multiple so-called transcriptional CDKs (CDK7, -8, -9, and -11) (to which CDK10 may belong; Fig. S11) have been shown to phosphorylate a variety of motifs in a non-proline-directed fashion, especially in the context of molecular docking with the substrate (20). Here, it can be hypothesized that the high-affinity interaction between CDK10 and the Pointed domain of ETS2 (6, 9) (Fig. 1C) would allow docking-mediated phosphorylation of atypical sites. The control of ETS2 degradation involves a number of players, including APC-Cdh1 (13) and the cullin-RING ligase CRL4 (21). The formal identification of the ubiquitin ligase

involved in the CDK10/cyclin M pathway and the elucidation of its concerted action with the other ubiquitin ligases to regulate ETS2 degradation will require further studies.Our results present a number of significant biological and medical implications. First, they shed light on the regulation of ETS2, which plays an important role in development (22) and is frequently deregulated in many cancers (23). Second, our results contribute to the understanding of the molecular mechanisms causing tamoxifen resistance associated with reduced CDK10 expression levels, and they suggest that, like CDK10 (6), cyclin M could also be a predictive clinical marker of hormone therapy response of ER -positive breast cancer patients. Third, our findings offer an interesting hypothesis on the molecular mechanisms underlying STAR syndrome. Ets2 transgenic mice showing a less than twofold overexpression of Ets2 present severe cranial abnormalities (24), and those observed in STAR patients could thus be caused at least in part by increased ETS2 protein levels. Another expected consequence of enhanced ETS2 expression levels would be a decreased risk to develop certain types of cancers and an increased risk to develop others. Studies on various mouse models (including models of Down syndrome, in which three copies of ETS2 exist) have revealed that ETS2 dosage can repress or promote tumor growth and, hence, that ETS2 exerts noncell autonomous functions in cancer (25). Intringuingly, one of the very few STAR patients identified so far has been diagnosed with a nephroblastoma (26). Finally, our findings will facilitate the general exploration of the biological functions of CDK10 and, in particular, its role in the control of cell division. Previous studies have suggested either a positive role in cell cycle control (5, 6) or a tumor-suppressive activity in some cancers (7, 8). The severe growth retardation exhibited by STAR patients strongly suggests that CDK10/cyclin M plays an important role in the control of cell proliferation.Previous SectionNext SectionMaterials and MethodsCloning of CDK10 and cyclin M cDNAs, plasmid constructions, tamoxifen response analysis, quantitative RT-PCR, mass spectrometry experiments, and antibody production are detailed in SI Materials and Methods.Yeast Two-Hybrid Interaction Assays. We performed yeast interaction mating assays as previously described (27).Mammalian Cell Cultures and Transfections. We grew human HEK293 and MCF7 cells in DMEM supplemented with 10% (vol/vol) FBS (Invitrogen), and we grew lymphoblastoid cells in RPMI 1640 GlutaMAX supplemented with 15% (vol/vol) FBS. We transfected HEK293 and MCF7 cells using Lipofectamine 2000 (Invitrogen) for plasmids, Lipofectamine RNAiMAX (Invitrogen) for siRNAs, and Jetprime (Polyplus) for plasmids/siRNAs combinations according to the manufacturers' instructions. We transfected lymphoblastoid cells by electroporation (Neon, Invitrogen). For ETS2 stability studies we treated MCF7 cells 32 h after transfection with 10 M MG132 (Fisher Scientific) for 16 h.Coimmunoprecipitation and Western Blot Experiments. We collected cells by scraping in PBS (or centrifugation for lymphoblastoid cells) and lysed them by sonication in a lysis buffer containing 60 mM -glycerophosphate, 15 mM p-nitrophenylphosphate, 25 mM 3-(N-morpholino)propanesulfonic acid (Mops) (pH 7.2), 15 mM EGTA, 15 mM MgCl2, 1 mM Na vanadate, 1 mM NaF, 1mM phenylphosphate, 0.1% Nonidet P-40, and a protease inhibitor mixture (Roche). We spun the lysates 15 min at 20,000 × g at 4 °C, collected the supernatants, and determined the protein content using a Bradford assay. We performed the immunoprecipitation experiments on 500 g of total

proteins, in lysis buffer. We precleared the lysates with 20 L of protein A or G-agarose beads, incubated 1 h 4 °C on a rotating wheel. We added 5 g of antibody to the supernatants, incubated 1 h 4 °C on a rotating wheel, added 20 L of protein A or G-agarose beads, and incubated 1 h 4 °C on a rotating wheel. We collected the beads by centrifugation 30 s at 18,000 × g at 4 °C and washed three times in a bead buffer containing 50 mM Tris (pH 7.4), 5 mM NaF, 250 mM NaCl, 5 mM EDTA, 5 mM EGTA, 0.1% Nonidet P-40, and a protease inhibitor coktail (Roche). We directly added sample buffer to the washed pellets, heat-denatured the proteins, and ran the samples on 10% Bis-Tris SDS/PAGE. We transferred the proteins onto Hybond nitrocellulose membranes and processed the blots according to standard procedures. For Western blot experiments, we used the following primary antibodies: anti-Myc (Abcam ab9106, 1:2,000), anti-V5 (Invitrogen R960, 1:5,000), anti-tubulin (Santa Cruz Biotechnology B-7, 1:500), anti-CDK10 (Covalab pab0847p, 1:500 or Santa Cruz Biotechnology C-19, 1:500), anti-CycM (home-made, dilution 1:500 or Covalab pab0882-P, dilution 1:500), anti-Raf1 (Santa Cruz Biotechnology C-20, 1:1,000), anti-ETS2 (Santa Cruz Biotechnology C-20, 1:1,000), anti-Flag (Sigma F7425, 1:1,000), and anti-actin (Sigma A5060, 1:5,000). We used HRP-coupled anti-goat (Santa Cruz Biotechnology SC-2033, dilution 1:2,000), anti-mouse (Bio-Rad 170-6516, dilution 1:3,000) or anti-rabbit (Bio-Rad 172-1019, 1:5,000) as secondary antibodies. We revealed the blots by enhanced chemiluminescence (SuperSignal West Femto, Thermo Scientific).Production and Purification of Recombinant Proteins.GST-CDK10(kd)/StrepII-CycM. We generated recombinant bacmids in DH10Bac Escherichia coli and baculoviruses in Sf9 cells using the Bac-to-Bac system, as described by the provider (Invitrogen). We infected Sf9 cells with GST-CDK10- (or GST-CDK10kd)-producing viruses, or coinfected the cells with StrepII-CycM-producing viruses, and we collected the cells 72 h after infection. To purify GST-fusion proteins, we spun 250 mL cells and resuspended the pellet in 40 mL lysis buffer (PBS, 250 mM NaCl, 0.5% Nonidet P-40, 50 mM NaF, 10 mM -glycerophosphate, and 0.3 mM Na-vanadate) containing a protease inhibitor mixture (Roche). We lysed the cells by sonication, spun the lysate 30 min at 15,000 × g, collected the soluble fraction, and added it to a 1-mL glutathione-Sepharose matrix. We incubated 1 h at 4 °C, washed four times with lysis buffer, one time with kinase buffer A (see below), and finally resuspended the beads in 100 L kinase buffer A containing 10% (vol/vol) glycerol for storage.6His-ETS2. We transformed Origami2 DE3 (Novagen) with the 6His-ETS2 expression vector. We induced expression with 0.2 mM isopropyl- -d-1-thiogalactopyranoside for 3 h at 22 °C. To purify 6His-ETS2, we spun 50 mL cells and resuspended the pellet in 2 mL lysis buffer (PBS, 300 mM NaCl, 10 mM Imidazole, 1 mM DTT, and 0.1% Nonidet P-40) containing a protease inhibitor mixture without EDTA (Roche). We lysed the cells at 1.6 bar using a cell disruptor and spun the lysate 10 min at 20,000 × g. We collected the soluble fraction and added it to 200 L Cobalt beads (Thermo Scientific). After 1 h incubation at 4 °C on a rotating wheel, we washed four times with lysis buffer. To elute, we incubated beads 30 min with elution buffer (PBS, 250 mM imidazole, pH 7.6) containing the protease inhibitor mixture, spun 30 s at 10,000 × g, and collected the eluted protein.Protein Kinase Assays. We mixed glutathione-Sepharose beads (harboring GST-CDK10 wt or kd, either monomeric or complexed with StrepII-CycM), 22.7 M BSA, 15 mM DTT, 100 M ATP, 5

Ci ATP[-32P], 7.75 M histone H1, or 1 M 6His-ETS2 and added kinase buffer A (25 mM Tris·HCl, 10 mM MgCl2, 1 mM EGTA, 1 mM DTT, and 3.7 M heparin, pH 7.5) up to a total volume of 30 L. We incubated the reactions 30 min at 30 °C, added Laemli sample buffer, heat-denatured the samples, and ran 10% Bis-Tris SDS/PAGE. We cut gel slices to detect GST-CDK10 and StrepII-CycM by Western blotting. We stained the gel slices containing the substrate with Coomassie (R-250, Bio-Rad), dried them, and detected the incorporated radioactivity by autoradiography. We identified four unrelated girls with anogenital and renal malformations, dysmorphic facial features, normal intellect and syndactyly of toes. A similar combination of features had been reported previously in a mother-daughter pair1 (Table 1 and Supplementary Note online). These authors noted clinical overlap with Townes-Brocks syndrome but suggested that the phenotype represented a separate autosomal dominant entity (MIM601446). Here we define the cardinal features of this syndrome as a characteristic facial appearance with apparent telecanthus and broad tripartite nasal tip, variable syndactyly of toes 2-5, hypoplastic labia, anal atresia and urogenital malformations (Fig. 1a-h). We also observed a variety of other features (Table 1). Figure 1: Clinical and molecular characterization of STAR syndrome. Figure 1 : Clinical and molecular characterization of STAR syndrome. (a-f) Facial appearances of cases 1-3 (apparent telecanthus, dysplastic ears and thin upper lips; a,c,e), and toe syndactyly 2-5, 3-5 or 4-5 (b,d,f) in these cases illustrate recognizable features of STAR syndrome (specific parental consent has been obtained for publication of these photographs). Anal atresia and hypoplastic labia are not shown. (g,h) X-ray films of the feet of case 2 showing only four rays on the left and delta-shaped 4th and 5th metatarsals on the right (h; compare to clinical picture in d). (i) Array-CGH data. Log2 ratio represents copy number loss of six probes spanning between 37.9 and 50.7 kb, with one probe positioned within FAM58A. The deletion does not remove parts of other functional genes. (j) Schematic structure of FAM58A and position of the mutations. FAM58A has five coding exons (boxes). The cyclin domain (green) is encoded by exons 2-4. The horizontal arrow indicates the deletion extending 5' in case 1, which includes exons 1 and 2, whereas the horizontal line below exon 5 indicates the deletion found in case 3, which removes exon 5 and some 3' sequence. The pink horizontal bars above the boxes indicate the amplicons used for qPCR and sequencing (one alternative exon 5 amplicon is not indicated because of space constraints). The mutation 201dupT (case 4) results in an immediate stop codon, and the 555+1G>A and 555-1G>A splice mutations in cases 2, 5 and 6 are predicted to be deleterious because they alter the conserved splice donor and acceptor site of intron 4, respectively. Full size image (97 KB) Table 1: Clinical features in STAR syndrome cases Table 1 - Clinical features in STAR syndrome cases Full table On the basis of the phenotypic overlap with Townes-Brocks, Okihiro and Feingold syndromes, we analyzed SALL1 (ref. 2), SALL4 (ref. 3) and MYCN4 but found no mutations in any of these genes (Supplementary Methods online). Next, we carried out genome-wide high-resolution oligonucleotide array comparative genomic hybridization (CGH)5 analysis (Supplementary Methods) of genomic DNA from the most severely affected individual (case 1, with lower lid coloboma, epilepsy and syringomyelia) and identified a heterozygous deletion of 37.9-50.7 kb on Xq28, which removed exons 1 and 2 of FAM58A (Fig. 1i,j). Using real-time

PCR, we confirmed the deletion in the child and excluded it in her unaffected parents (Supplementary Fig. 1a online, Supplementary Methods and Supplementary Table 1 online). Through CGH with a customized oligonucleotide array enriched in probes for Xq28, followed by breakpoint cloning, we defined the exact deletion size as 40,068 bp (g.152,514,164_152,554,231del(chromosome X, NCBI Build 36.2); Fig. 1j and Supplementary Figs. 2,3 online). The deletion removes the coding regions of exons 1 and 2 as well as intron 1 (2,774 bp), 492 bp of intron 2, and 36,608 bp of 5' sequence, including the 5' UTR and the entire KRT18P48 pseudogene (NCBI gene ID 340598). Paternity was proven using routine methods. We did not find deletions overlapping FAM58A in the available copy number variation (CNV) databases. Subsequently, we carried out qPCR analysis of the three other affected individuals (cases 2, 3 and 4) and the mother-daughter pair from the literature (cases 5 and 6). In case 3, we detected a de novo heterozygous deletion of 1.1-10.3 kb overlapping exon 5 (Supplementary Fig. 1b online). Using Xq28-targeted array CGH and breakpoint cloning, we identified a deletion of 4,249 bp (g.152,504,123_152,508,371del(chromosome X, NCBI Build 36.2); Fig. 1j and Supplementary Figs. 2,3), which removed 1,265 bp of intron 4, all of exon 5, including the 3' UTR, and 2,454 bp of 3' sequence. We found heterozygous FAM58A point mutations in the remaining cases (Fig. 1j, Supplementary Fig. 2, Supplementary Methods and Supplementary Table 1). In case 2, we identified the mutation 555+1G>A, affecting the splice donor site of intron 4. In case 4, we identified the frameshift mutation 201dupT, which immediately results in a premature stop codon N68XfsX1. In cases 5 and 6, we detected the mutation 556-1G>A, which alters the splice acceptor site of intron 4. We validated the point mutations and deletions by independent rounds of PCR and sequencing or by qPCR. We confirmed paternity and de novo status of the point mutations and deletions in all sporadic cases. None of the mutations were seen in the DNA of 60 unaffected female controls, and no larger deletions involving FAM58A were found in 93 unrelated array-CGH investigations. By analyzing X-chromosome inactivation (Supplementary Methods and Supplementary Fig. 4 online), we found complete skewing of X inactivation in cases 1 and 3-6 and almost complete skewing in case 2, suggesting that cells carrying the mutation on the active X chromosome have a growth disadvantage during fetal development. Using RT-PCR on RNA from lymphoblastoid cells of case 2 (Supplementary Fig. 2), we did not find any aberrant splice products as additional evidence that the mutated allele is inactivated. Furthermore, FAM58A is subjected to X inactivation6. In cases 1 and 3, the parental origin of the deletions could not be determined, as a result of lack of informative SNPs. Case 5, the mother of case 6, gave birth to two boys, both clinically unaffected (samples not available). We cannot exclude that the condition is lethal in males. No fetal losses were reported from any of the families. The function of FAM58A is unknown. The gene consists of five coding exons, and the 642-bp coding region encodes a protein of 214 amino acids. GenBank lists a mRNA length of 1,257 bp for the reference sequence (NM_152274.2). Expression of the gene (by EST data) was found in 27 of 48 adult tissues including kidney, colon, cervix and uterus, but not heart (NCBI expression viewer, UniGene Hs.496943). Expression was also noted in 24 of 26 listed tumor tissues as well as in embryo and fetus. Genes homologous to FAM58A (NCBI HomoloGene: 13362) are found on the X chromosome in the chimpanzee and the

dog. The zebrafish has a similar gene on chromosome 23. However, in the mouse and rat, there are no true homologs. These species have similar but intronless genes on chromosomes 11 (mouse) and 10 (rat), most likely arising from a retrotransposon insertion event. On the murine X chromosome, the flanking genes Atp2b3 and Dusp9 are conserved, but only remnants of the FAM58A sequence can be detected. FAM58A contains a cyclin-box-fold domain, a protein-binding domain found in cyclins with a role in cell cycle and transcription control. No human phenotype resulting from a cyclin gene mutation has yet been reported. Homozygous knockout mice for Ccnd1 (encoding cyclin D1) are viable but small and have reduced lifespan. They also have dystrophic changes of the retina, likely as a result of decreased cell proliferation and degeneration of photoreceptor cells during embryogenesis7, 8. Cyclin D1 colocalizes with SALL4 in the nucleus, and both proteins cooperatively mediate transcriptional repression9. As the phenotype of our cases overlaps considerably with that of Townes-Brocks syndrome caused by SALL1 mutations1, we carried out co-immunoprecipitation to find out if SALL1 or SALL4 would interact with FAM58A in a manner similar to that observed for SALL4 and cyclin D1. We found that FAM58A interacts with SALL1 but not with SALL4 (Supplementary Fig. 5 online), supporting the hypothesis that FAM58A and SALL1 participate in the same developmental pathway. How do FAM58A mutations lead to STAR syndrome? Growth retardation (all cases; Table 1) and retinal abnormalities (three cases) are reminiscent of the reduced body size and retinal anomalies in cyclin D1 knockout mice7, 8. Therefore, a proliferation defect might be partly responsible for STAR syndrome. To address this question, we carried out a knockdown of FAM58A mRNA followed by a proliferation assay. Transfection of HEK293 cells with three different FAM58A-specific RNAi oligonucleotides resulted in a significant reduction of both FAM58A mRNA expression and proliferation of transfected cells (Supplementary Methods and Supplementary Fig. 6 online), supporting the link between FAM58A and cell proliferation. We found that loss-of-function mutations of FAM58A result in a rather homogeneous clinical phenotype. The additional anomalies in case 1 are likely to result from an effect of the 40-kb deletion on expression of a neighboring gene, possibly ATP2B3 or DUSP9. However, we cannot exclude that the homogeneous phenotype results from an ascertainment bias and that FAM58A mutations, including missense changes, could result in a broader spectrum of malformations. The genes causing the overlapping phenotypes of STAR syndrome and Townes-Brocks syndrome seem to act in the same pathway. Of note, MYCN, a gene mutated in Feingold syndrome, is a direct regulator of cyclin D2 (refs. 10,11); thus, it is worth exploring whether the phenotypic similarities between Feingold and STAR syndrome might be explained by direct regulation of FAM58A by MYCN. FAM58A is located approximately 0.56 Mb centromeric to MECP2 on Xq28. Duplications overlapping both MECP2 and FAM58A have been described and are not associated with a clinical phenotype in females12, but no deletions overlapping both MECP2 and FAM58A have been observed to date13. Although other genes between FAM58A and MECP2 have been implicated in brain development, FAM58A and MECP2 are the only genes in this region known to result in X-linked dominant phenotypes; thus, deletion of both genes on the same allele might be lethal in both males and females.

[ ]:

	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…

[ ]:
```
stop_words = set(stopwords.words('english'))
def preprocess_text(description, index, column):
    if type(description) is not int:
        string = ""
        # Replacing every special character with space
        # excluding ^
        description = re.sub('[^a-zA-Z0-9\n]', ' ', description)
        # Replace mutiple space with single space
        description = re.sub('\s+',' ', description)
        # converting all the characters to lower case to maintain consistency␣
 ↪of the Dataset.
        description = description.lower()

        # removing the stopwords
        for word in description.split():
            if word not in stop_words:
                string += word + " "

        # replacing the text with string after removing the stopwords
        data_text[column][index] = string

print(stop_words)
```

```
{'off', 'wouldn', 'and', 'each', 'any', 'having', "that'll", "aren't", 'with',
'itself', 'be', 'i', 'has', 'o', 'them', 'should', 'then', 'here', 't',
'theirs', 'between', 'her', "didn't", 'once', 'as', "hadn't", 'in', 'ma', 'y',
'they', 'only', 'for', "you've", 'yours', 'his', 'just', 'of', "shouldn't",
'up', 'because', 'that', 'its', 'does', 'shouldn', 'himself', 'at', "wouldn't",
'hasn', 'won', 'such', 'herself', 'whom', "needn't", 'my', 'are', "mustn't",
'myself', "couldn't", 'you', 'doesn', 'very', 'than', 'don', 'both', "wasn't",
"shan't", 'weren', 'when', 'below', 'can', 'm', 'your', 'the', 'hers',
'themselves', 'this', 'a', 'other', 'didn', 'nor', 'to', "she's", 'until', 's',
'ain', 'an', 'she', 'ours', "you'll", "you're", 'now', 'have', 'it', 'not',
'he', 'on', "you'd", 'again', 'aren', 'haven', 'yourself', 'during', 'shan',
'same', "isn't", 'further', 'mightn', 'if', 'will', 'couldn', "mightn't",
'mustn', 'while', 'wasn', 'some', 'we', 'isn', 'those', 'these', 'where', 'him',
'ourselves', 'down', 'd', 'into', 'did', 'me', "won't", 'above', 'no', 'had',
'too', 'being', 'who', "don't", 're', 'few', 've', 'after', 'all', 'how',
"it's", 'or', "hasn't", 'from', 'am', "doesn't", 'is', 'over', 'more', 'but',
'most', 'before', 'doing', 'do', 'hadn', 'yourselves', 'there', 'about', 'own',
```

```
'll', 'out', "haven't", 'was', 'what', 'by', 'which', 'been', 'so', 'their',
'under', 'were', 'needn', "should've", 'why', 'against', 'our', "weren't",
'through'}
```

[ ]:
```
start_time = time.perf_counter()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        # preprocessing the every text values
        preprocess_text(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
end_time = time.perf_counter()
print('Time took for preprocessing the text :',end_time - 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 : 50.70352443200001 seconds
```

[ ]:
```
# after preprocessing the text feature will look like this
data_text['TEXT'][2]
```

[ ]:
```
'abstract background non small cell lung cancer nsclc heterogeneous group
disorders number genetic proteomic alterations c cbl e3 ubiquitin ligase adaptor
molecule important normal homeostasis cancer determined genetic variations c cbl
relationship receptor tyrosine kinases egfr met functionality nsclc methods
findings using archival formalin fixed paraffin embedded ffpe extracted genomic
dna show c cbl mutations occur somatic fashion lung cancers c cbl mutations
mutually exclusive met egfr mutations however independent p53 kras mutations
normal tumor pairwise analysis significant loss heterozygosity loh c cbl locus
22 n 8 37 none samples revealed mutation remaining copy c cbl c cbl loh also
positively correlated egfr met mutations observed samples using select c cbl
somatic mutations s80n h94y q249e w802 obtained caucasian taiwanese african
american samples respectively transfected nsclc cell lines increased cell
viability cell motility conclusions taking overall mutation rate c cbl
combination somatic missense mutation loh clear c cbl highly mutated lung
cancers may play essential role lung tumorigenesis metastasis go introduction us
alone year approximately 219 400 people diagnosed lung cancers 145 000 succumb
disease 1 number roughly equivalent combined mortality rates cancers breast
prostate colon liver kidney melanoma 1 addition prognosis usually poor five year
survival rate less 15 also significant ethnic differences lung cancer outcome
worse blacks compared whites gender differences also striking women
significantly better prognosis compared men number genetic alterations occur
lung cancer example nsclc mutations kras p53 egfr met identified many pathways
especially receptor tyrosine kinases rtks controlled c cbl cbl casitas b lineage
```

lymphoma mammalian gene located human chromosome 11q23 3 2 involved cell signaling protein ubiquitination 3 cbl proteins belong ring finger class ubiquitin ligases e3 three homologues c cbl cbl b cbl 3 4 c cbl cbl b genes ubiquitously expressed highest levels hematopoietic tissues 5 c cbl consists four regions encoding functionally distinct protein domains n terminal tyrosine kinase binding tkb domain linker region catalytic ring finger domain proline rich region c terminal ubiquitin associated uba domain also overlaps leucine zipper lz domain 3 tkb ring finger domains essential ligand induced ubiquitination rtks 6 7 8 9 ring finger domain required recruitment e2 ubiquitin conjugating enzymes tkb domain includes four helix bundle 4h calcium biding ef hand modified sh2 domain binds phosphotyrosine residues 3 10 11 12 addition proline rich region c cbl associate sh3 domain grb2 indirectly recruit c cbl rtks via grb2 adaptor protein 7 13 14 c cbl also binds egfr acts e3 targets egfr ubiquitination degradation furthermore cbl desensitizes egf signaling opposes cellular proliferation induced egf 15 egf activation also appears activate tyrosine kinase src phosphorylates c cbl turn activates ubiquitination degradation egfr 16 17 18 recent study shows defective endocytosis egfr characterized deletion mutant point mutation l858r whereby association c cbl subsequent ubiquitination impaired 19 recently first human c cbl mutations reported acute myeloid leukemia aml patients 20 mutation r420q inhibits fms like tyrosine kinase 3 flt3 internalization ubiquitination 20 e3 activity important oncogenesis c cbl dual separate function signal transduction molecule previously shown c cbl important binding crkl bcr abl hematopoietic cells also bind modulate functions cytoskeleton binding proteins like talin paxillin tkb domain important binding number molecules function signal transduction given critical role cbl normal homeostasis cancer hypothesized might mutated lung cancers study report novel c cbl somatic mutations s80n h94y q249e w802 caucasian taiwanese african american lung cancer patients respectively expressing mutations nsclc cell lines lead increased proliferation cell motility show c cbl mutations occur without met egfr mutations mutually exclusive loh c cbl locus additionally c cbl loh associated either met egfr mutations thus hypothesize c cbl mutations might contribute oncogenic potential met egfr lung cancer go methods ethics statement written consent research human subjects obtained institutional review board university chicago covers research performed laboratory following contact information institutional review board university chicago mcgiffert hall 5751 woodlawn ave 2nd floor chicago il 60637 written informed consents received patients whose tissue samples used study tissue samples lung cancer tissue paired adjacent normal lung tissues obtained 50 caucasian 29 african americans 40 taiwanese nsclc patients recruited university chicago hospital chicago usa caucasian african american patients taipei veterans general hospital taiwan taiwanese patients obtaining appropriate institutional review board permission informed consent patients 119 samples 77 men 38 women 4 unknown age diagnosis ranging 47 90 years terms tumor types 53 adenocarcinoma 32 squamous cell carcinoma 34 large cell carcinoma 49 stage 14 stage ii 34 stage iii 13 stage iv table s1 cell culture human non small cell lung carcinoma cells a549 h358 maintained dmem rpmi 1640 respectively human embryonic kidney 293t cells cultured dmem media supplemented 10 fetal bovine serum 100 units ml penicillin

100 g ml streptomycin invitrogen carlsbad ca cells cultured 37 c humidified incubator containing 5 co2 c cbl gene mutational analysis exons 2 16 c cbl gene individually amplified polymerase chain reaction pcr primers listed table s2 pcr conditions 1 cycle 95 c 5 minutes 35 cycles 94 c 30 seconds 58 c 30 seconds 72 c 2 minutes one cycle 72 c 10 minutes pcr products treated exosap usb corporation cleveland oh sequenced big dye terminator chemistry applied biosystems foster city ca sequencing performed forward coding strand confirmation c cbl alterations performed sequencing reverse strand well chromatograms analyzed mutations using mutation surveyor v2 61 softgenetics state college pa plasmid constructs site directed mutagenesis wild type c cbl cdna insert subcloned paltermax expression vector using xhoi sali restriction enzyme sites promega madison wi using parental plasmid paltermax c cbl tkb domain double mutation s80n h94y point mutation q249e c terminal point mutation w802 c cbl created using following primers 5 gctggcgctaaagaataacccaccttatatcttagac 3 5 ctaccagatacctaccagtatctccgtactatcttgtc 3 double mutation s80n h94y 5 ctttacccgactctttgagccctggtcctctttgc 3 q249e 5 cagctcctcctttggctgattgtctctggatggtgatc 3 w802 along complementary primers using quickchange site directed mutagenesis xl kit stratagene la jolla ca according manufacturer instructions constructs confirmed point mutations standard dna sequencing strands loss heterozygosity loh analysis five microsatellites chromosome 11 3 11q within 200 kb downstream c cbl gene 2 control markers 11p selected analysis table s3 established microsatellite markers respective primer sequences selected geneloc database http genecards weizmann ac il geneloc index shtml weizmann institute science rehovot israel primers custom designed forward primer fluorescently labeled 5 end fam pet ned vic applied biosystems primer annealing temperatures duplex scores evaluated nist primer tools http yellow nist gov 8444 dnaanalysis primertoolspage national institute standards technology gaithersburg md primers verified performing pcr control dna isolated tk6 cells resolving products agarose gels bands visualized uv transilluminator genomic dna extracted tumor samples paired normal lung tissue primers grouped multiplex combinations shown table s4 marker d11s929 served internal control check consistency pcrs peaks capillary electrophoresis multiplex pcrs carried volume 10 l contained 1 l genomic dna 20 50 ng 0 5 primer 1 0 total primer pair 400 dntps 1x pcr buffer containing mgcl2 0 2 u taq dna polymerase pcr performed abi geneamp 9700 pcr system following conditions 5 min 94 c 30 cycles 30 sec 94 c 1 min 60 c 1 min 72 c 5 min 72 c pcr products separated capillary electrophoresis abi 3130xl dna analyzer chromatograms analyzed peak scanner 1 0 genemapper 3 7 software applied biosystems allelic alterations area peaks produced dna pcr products quantified allele ratio allelic areas calculated tumor paired normal dna sample qloh allelic ratio tumor peaks divided allelic ratio paired normal sample 0 5 2 0 c cbl least one 11q marker least two separate experiments sample considered allelic imbalance interpreted loh samples evaluated least two separate experiments samples showing prospective loh c cbl repeated third time included new control marker bax locus data shown chromosome 19 verify integrity sample dna transfection c cbl constructs a549 cell line transfected using fugene hd roche nutley nj reagent according manufacturer instructions eight g plasmid dna containing either insert empty vector wild type

c cbl s80n h94y c cbl q249e c cbl w802 cbl used transfection 6 well culture plate cells harvested 48 h transfection analyzed expression c cbl knockdown c cbl knockdown performed using lentiviral transduction using mission lentiviral transduction particles sigma aldrich st louis mo per manufacturer instructions briefly 1 105 h358 cells well seeded 6 well plates infected following day c cbl lentiviral shrna constructs generate stable c cbl knockdown cell lines cells selected 2 days 1 g ml puromycin c cbl levels determined using whole cell lysates immunoblotting anti cbl antibody santa cruz biotechnologies santa cruz ca cell viability assay cells transfected described transfection assay forty eight hours transfection viability cells assessed using trypan blue exclusion wound healing assay a549 cells seeded 6 well plates cultured 48 h 100 confluent medium changed cells transfected described transfection assay twelve hours transfection straight scratch made across cell layer using 1 ml pipette tip cells gently washed 1 pbs remove cellular debris media replaced photographs taken wound region every 12 h 48 h western blot analysis forty eight hours transfection cells collected washed twice 1x pbs lysed ice cold lysis buffer 0 5m tris hcl ph 7 4 1 5 nacl 2 5 deoxycholic acid 10 mm edta 10 np 40 0 5 mm dtt 1 mm phenylmethylsulfonyl fluoride 5 g ml leupeptin 10 g ml aprotinin 5 minutes lysate centrifuged 13 000 rpm 20 minutes 4 c protein content supernatant measured total cell lysates 50 g well separated sds page electrophoresis gels transferred onto nitrocellulose membranes whatman piscataway nj membranes blocked 5 non fat dry milk phosphate buffered saline containing tween 20 pbst 1x pbs 0 1 tween 20 1 h room temperature incubated appropriate primary antibody 4 c overnight membranes washed three times pbst probed appropriate horseradish peroxidase hrp conjugated secondary antibody 1 h room temperature membranes washed three times pbst bands visualized using western blot chemiluminescence reagent biorad valencia ca chemidoc gel documentation system biorad valencia ca antibodies obtained santa cruz biotechnologies used following dilutions c cbl 1 5000 c met 1 5000 egfr 1 5000 ubiquitin 1 1000 ha 1 5000 actin 1 10 000 flow cytometry cell cycle analysis carried flow cytometry approximately 2 106 cells grown media containing 10 fbs cells harvested trypsin edta treatment washed 1x pbs three times fixed ice cold 70 ethanol 2 h cells washed cold pbs stained solution containing 25 g ml propidium iodide 200 g ml rnase 0 1 triton x 100 30 minutes dark cell cycle analysis performed using guava pca 96 flow cytometer guava technologies millipore billerica ubiquitin ligase activity 293t cells maintained culture dmem supplemented 10 fbs 1 penicillin 100 units ml streptomycin 100 g ml transfected 0 2 g egfr pcdna3 2 g ha tagged c cbl constructs indicated using calcium phosphate according manufacturer protocol profection promega madison wi twenty four hours post transfection cells starved overnight dmem supplemented 0 5 fbs treated without egf 100 ng ml 15 min cells collected washed two times ice cold pbs containing 0 2 mm sodium orthovanadate lysed ice cold lysis buffer 10 mm tris hcl ph 7 5 150 mm nacl 5 mm edta 1 triton x100 10 glycerol 2 mm sodium orthovanadate protease inhibitors lysates cleared debris centrifugation 16 000 g 10 min 4 c egfr immunoprecipitations performed 200 g cleared lysate using 250 ng rabbit anti egfr protein g plus sepharose overnight 4 c precipitations washed 5 times lysis buffer boiling laemmli buffer elutions immunoblotted anti ubiquitin egfr twenty micrograms cleared lysate

immunoblotted c cbl constructs using anti ha statistical analysis mutation rates different groups compared using fisher exact test continuous variables group comparisons performed using analysis variance anova followed sidak adjustment multiple comparisons experiments involving repeated measurements time analyzed using repeated measures anova greenhouse geisser adjustment degrees freedom analyses conducted using stata v10 1 software stata corporation college station tx go results c cbl gene mutations lung cancer investigate role c cbl lung cancer analyzed genomic dna tumor paired normal samples drawn multiple ethnicities lung tumor samples represented caucasians n 50 african americans n 29 taiwanese n 40 lung cancer patients designed 12 pairs primers sequence coding region c cbl gene spans exons 2 16 table s2 identified 8 unique somatic mutations c cbl exons among 8 different patients variation l620f known snp rs2227988 exon 11 also detected importantly eight novel non synonymous mutations confirmed sequencing strands c cbl genomic dna obtained lung tumor samples table 1 moreover none 8 mutations detected corresponding normal tissue indicating somatic mutations four synonymous single nucleotide variations snvs also identified used study table 1 table 1 c cbl mutation analysis 119 lung cancer patient tumor tissues three 8 novel non synonymous mutations located tkb tyrosine kinase binding domain s80n h94y q249e one ring finger domain v391i one proline rich region 72515 72517 del atg three c terminal region w802 r830k a848t c cbl protein figure 1a figure s1 figure 1b show model chromatograms representative samples figure 1 figure 1 c cbl mutations loh non small cell lung cancer 11q loh c cbl gene paired lung tumor normal lung tissue samples taiwanese patients n 37 investigated loh eight 21 6 showed loh c cbl locus chromosome 11 29 samples 78 4 revealed normal allelic contribution microsatellite markers figures 1c c cbl mutations different ethnic groups c cbl double mutant s80n h94y found patient overall mutation rate c cbl lung tumors 6 7 8 119 frequency c cbl mutation highest large cell carcinoma 14 7 5 34 patients followed squamous carcinoma 6 3 2 32 patients least observed adenocarcinoma ad 1 8 1 53 patients although rates statistically significant p 0 292 mutation rates 6 0 among caucasians 0 20 ad 0 10 sq 3 20 lc 13 8 african americans 1 10 ad 1 10 sq 2 9 lc 2 5 0 23 ad 1 12 sq 0 5 lc taiwanese population additionally two taiwanese patients lung cancer one squamous one adenocarcinoma known snp l620f ethnic differences statistically significant however power detect differences low mutations met egfr co associated c cbl alterations since east asians lung cancer higher frequency egfr met mutations lung tumors 21 22 also determined mutations egfr met taiwanese cohort samples compared results observed c cbl alterations loh mutations 37 samples tested find overlap c cbl mutations c cbl loh figure 2 three c cbl mutants including known l620f snp rs2227988 one samples met mutation n375s egfr mutation l858r among 8 samples loh c cbl locus 5 additional mutation met n375s 2 egfr exon 19 deletion twenty six samples neither c cbl mutation c cbl loh 3 patients c cbl mutation c cbl loh among 26 samples 9 met mutation 8 n375s 1 l211w 13 egfr mutation 7 exon 9 deletion 6 l858r 4 met egfr mutation thus rate met egfr mutations among patients loh c cbl locus 7 8 similar seen patients without c cbl mutation loh 22 26 patients p 0 99 4 patients identifiable mutation c cbl met egfr represented 10 8 37 patients analyzed taiwanese patient cohort conversely 89 2 taiwanese lung cancer patients

identifiable mutation either c cbl met egfr combination three genes figure 2 additionally determined p53 kras mutations taiwanese cohorts two p53 1 kras mutation detected single kras mutation overlapped one p53 mutation patient also egfr exon 19 deletion c cbl mutation p53 mutation sample c cbl loh concurrent met n375s mutation thus taiwanese samples analyzed p53 kras mutations c cbl mutations mutually exclusive data shown figure 2 figure 2 c cbl mutations relationship met egfr mutations lung cancer cellular functions c cbl alterations context lung tumorigenesis e3 activity intact mutant c cbl proteins investigate whether different c cbl mutations affect e3 activity egfr chosen model substrate c cbl e3 function c cbl mutants tested enhanced ubiquitination activated egfr similar wild type c cbl protein result demonstrates catalytic activity c cbl mutants impaired egfr substrate figure 3a figure 3 figure 3 ubiquitination viability expression cell cycle analysis various c cbl mutants b effect lung cancer cell viability effect representative c cbl mutant three ethnic backgrounds lung cancer cell viability cell lines determined s80n h94y double mutation q249e w802 identified lung tumor samples obtained caucasian taiwanese african american respectively described methods c cbl wild type wt three mutants expressed cloning paltermax vector a549 cells cells express relatively low basal levels endogenous c cbl data shown transfection efficiency comparable different groups number cells transfected c cbl wild type construct 70 compared control cells transfected empty vector cells transfected s80n h94y q249e w802 c cbl mutant constructs resulted increased number viable cells 132 3 120 8 147 9 higher respectively relative empty vector control transfected cells significantly different wild type construct p 0 022 p 0 049 p 0 008 respectively figure 3b relative levels c cbl protein whole cell lysates prepared samples obtained parallel experiment determined c cbl protein levels samples representing untransfected empty vector transfected cells comparable representing c cbl wt three c cbl mutants comparable figure 3c c effect cell cycle investigate increases cell viability different c cbl mutants due increased cellular proliferation cell cycle analysis performed a549 cells transfected c cbl wt three different mutants s80n h94y q249e w802 empty vector transfectant used control forty eight hours transfection cell cycle analysis performed described materials methods significant change subg1 g1 phase cell cycle among different mutants compared wt construct p 0 64 p 0 40 p 0 28 respectively g2 phase cell cycle showed increase cell numbers three mutants s80n h94y q249e w802 compared wt difference statistically significant p 0 25 figure 3d effect cell motility investigate effect expression three c cbl mutants cell migration carried wound healing assay described materials methods closing scratch wound monitored 0 12 24 36 48 h figure 4a samples represented cells transfected mutants wound gap much smaller seen sample represented cells transfected c cbl wt p 0 001 also determined rate wound closure five groups 48 h wild type c cbl transfectants showed 61 1 open wound s80n h94y q249e w802 mutants showed 18 7 23 9 34 3 open wound respectively p 0 001 figure 4b figure 4 figure 4 c cbl mutations affect wound healing a549 cells e c cbl knockdown increases cell viability hypothesized loh seen samples could lead decreased expression c cbl thus tested effect c cbl knockdown lung cancer cells compared a549 h358 lung cancer cells express relatively high levels endogenous c cbl data shown c cbl

expression knocked using lentiviral construct expressed c cbl specific shrna compared results transduced scrambled shrna results shown figure 5 identified several clones revealed varying degrees c cbl knockdown showing different sets c cbl lentiviral shrna knockdown efficiency figure 5a clones tested clone 27 chosen experiments equal amount cells seeded 6 well plate cell proliferation measured various times results depicted figure 5b expected number cells increased time dependent fashion 100 190 relative scrambled shrna control span 48 h p 0 0002 figure 5b cell cycle phases h358 cells knocked c cbl shrna looked compared scrambled shrna discernable differences two constructs different phases cell cycle data shown figure 5 figure 5 knockdown c cbl using shrna increases cell proliferation go discussion results demonstrate c cbl somatically mutated loh lung cancers significantly contribute enhanced cell viability motility also high prevalence loh respect c cbl lung tumors harbored met egfr mutation present study demonstrated occurrence c cbl mutations lung cancer patients especially different ancestral variations mutations c cbl recently reported juvenile myelomonocytic leukemia myeloid malignancies aml study mutation r420q located junction ring finger linker region inhibited fms like tyrosine kinase 3 flt3 internalization ubiquitination 20 thus contributing gain function rtk addition mutations h398y c384r l380p mapped ring finger domain linker region c cbl required e3 activity 23 24 25 26 27 additionally homozygous mutations ring finger domain c cbl gene described result acquired uniparental disomy upd 26 important note results indicate loh 11q23 locus mutually exclusive missense mutations c cbl somatic mutations heterozygous mutations aml led abrogation e3 activity leading prolonged rtk activation addition mutants located linker region surrounding ring finger domain exhibited enhanced akt signaling response cytokine stimulation 26 addition shown nh3t3 cells neither mutations ring finger linker region causes transformation however certain mutations perturbs ubiquitination others affect receptor recycling prolong kinase activity 28 report c cbl mutations mapped ring finger domain also tkb domain proline rich domain c terminal region none mapped linker region reported aml studies described 23 24 25 26 29 addition 8 mutants detected found different ethnic backgrounds example s80n h94y q249e w802 detected caucasians taiwanese african americans respectively results point difference lung cancer cancers also genetic polymorphism among different races cancer interestingly large disparity african american ethnic populations lung cancer 30 previously shown low frequency egfr met mutation african americans compared taiwanese caucasians 31 study number african american samples analyzed relatively fewer found 3 mutations unique ethnicity would behoove us study genetic alterations occur determine targeted therapeutics african americans results provide evidence importance c cbl tumorigenesis potential signaling prediction based aml data would v391i ring finger domain mutation would affect e3 activity also important determine binding partners c cbl tkb domain proline rich domain mutations previously shown tkb domain bind growth factor receptors important determine cross binding mutants met egfr would also important future look fluorescence situ hybridization copy number changes c cbl lung cancer c cbl plays important role regulating rtk mediated signaling k63 poly ubiquitination subsequent downregulation rtks followed lysosomal degredation 3 mono ubiquitination ubiquitinated k63 linked

chains substrates c cbl may lead enhancement biological biochemical functions reviewed hermann et al 2007 32 mutations analyzed studies point fact e3 activity c cbl egfr intact egfr levels various mutants remain figure s2 multiple kinases rtks non rtks could acted upon c cbl including erbs pdgfr fms met c kit vegfr flt 1 ron fgfr ir well syk fyn lck fgr lyn c abl 3 lung cancers relevant substrates c cbl terms degradation signal transduction yet identified observation c cbl somatic mutations especially s80n h94y q249e w802 showed increased cell viability cell motility agreement physiological role cbl regulation apoptosis differentiation identified drosophila significant 33 previously shown activating c cbl mutation downregulates egfr signaling decreases cellular proliferation migration breast cancer cell lines 34 although role c cbl negative regulation rtks well substantiated thereby suggesting natural tumor suppressor studies cancer cells revealed tumor suppressor tumor promoting activities depending type c cbl mutation number alleles c cbl locus 24 agreement three c cbl mutants described appear tumor growth metastasis promoting properties although mutants outside ring finger linker region c cbl downstream effects significant cause increased proliferation migration substrate affected mutations known yet raises possibility cellular functions c cbl independent ubiquitin ligase activity area currently investigating oncogenic nature rtks addiction cancers growth signals given clustering c cbl egfr met mutations possible transforming effect c cbl mutations likely combinatorial effect three also show loh c cbl found significant number samples harbored met egfr mutations fact 7 lung tumor samples likely c cbl mutations additional 22 likely harbor c cbl related loh makes c cbl highly mutated molecule lung cancer since loh alone enough cause transforming event 35 36 37 associated mutation met egfr locus yet another rtk discussed may play role carcinogenesis predict loh c cbl results haploinsufficiency downplays rtk ubiquitination leading hyperactivity rtks however whether sufficient cause tumorigenesis remains determined consistent hypothesis fact c cbl mice increased kinase activity lymphocytes sufficient tumor formation 35 36 37 c cbl loh could also lead increased expression c cbl allele compensate loss allele alternately could form synergy working reduced c cbl levels mutated receptors exacerbate phenotype alone previous studies lab others shown east asians lung cancers relatively high frequencies gain function mutations rtks egfr met 31 cohort japanese patients activating met mutation identified splice region deletes juxtamembrane domain involved e3 activity c cbl 38 study also found activation met mutually exclusive egfr kras her2 gene mutations 38 failed detect mutations significant numbers lung tumor samples obtained african americans n 29 caucasian n 50 patients one met mutation identified groups whereas 1 3 egfr mutations identified african american caucasian cohorts respectively egfr mutations earlier identified one key mutations affecting lung adenocarcinoma patients comprehensive study 188 patients 39 study encompasses different histologies nsclc however published series find mutations c cbl met unlike study encompassed different subtypes nsclc important note recently shown met mutations lung cancer majority germline 31 reported earlier c cbl mutations small cohort taiwanese lung cancer samples 40 efforts understand ethnic differences lung oncogenome also looked pax transcription factors pax5 pax8 highly expressed lung cancers however

```
preferential expression mutations genes lung tumor samples african americans
study show relatively high frequency c cbl mutations lung cancers especially
large cell type among caucasians particularly among african americans therefore
propose c cbl efficacious target lung cancers african americans needs
substantiated important prognosis african americans lung cancer especially men
much poorer compared caucasian counterparts 41 conclusion results presented
study demonstrate c cbl frequently mutated even lost lung cancers results
support role c cbl mutants independent ubiquitination activity given relatively
high mutation rates c cbl well rtks met egfr likely combined effect could
synergistic promoting tumorigenesis '
```

[ ]:
```
type(data_text)
```

[ ]:
```
pandas.core.frame.DataFrame
```

[ ]:
```
data_text.head()
```

[ ]:
```
   ID                                               TEXT
0   0  cyclin dependent kinases cdks regulate variety…
1   1  abstract background non small cell lung cancer…
2   2  abstract background non small cell lung cancer…
3   3  recent evidence demonstrated acquired uniparen…
4   4  oncogenic mutations monomeric casitas b lineag…
```

[ ]:
```
# creating a new dataset result by combining both variants and text
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

[ ]:
```
   ID    Gene             Variation  Class  \
0   0  FAM58A  Truncating Mutations      1
1   1     CBL                 W802*      2
2   2     CBL                 Q249E      2
3   3     CBL                 N454D      3
4   4     CBL                 L399V      4

                                                TEXT
0  cyclin dependent kinases cdks regulate variety…
1  abstract background non small cell lung cancer…
2  abstract background non small cell lung cancer…
3  recent evidence demonstrated acquired uniparen…
4  oncogenic mutations monomeric casitas b lineag…
```

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

[ ]:
```
        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
```

[ ]:
```
# replacing the missing text values by concatinating the its gene and variation␣
 ↪text.
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +'␣
 ↪'+result['Variation']
```

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

[ ]:
```
         ID   Gene Variation  Class          TEXT
1109  1109  FANCA    S1088F      1  FANCA S1088F
```

3 Test, Train and Cross Validation Split

[ ]:
```
y_class = result['Class']
print(type(y_class))
y_class
```

```
<class 'pandas.core.series.Series'>
```

[ ]:
```
0       1
1       2
2       2
3       3
4       4
       ..
3316    4
3317    1
3318    1
3319    4
3320    4
Name: Class, Length: 3321, dtype: int64
```

[ ]:
```
y_class = result['Class'].values
print(type(y_class))
y_class
```

```
<class 'numpy.ndarray'>
```

[ ]:
```
array([1, 2, 2, …, 1, 4, 4])
```

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

[ ]:
```
X_train, test_df, Y_train, y_test = train_test_split(result, y_class, test_size␣
 ↪= 0.2, stratify = y_class)
train_df, cv_df, y_train, y_cv = train_test_split(X_train, Y_train, test_size =␣
 ↪0.2, stratify = Y_train)
# train_df -- y_train
# cv_df -- y_cv
# test_df -- y_test
print(train_df.shape)
train_df.head()
print(type(train_df))
print(type(y_train))
```

```
(2124, 5)
<class 'pandas.core.frame.DataFrame'>
<class 'numpy.ndarray'>
```

[ ]:
```
print("Number of datapoints in Training Dataset : ", train_df.shape[0])
print("Number of datapoints in Cross Validation dataset : ", cv_df.shape[0])
print("Number of datapoints in Test dataset : ", test_df.shape[0])
```

```
Number of datapoints in Training Dataset :  2124
Number of datapoints in Cross Validation dataset :  532
Number of datapoints in Test dataset :  665
```

[ ]:

[ ]:

Distribution of Class labels in Train, Test and Cross Validation datasets

[ ]:
```
train_df['Class'].value_counts()
```

[ ]:
```
7    609
4    439
1    363
2    289
6    176
5    155
3     57
9     24
8     12
Name: Class, dtype: int64
```

[ ]:
```
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cross_val_class_distribution = cv_df['Class'].value_counts().sort_index()
print(type(train_class_distribution))
print(train_class_distribution)
```

```
print(train_class_distribution.values)
print(type(train_class_distribution.values))
```

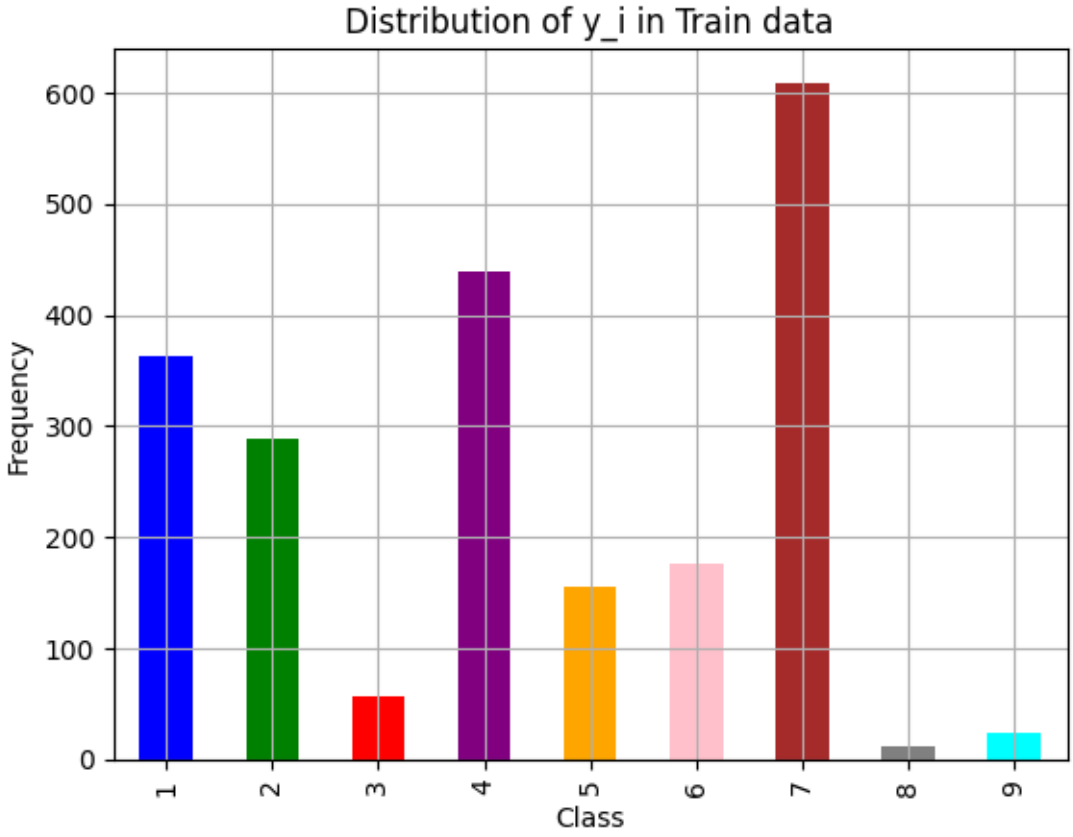
```
<class 'pandas.core.series.Series'>
1    363
2    289
3     57
4    439
5    155
6    176
7    609
8     12
9     24
Name: Class, dtype: int64
[363 289  57 439 155 176 609  12  24]
<class 'numpy.ndarray'>
```

3.0.1 Plotting the Distribution of Class labels

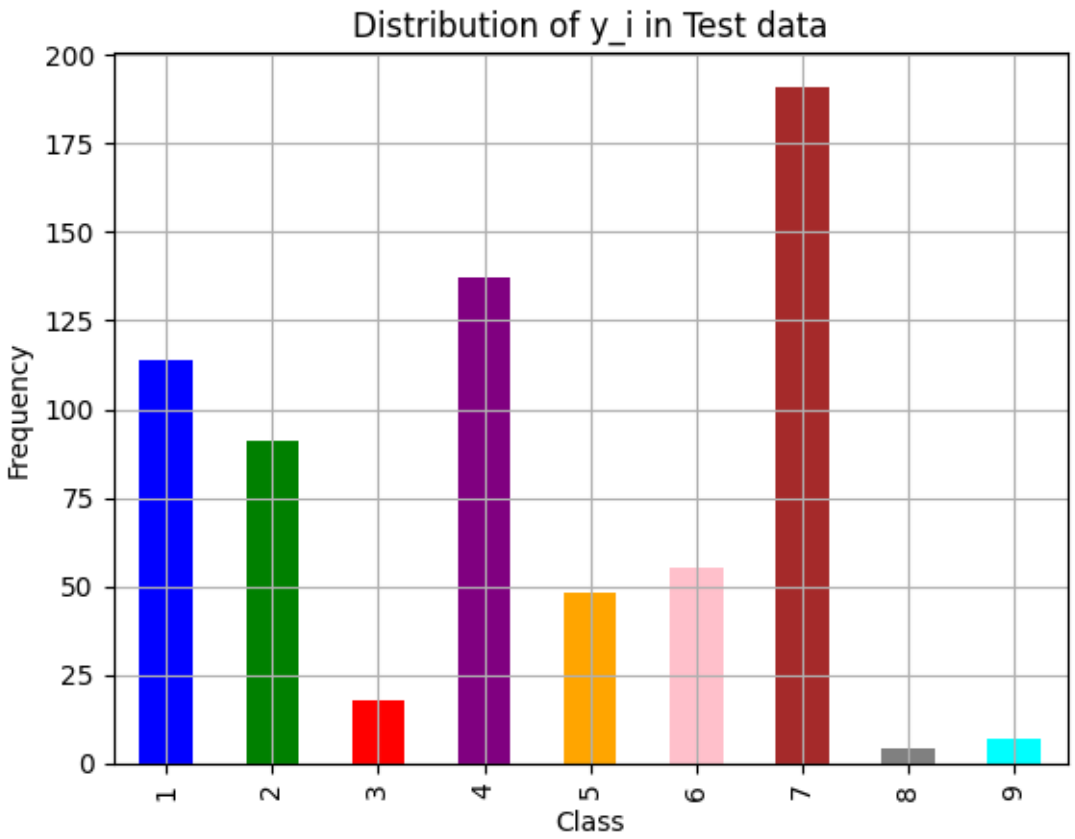
[ ]:
```
colors = ['blue', 'green', 'red', 'purple', 'orange', 'pink', 'brown', 'gray',␣
 ↪'cyan']
```

[ ]:
```
train_class_distribution.plot(kind = 'bar', xlabel = 'Class', ylabel =␣
 ↪'Frequency', title = "Distribution of y_i in Train data", grid = 'true',␣
 ↪color = colors)
plt.show()
for i in range(1,10):
    print("Number of data points in Class ", i , " : ",␣
 ↪train_class_distribution.values[i-1], "(", round(train_class_distribution.
 ↪values[i-1]/train_df.shape[0] * 100, 3), "%)")

print("\n", '*'*100, "\n")

test_class_distribution.plot(kind = 'bar', xlabel = 'Class', ylabel =␣
 ↪'Frequency', title = "Distribution of y_i in Test data", grid = 'true',␣
 ↪color = colors)
plt.show()
for i in range(1,10):
    print("Number of data points in Class ", i , " : ", test_class_distribution.
 ↪values[i-1], "(", round(test_class_distribution.values[i-1]/test_df.shape[0]␣
 ↪* 100, 3), "%)")

print("\n", '*'*100, "\n")

cross_val_class_distribution.plot(kind = 'bar', xlabel = 'Class', ylabel =␣
 ↪'Frequency', title = "Distribution of y_i in Cross Validation data", grid =␣
 ↪'true', color = colors)
```

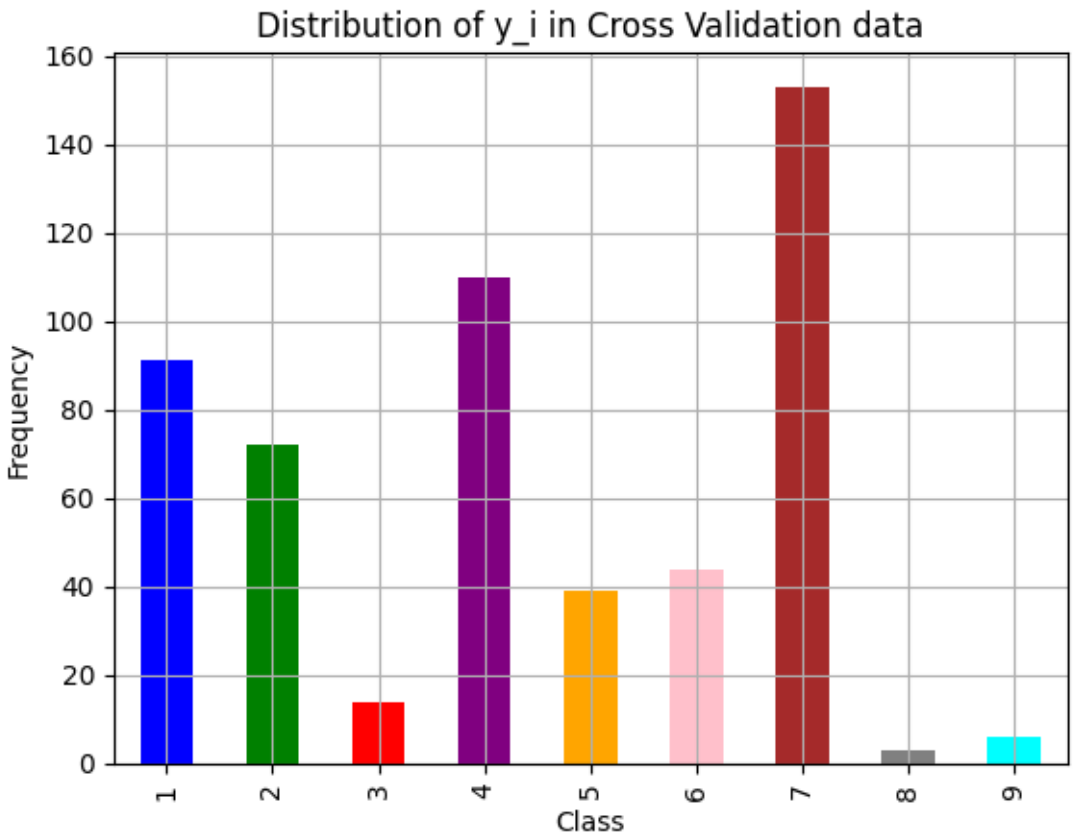
```
plt.show()
for i in range(1,10):
    print("Number of data points in Class ", i , " : ",␣
 ↪cross_val_class_distribution.values[i-1], "(",␣
 ↪round(cross_val_class_distribution.values[i-1]/test_df.shape[0] * 100, 3),␣
 ↪"%)")
```




```
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  3  :  57 ( 2.684 %)
Number of data points in Class  4  :  439 ( 20.669 %)
Number of data points in Class  5  :  155 ( 7.298 %)
Number of data points in Class  6  :  176 ( 8.286 %)
Number of data points in Class  7  :  609 ( 28.672 %)
Number of data points in Class  8  :  12 ( 0.565 %)
Number of data points in Class  9  :  24 ( 1.13 %)

 ************************************************************************************
**********************
```




```
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  3  :  18 ( 2.707 %)
Number of data points in Class  4  :  137 ( 20.602 %)
Number of data points in Class  5  :  48 ( 7.218 %)
Number of data points in Class  6  :  55 ( 8.271 %)
Number of data points in Class  7  :  191 ( 28.722 %)
Number of data points in Class  8  :  4 ( 0.602 %)
Number of data points in Class  9  :  7 ( 1.053 %)

 ************************************************************************************
**********************
```




```
Number of data points in Class  1  :  91 ( 13.684 %)
Number of data points in Class  2  :  72 ( 10.827 %)
Number of data points in Class  3  :  14 ( 2.105 %)
Number of data points in Class  4  :  110 ( 16.541 %)
Number of data points in Class  5  :  39 ( 5.865 %)
Number of data points in Class  6  :  44 ( 6.617 %)
Number of data points in Class  7  :  153 ( 23.008 %)
Number of data points in Class  8  :  3 ( 0.451 %)
Number of data points in Class  9  :  6 ( 0.902 %)
```

4 Random Model

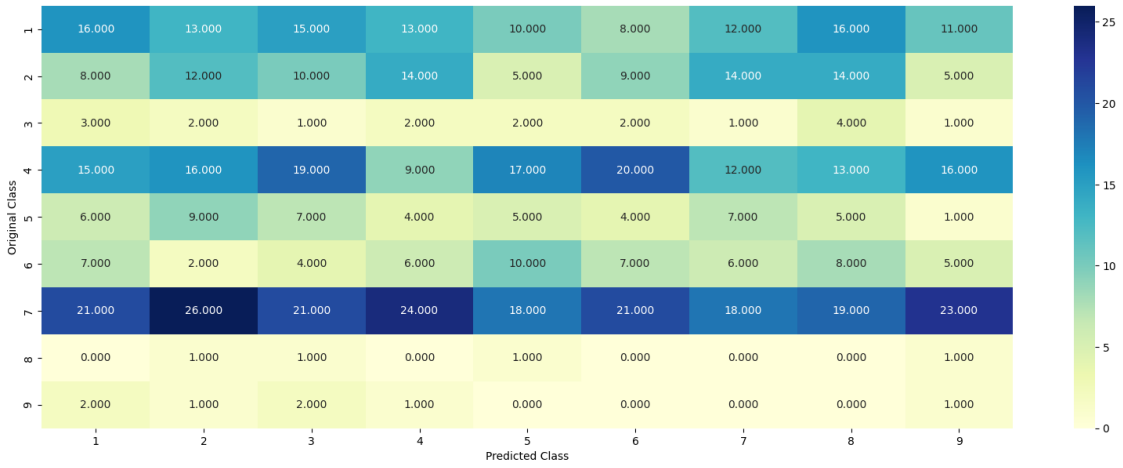
[ ]:
```
def plot_confusion_matrix(y_test, y_predict, colorMap = "YlGnBu"):
    C = confusion_matrix(y_test, y_predict)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))

    labels = [1,2,3,4,5,6,7,8,9]
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
```

```
    sns.heatmap(C, annot=True, cmap=colorMap, fmt=".3f", xticklabels=labels,␣
 ↪yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap=colorMap, fmt=".3f", xticklabels=labels,␣
 ↪yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

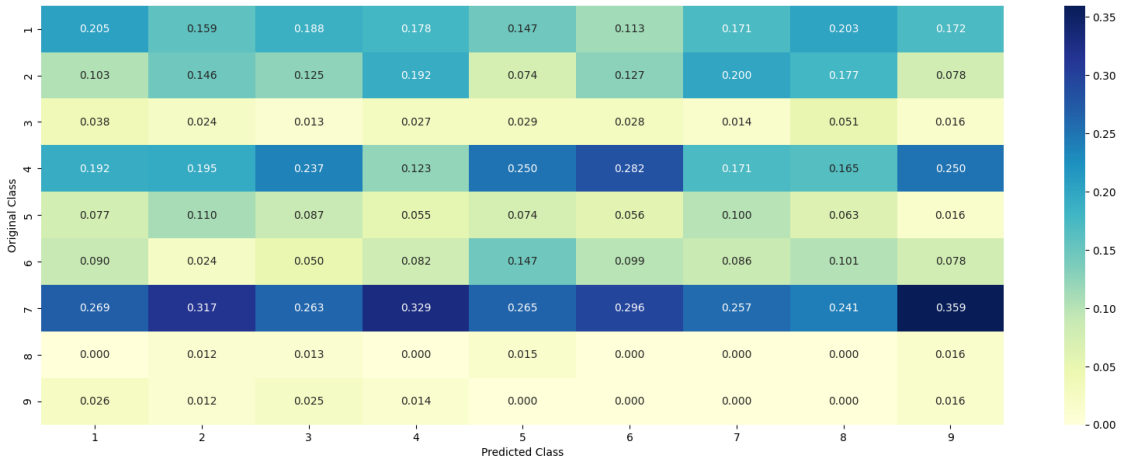
[ ]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their␣
 ↪sum
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]
cv_predicted_y = np.zeros((cv_data_len,9))
# print(test_data_len) - 665
# print(cv_data_len) - 532
# print(cv_predicted_y.shape) - (532, 9)


# print(random_probs)
# print(sum(sum(random_probs))) 2-D so double sum
# print(random_probs / sum(sum(random_probs)))

# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.
 ↪html
for i in range(cv_data_len):
    random_probs = np.random.rand(1,9) # 1 row, 9 columns
    cv_predicted_y[i] = random_probs / ((random_probs/
 ↪sum(sum(random_probs)))[0])
print("Log loss on Cross Validation Data using Random Model :␣
 ↪",log_loss(y_cv,cv_predicted_y))
```

```
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))

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

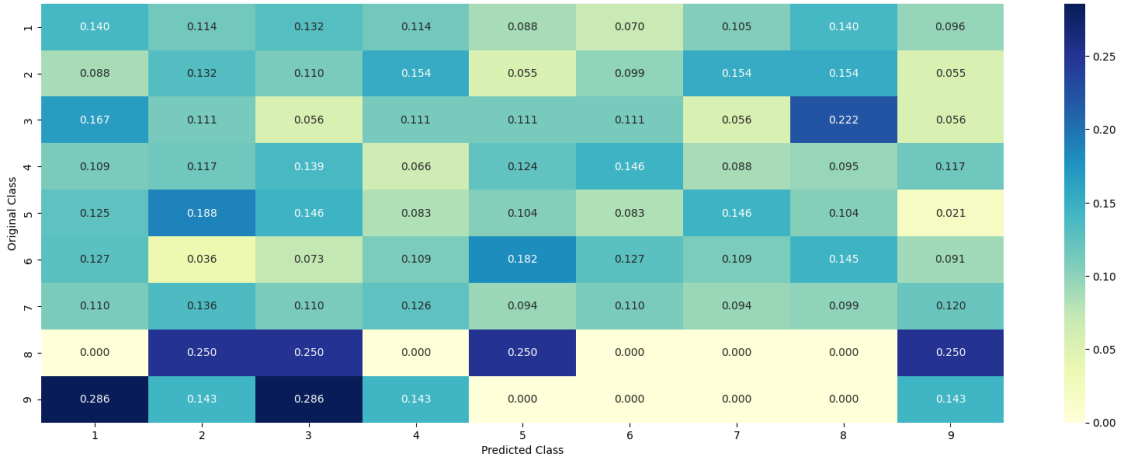
```
Log loss on Cross Validation Data using Random Model :  2.1972245773362196
Log loss on Test Data using Random Model 2.473478484617847
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



5 Univariate Analysis on GENE Feature

How many different types of the genes are there and how they are distributed?

[ ]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes or different categories of gene :', unique_genes.
  ↪shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))


# print(train_df)
# print(train_df['Gene'])
# print(train_df['Gene'].value_counts())
# print(unique_genes.shape) (232, )

# print(train_df['Gene'].value_counts())
```

```
Number of Unique Genes or different categories of gene : 229
BRCA1     182
EGFR      103
TP53       99
PTEN       83
BRCA2      82
BRAF       59
KIT        55
ERBB2      46
ALK        42
```

```
PIK3CA     39
Name: Gene, dtype: int64
```

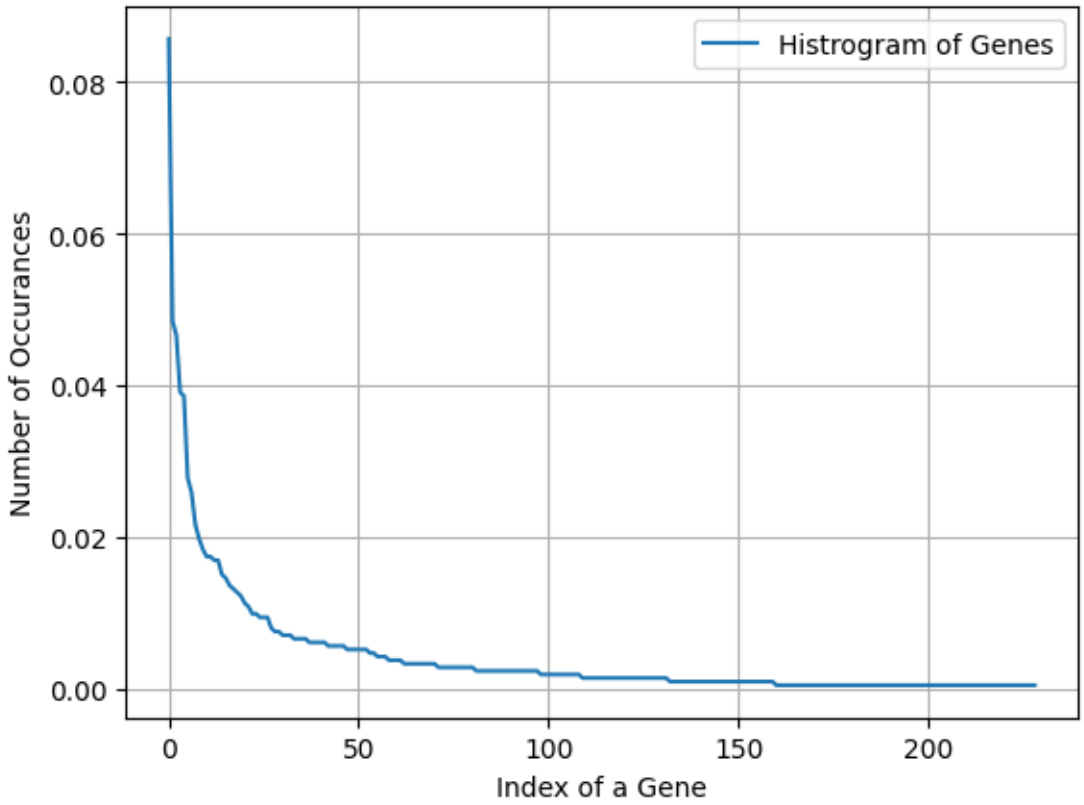
[ ]:
```
print(len(unique_genes.values))
unique_genes.values
```

```
229
```

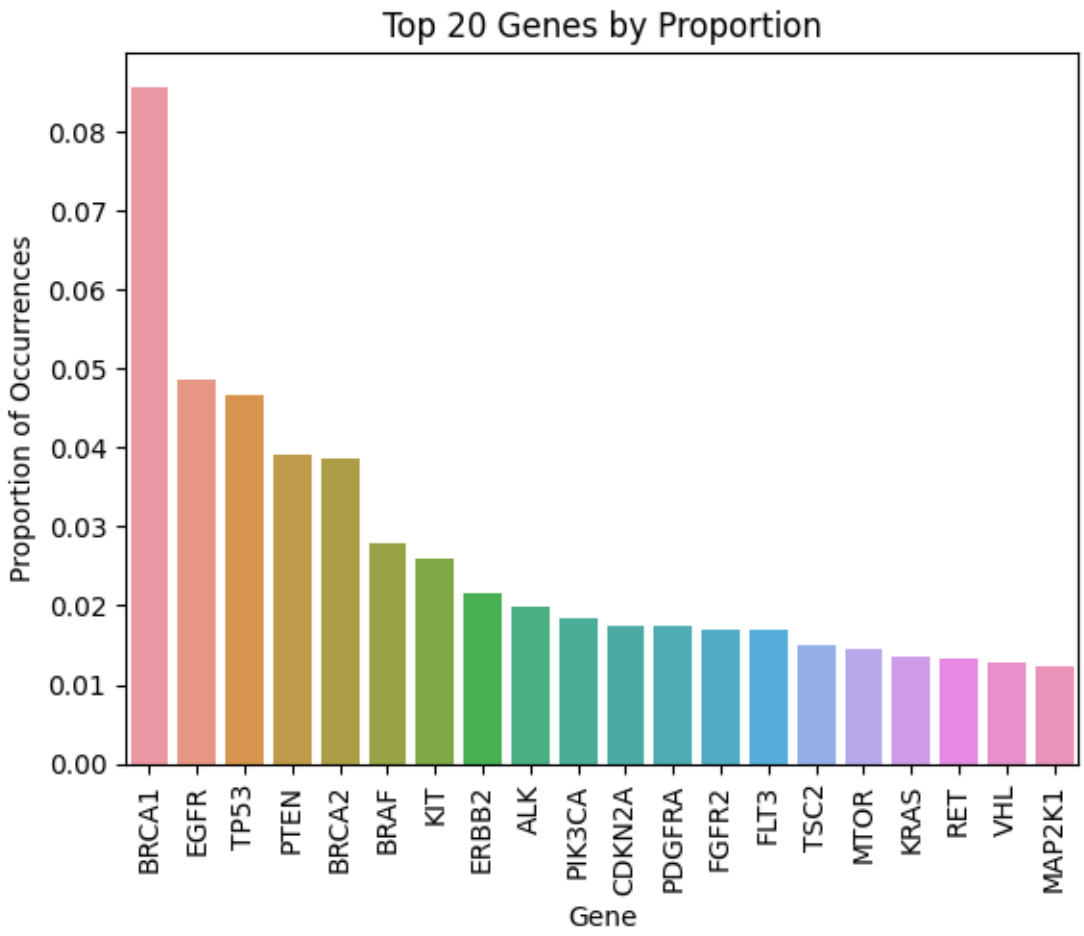
[ ]:
```
array([182, 103,  99,  83,  82,  59,  55,  46,  42,  39,  37,  37,  36,
        36,  32,  31,  29,  28,  27,  26,  24,  23,  21,  21,  20,  20,
        20,  17,  16,  16,  15,  15,  15,  14,  14,  14,  14,  13,  13,
        13,  13,  13,  12,  12,  12,  12,  12,  11,  11,  11,  11,  11,
        11,  10,  10,   9,   9,   9,   8,   8,   8,   8,   7,   7,   7,
         7,   7,   7,   7,   7,   7,   6,   6,   6,   6,   6,   6,   6,
         6,   6,   6,   5,   5,   5,   5,   5,   5,   5,   5,   5,   5,
         5,   5,   5,   5,   5,   5,   5,   4,   4,   4,   4,   4,   4,
         4,   4,   4,   4,   4,   3,   3,   3,   3,   3,   3,   3,   3,
         3,   3,   3,   3,   3,   3,   3,   3,   3,   3,   3,   3,   3,
         3,   3,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,
         2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,   2,
         2,   2,   2,   2,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1])
```

[ ]:
```
s = sum(unique_genes.values);
print('SUM : ', s)
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```

```
SUM :  2124
```

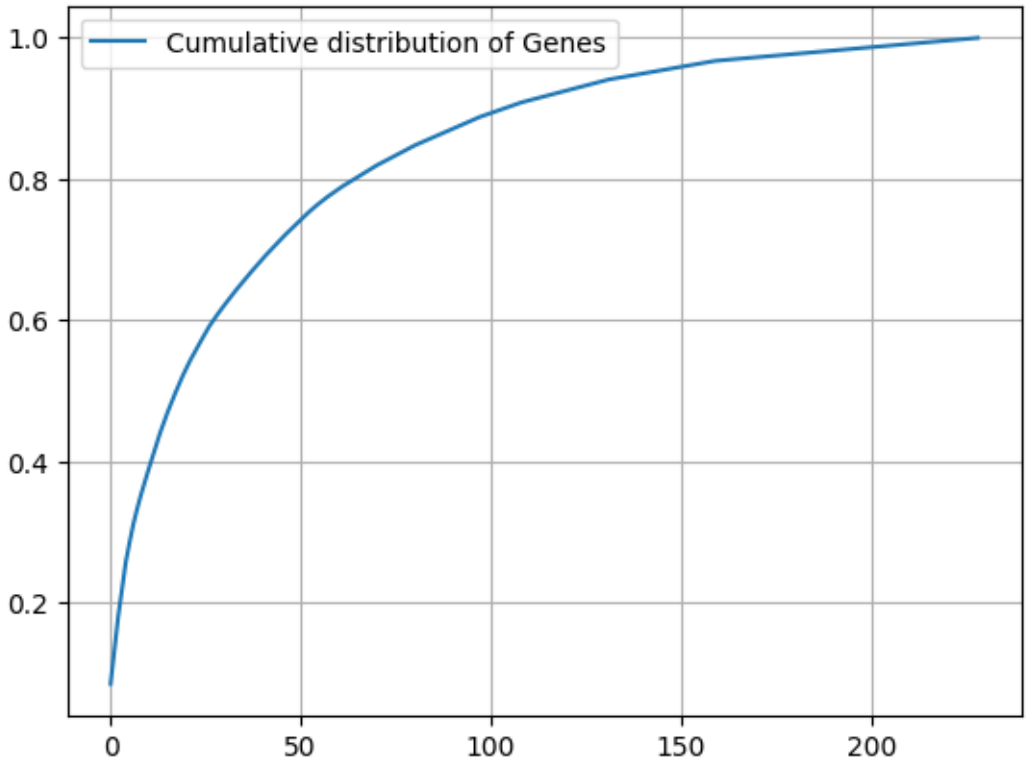


[ ]:
```
proportions = unique_genes / unique_genes.sum()

sns.barplot(x=proportions.index[:20], y=proportions.values[:20])
plt.xlabel('Gene')
plt.ylabel('Proportion of Occurrences')
plt.title('Top 20 Genes by Proportion')
plt.xticks(rotation=90)
plt.show()
```



[ ]:

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



6 1. ONE HOT ENCODING

7 2. LAPLACE SMOOTHING

8 3. RESPONSE CODING

[ ]:
```
# build a vector (1*9) , the first element = (number of times it occured in␣
 ↪class1 + 10*alpha / number of time it occurred in total data+90*alpha)
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] if the word not in given␣
 ↪text

def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()

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

    for i, denominator in value_count.items():
        # vec is 9d 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.shape[0] will contains number of pts s.t class label == 1␣
↪and gene g'i == 'BRCA1'


            # (train_df['Class']==1) & (train_df['Gene']=='BRCA1') we can use␣
↪this as masking
            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 occured in whole data
            # the denominator will contains the number of time gene g'i occurs.
            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

# print(gv_dict)
    #     {'BRCA1': [0.200757575757575, 0.037878787878788, 0.
↪068181818181818177, 0.13636363636363635, 0.25, 0.1931818181818181, 0.
↪037878787878788, 0.037878787878788, 0.037878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.
↪061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.
↪066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.
↪056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.
↪068181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.
↪0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.
↪060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.
↪34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.
↪060606060606060608],
```

```
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.
 ↪069182389937106917, 0.46540880503144655, 0.07547169811320754, 0.
 ↪062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.
 ↪062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.
 ↪072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.
 ↪066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.
 ↪066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.
 ↪073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.
 ↪080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.
 ↪066666666666666666],
    #      ...
    #     }

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):

    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 data
    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,-1])
    return gv_fea


# print(dict(value_count).keys())
# dict_keys(['BRCA1', 'TP53', 'EGFR', 'PTEN', 'BRCA2', 'KIT', 'BRAF', 'ERBB2',␣
 ↪'FLT3', 'PIK3CA', 'PDGFRA',
# 'ALK', 'CDKN2A', 'FGFR2', 'MAP2K1', 'TSC2', 'MTOR', .....])
```

[ ]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
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))
print("train_gene_feature_responseCoding is converted feature using respone␣
 ↪coding method. The shape of gene feature:",␣
 ↪train_gene_feature_responseCoding.shape)
```

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

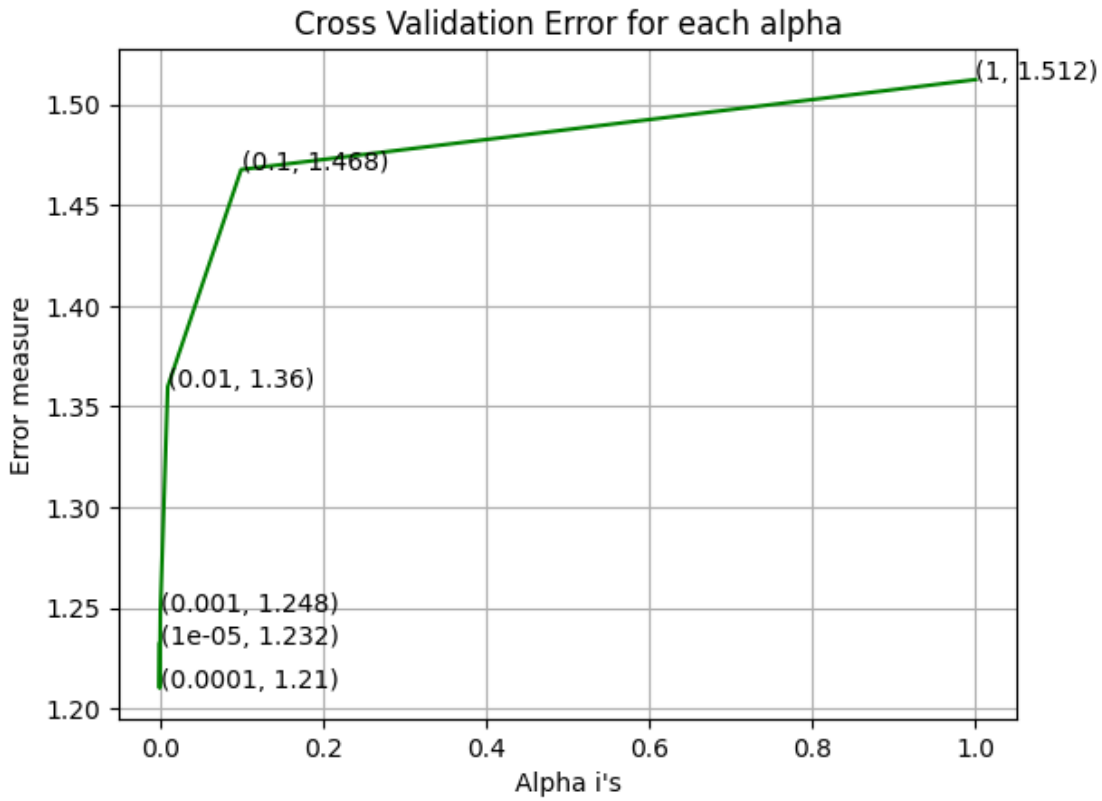
[ ]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.
 ↪fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])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

[ ]:
```
(2124, 229)
```

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


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: 1.2322356328549267
For values of alpha =  0.0001 The log loss is: 1.210324953588144
For values of alpha =  0.001 The log loss is: 1.2478861657003089
For values of alpha =  0.01 The log loss is: 1.359824984161938
For values of alpha =  0.1 The log loss is: 1.4675888048203702
For values of alpha =  1 The log loss is: 1.5122190743791426
```




```
For values of best alpha =  0.0001 The train log loss is: 0.978504294460452
For values of best alpha =  0.0001 The cross validation log loss is:
1.210324953588144
For values of best alpha =  0.0001 The test log loss is: 1.1990286385118565
```

[ ]:
```
print("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)
```

```
How many data points in Test and CV datasets are covered by the  229  genes in
train dataset?
Ans
1. In test data 641 out of 665 : 96.39097744360903
2. In cross validation data 509 out of  532 : 95.67669172932331
```

[ ]:

Univariate Analysis on Variation Feature

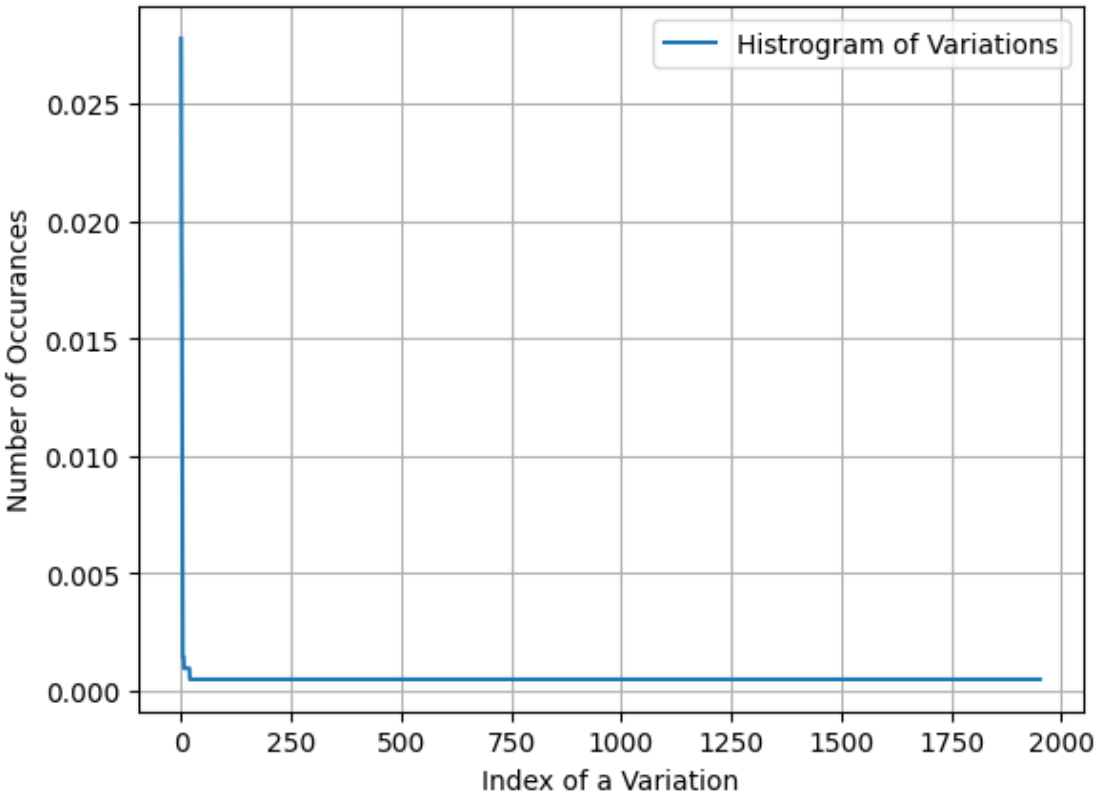
[ ]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1953
Truncating_Mutations    59
Amplification           43
Deletion                35
Fusions                 19
T58I                     3
Overexpression           3
G12V                     3
G13C                     2
G67R                     2
S308A                    2
Name: Variation, dtype: int64
```

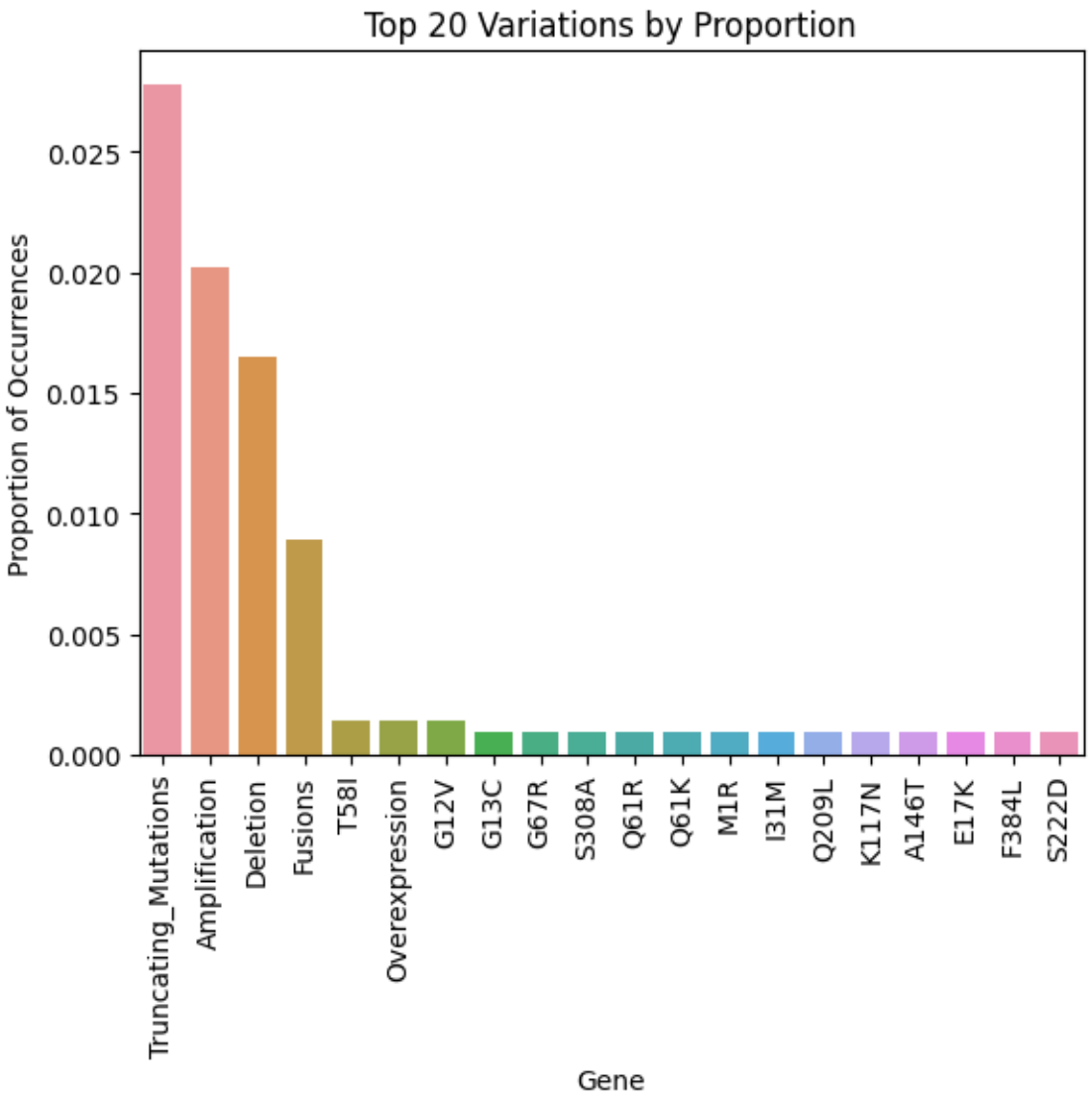
[ ]:
```
print("There are", unique_variations.shape[0] ,"different categories of␣
 ↪variations in the train data, and they are distibuted as follows",)
```

Ans: There are 1953 different categories of variations in the train data, and they are distibuted as follows

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

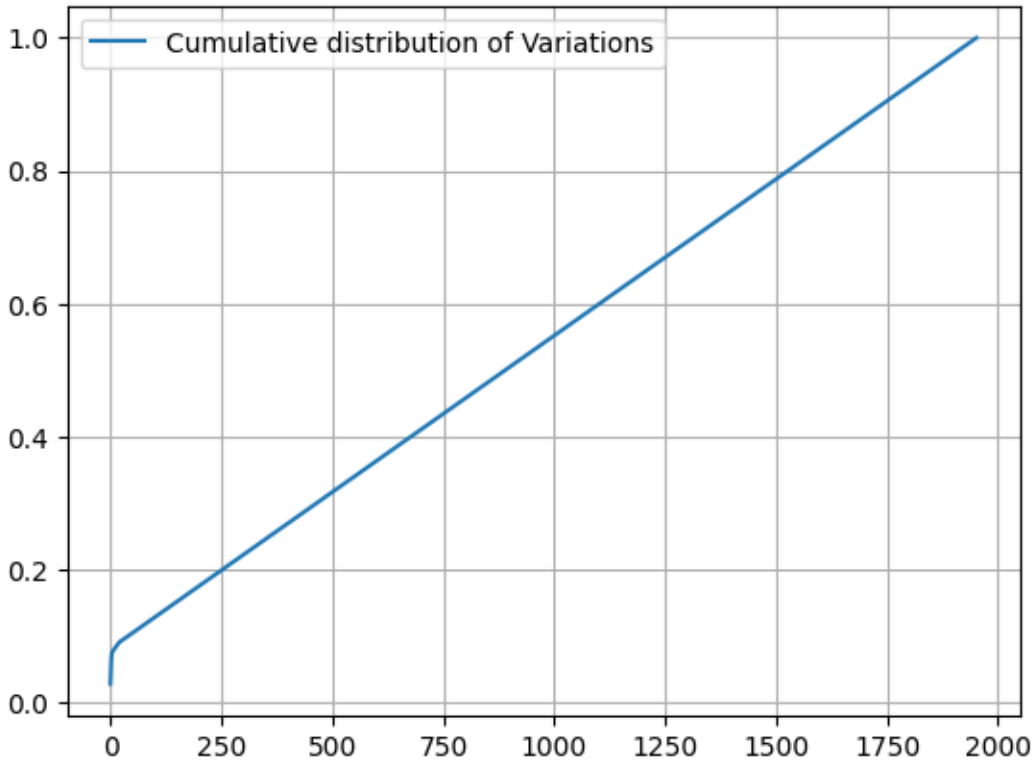


[ ]:
```
proportions = unique_variations / unique_variations.sum()

sns.barplot(x=proportions.index[:20], y=proportions.values[:20])
plt.xlabel('Gene')
plt.ylabel('Proportion of Occurrences')
plt.title('Top 20 Variations by Proportion')
plt.xticks(rotation=90)
plt.show()
```

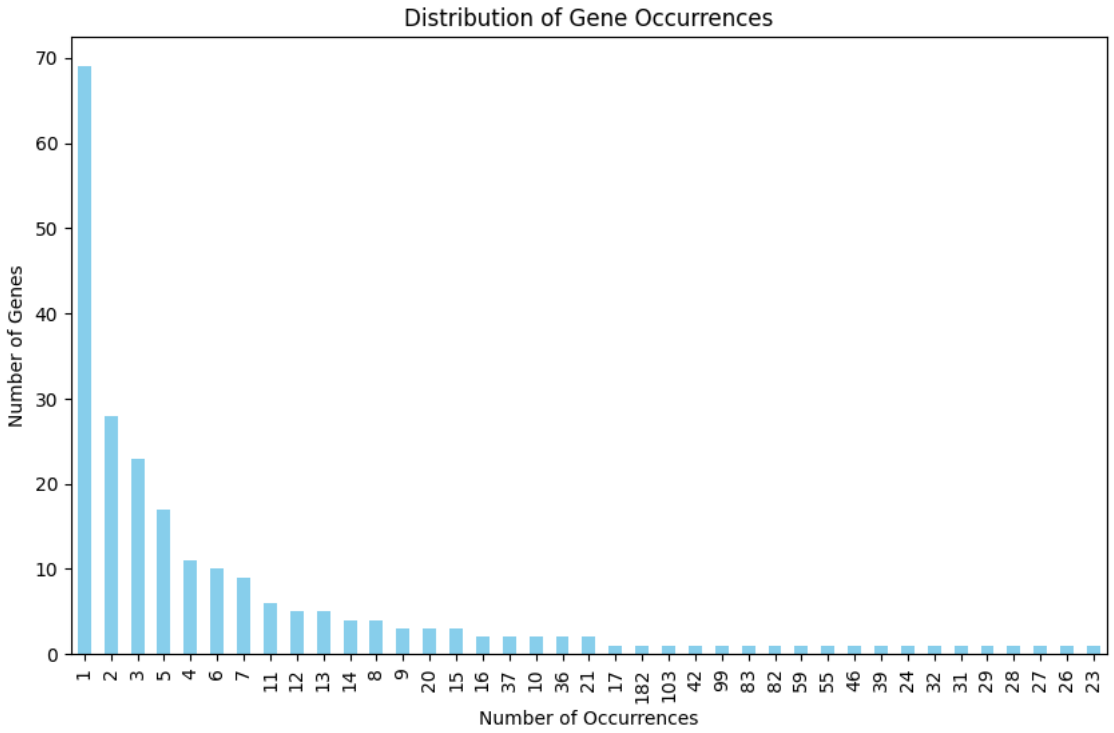


[ ]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02777778 0.0480226  0.06450094 … 0.99905838 0.99952919 1.        ]
```



[ ]:
```
# Plotting gene distribution
plt.figure(figsize=(10, 6))
unique_genes.value_counts().plot(kind='bar', color='skyblue')
plt.xlabel('Number of Occurrences')
plt.ylabel('Number of Genes')
plt.title('Distribution of Gene Occurrences')
plt.show()
```



How to featurize this Variation feature ?

There are two ways we can featurize this variable:

One hot Encoding

Response coding

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

[ ]:
```
# 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))
```

[ ]:
```
print("train_variation_feature_responseCoding is a converted feature using the␣
 ↪response coding method. The shape of Variation feature:",␣
 ↪train_variation_feature_responseCoding.shape)
```

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

[ ]:
```
# 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'])
```

[ ]:
```

```

[ ]:
```
print("train_variation_feature_onehotEncoded is converted feature using the␣
 ↪onne-hot encoding method. The shape of Variation feature:",␣
 ↪train_variation_feature_onehotCoding.shape)
```

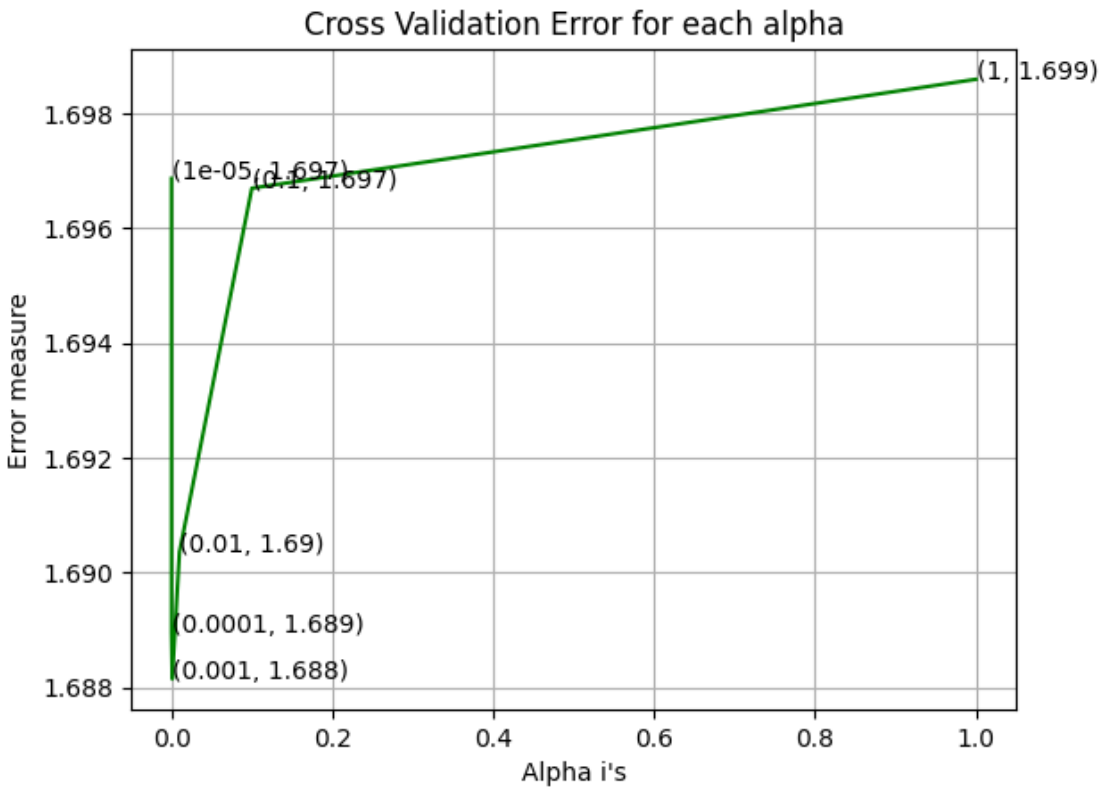
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1981)

How good is this Variation feature in predicting y_i?

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

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")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, 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_loss(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, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.69686986047081
For values of alpha =  0.0001 The log loss is: 1.688960659213454
For values of alpha =  0.001 The log loss is: 1.6881469273312195
For values of alpha =  0.01 The log loss is: 1.6903588114359285
For values of alpha =  0.1 The log loss is: 1.6967007322208325
For values of alpha =  1 The log loss is: 1.698598852388498
```




```
For values of best alpha =  0.001 The train log loss is: 1.0415750243166686
For values of best alpha =  0.001 The cross validation log loss is:
1.6881469273312195
For values of best alpha =  0.001 The test log loss is: 1.6887139410583518
```

[ ]:
```
print("How many data points are covered by total ", unique_variations.shape[0],␣
 ↪" genes in test 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.shape[0])*100)
```

```
How many data points are covered by total  1953  genes in test and cross
validation data sets?
Ans
1. In test data 84 out of 665 : 12.631578947368421
2. In cross validation data 62 out of  532 : 11.654135338345863
```

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 predicitng 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
```

[ ]:
```
import math
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
```

[ ]:
```
# building a CountVectorizer with all the words that occured 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_out()

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

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


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

```
Total number of unique words in train data : 53257
```

[ ]:

[ ]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries 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 buid 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)
```

[ ]:
```
#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)
```

[ ]:
```
# 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
```

[ ]:
```
# 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)
```

[ ]:
```
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()))
```

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

```
Counter({3: 5490, 4: 3727, 6: 2872, 5: 2538, 7: 2450, 9: 2096, 8: 1758, 10:
1449, 12: 1392, 13: 1077, 11: 942, 14: 922, 18: 815, 15: 721, 16: 689, 20: 570,
24: 555, 21: 524, 17: 521, 19: 515, 25: 445, 22: 438, 23: 428, 30: 411, 26: 388,
38: 373, 27: 359, 28: 337, 36: 292, 32: 276, 52: 272, 33: 261, 31: 259, 29: 243,
35: 227, 40: 221, 34: 220, 48: 210, 42: 208, 39: 207, 43: 197, 44: 188, 54: 182,
37: 181, 47: 178, 46: 173, 45: 170, 41: 165, 49: 164, 60: 155, 50: 143, 56: 142,
51: 131, 57: 129, 53: 124, 55: 121, 61: 118, 62: 115, 64: 112, 58: 108, 59: 107,
77: 105, 69: 104, 70: 103, 65: 103, 63: 102, 72: 94, 67: 92, 71: 89, 75: 88, 94:
87, 78: 86, 84: 83, 66: 83, 76: 82, 68: 82, 90: 78, 81: 78, 73: 77, 82: 76, 79:
75, 104: 73, 91: 73, 95: 72, 74: 70, 80: 69, 87: 64, 100: 63, 83: 62, 89: 59,
86: 58, 85: 57, 111: 56, 120: 55, 96: 55, 93: 55, 88: 54, 110: 53, 98: 53, 97:
52, 115: 51, 103: 51, 99: 51, 119: 50, 102: 49, 113: 48, 105: 48, 112: 47, 109:
47, 143: 45, 117: 45, 132: 44, 108: 44, 101: 44, 121: 43, 107: 43, 141: 42, 135:
42, 92: 42, 144: 41, 131: 41, 128: 41, 124: 41, 123: 41, 153: 39, 150: 39, 139:
39, 116: 39, 156: 38, 134: 38, 160: 37, 127: 37, 114: 37, 145: 36, 136: 36, 125:
36, 106: 36, 142: 35, 118: 35, 147: 34, 140: 34, 126: 34, 154: 33, 152: 33, 176:
32, 159: 32, 163: 31, 158: 31, 137: 31, 138: 30, 149: 29, 129: 29, 194: 28, 193:
28, 157: 27, 151: 27, 133: 27, 130: 27, 210: 26, 203: 26, 165: 26, 148: 26, 122:
26, 192: 25, 185: 25, 177: 25, 161: 25, 155: 25, 184: 24, 183: 24, 178: 24, 167:
24, 223: 23, 212: 23, 196: 23, 186: 23, 179: 23, 168: 23, 217: 22, 187: 22, 181:
22, 173: 22, 172: 22, 218: 21, 207: 21, 190: 21, 189: 21, 164: 21, 236: 20, 226:
20, 202: 20, 174: 20, 171: 20, 146: 20, 285: 19, 250: 19, 242: 19, 206: 19, 205:
19, 200: 19, 191: 19, 180: 19, 169: 19, 343: 18, 294: 18, 291: 18, 283: 18, 264:
18, 260: 18, 259: 18, 255: 18, 251: 18, 245: 18, 227: 18, 225: 18, 222: 18, 219:
18, 199: 18, 198: 18, 188: 18, 162: 18, 292: 17, 249: 17, 247: 17, 234: 17, 224:
```

17, 195: 17, 175: 17, 273: 16, 239: 16, 231: 16, 216: 16, 213: 16, 208: 16, 204:
16, 170: 16, 298: 15, 281: 15, 278: 15, 276: 15, 262: 15, 256: 15, 241: 15, 209:
15, 201: 15, 182: 15, 166: 15, 435: 14, 371: 14, 362: 14, 358: 14, 287: 14, 286:
14, 282: 14, 270: 14, 266: 14, 265: 14, 237: 14, 235: 14, 232: 14, 230: 14, 229:
14, 444: 13, 438: 13, 374: 13, 315: 13, 312: 13, 301: 13, 275: 13, 272: 13, 257:
13, 248: 13, 244: 13, 214: 13, 396: 12, 373: 12, 357: 12, 331: 12, 311: 12, 297:
12, 277: 12, 268: 12, 253: 12, 240: 12, 238: 12, 215: 12, 211: 12, 464: 11, 405:
11, 382: 11, 368: 11, 356: 11, 337: 11, 321: 11, 319: 11, 316: 11, 314: 11, 308:
11, 303: 11, 296: 11, 293: 11, 271: 11, 263: 11, 261: 11, 254: 11, 243: 11, 228:
11, 221: 11, 485: 10, 453: 10, 412: 10, 366: 10, 351: 10, 347: 10, 342: 10, 339:
10, 336: 10, 333: 10, 332: 10, 327: 10, 317: 10, 310: 10, 290: 10, 288: 10, 269:
10, 267: 10, 246: 10, 233: 10, 220: 10, 197: 10, 769: 9, 511: 9, 484: 9, 461: 9,
424: 9, 414: 9, 399: 9, 355: 9, 354: 9, 344: 9, 341: 9, 329: 9, 325: 9, 313: 9,
307: 9, 302: 9, 300: 9, 295: 9, 289: 9, 279: 9, 258: 9, 625: 8, 576: 8, 566: 8,
564: 8, 553: 8, 542: 8, 507: 8, 502: 8, 449: 8, 446: 8, 445: 8, 440: 8, 425: 8,
422: 8, 418: 8, 416: 8, 413: 8, 398: 8, 391: 8, 390: 8, 384: 8, 372: 8, 340: 8,
338: 8, 335: 8, 334: 8, 322: 8, 309: 8, 306: 8, 304: 8, 299: 8, 284: 8, 280: 8,
274: 8, 252: 8, 758: 7, 623: 7, 611: 7, 603: 7, 589: 7, 557: 7, 552: 7, 541: 7,
520: 7, 513: 7, 509: 7, 481: 7, 477: 7, 459: 7, 456: 7, 439: 7, 411: 7, 406: 7,
401: 7, 393: 7, 389: 7, 380: 7, 379: 7, 369: 7, 367: 7, 360: 7, 353: 7, 352: 7,
350: 7, 349: 7, 345: 7, 330: 7, 326: 7, 320: 7, 987: 6, 969: 6, 779: 6, 773: 6,
733: 6, 714: 6, 691: 6, 688: 6, 656: 6, 654: 6, 650: 6, 629: 6, 600: 6, 587: 6,
585: 6, 568: 6, 565: 6, 561: 6, 544: 6, 537: 6, 533: 6, 530: 6, 522: 6, 501: 6,
497: 6, 492: 6, 491: 6, 490: 6, 480: 6, 473: 6, 472: 6, 469: 6, 460: 6, 441: 6,
437: 6, 431: 6, 430: 6, 428: 6, 402: 6, 387: 6, 385: 6, 381: 6, 376: 6, 364: 6,
348: 6, 346: 6, 328: 6, 323: 6, 1877: 5, 1376: 5, 1319: 5, 1262: 5, 1128: 5,
1052: 5, 958: 5, 907: 5, 892: 5, 846: 5, 815: 5, 799: 5, 797: 5, 795: 5, 788: 5,
778: 5, 766: 5, 763: 5, 756: 5, 739: 5, 728: 5, 717: 5, 716: 5, 707: 5, 669: 5,
661: 5, 649: 5, 618: 5, 605: 5, 594: 5, 586: 5, 574: 5, 572: 5, 554: 5, 548: 5,
547: 5, 526: 5, 517: 5, 515: 5, 514: 5, 500: 5, 499: 5, 493: 5, 483: 5, 452: 5,
451: 5, 448: 5, 433: 5, 423: 5, 410: 5, 408: 5, 404: 5, 403: 5, 395: 5, 394: 5,
392: 5, 388: 5, 386: 5, 378: 5, 377: 5, 370: 5, 365: 5, 324: 5, 318: 5, 1398: 4,
1331: 4, 1318: 4, 1311: 4, 1277: 4, 1274: 4, 1257: 4, 1223: 4, 1211: 4, 1206: 4,
1201: 4, 1170: 4, 1150: 4, 1102: 4, 1100: 4, 1087: 4, 1079: 4, 1017: 4, 1014: 4,
1013: 4, 993: 4, 991: 4, 982: 4, 932: 4, 922: 4, 918: 4, 917: 4, 886: 4, 881: 4,
877: 4, 875: 4, 871: 4, 863: 4, 859: 4, 847: 4, 841: 4, 833: 4, 830: 4, 829: 4,
805: 4, 802: 4, 801: 4, 789: 4, 777: 4, 775: 4, 774: 4, 755: 4, 751: 4, 740: 4,
726: 4, 708: 4, 701: 4, 700: 4, 673: 4, 670: 4, 667: 4, 663: 4, 653: 4, 648: 4,
647: 4, 645: 4, 639: 4, 638: 4, 631: 4, 627: 4, 624: 4, 617: 4, 616: 4, 610: 4,
607: 4, 606: 4, 601: 4, 599: 4, 588: 4, 583: 4, 580: 4, 575: 4, 558: 4, 546: 4,
543: 4, 539: 4, 536: 4, 535: 4, 532: 4, 527: 4, 512: 4, 508: 4, 498: 4, 496: 4,
482: 4, 479: 4, 476: 4, 474: 4, 470: 4, 468: 4, 465: 4, 462: 4, 457: 4, 455: 4,
447: 4, 442: 4, 432: 4, 427: 4, 426: 4, 420: 4, 419: 4, 400: 4, 397: 4, 383: 4,
363: 4, 361: 4, 359: 4, 305: 4, 4828: 3, 4665: 3, 4334: 3, 3069: 3, 2967: 3,
2634: 3, 2495: 3, 2471: 3, 2405: 3, 2378: 3, 2161: 3, 2126: 3, 2047: 3, 2003: 3,
1928: 3, 1912: 3, 1846: 3, 1840: 3, 1811: 3, 1809: 3, 1795: 3, 1776: 3, 1733: 3,
1692: 3, 1685: 3, 1684: 3, 1643: 3, 1615: 3, 1600: 3, 1599: 3, 1574: 3, 1556: 3,
1554: 3, 1553: 3, 1536: 3, 1489: 3, 1476: 3, 1467: 3, 1443: 3, 1425: 3, 1402: 3,

1397: 3, 1393: 3, 1371: 3, 1362: 3, 1348: 3, 1329: 3, 1307: 3, 1297: 3, 1293: 3,
1285: 3, 1281: 3, 1275: 3, 1266: 3, 1261: 3, 1260: 3, 1253: 3, 1230: 3, 1221: 3,
1213: 3, 1205: 3, 1184: 3, 1176: 3, 1168: 3, 1165: 3, 1163: 3, 1160: 3, 1158: 3,
1143: 3, 1127: 3, 1125: 3, 1122: 3, 1099: 3, 1086: 3, 1078: 3, 1068: 3, 1048: 3,
1043: 3, 1037: 3, 1026: 3, 1002: 3, 996: 3, 984: 3, 972: 3, 967: 3, 966: 3, 952:
3, 940: 3, 930: 3, 929: 3, 926: 3, 923: 3, 901: 3, 891: 3, 890: 3, 889: 3, 882:
3, 880: 3, 872: 3, 853: 3, 840: 3, 837: 3, 827: 3, 825: 3, 807: 3, 800: 3, 798:
3, 794: 3, 793: 3, 792: 3, 787: 3, 785: 3, 783: 3, 771: 3, 767: 3, 765: 3, 760:
3, 745: 3, 741: 3, 738: 3, 737: 3, 734: 3, 731: 3, 730: 3, 720: 3, 719: 3, 711:
3, 710: 3, 706: 3, 695: 3, 694: 3, 693: 3, 687: 3, 683: 3, 682: 3, 680: 3, 679:
3, 676: 3, 675: 3, 672: 3, 666: 3, 665: 3, 659: 3, 658: 3, 651: 3, 646: 3, 644:
3, 642: 3, 641: 3, 640: 3, 635: 3, 633: 3, 622: 3, 621: 3, 613: 3, 602: 3, 597:
3, 596: 3, 595: 3, 593: 3, 592: 3, 584: 3, 570: 3, 567: 3, 562: 3, 560: 3, 559:
3, 550: 3, 540: 3, 534: 3, 529: 3, 528: 3, 525: 3, 523: 3, 519: 3, 510: 3, 486:
3, 478: 3, 471: 3, 466: 3, 463: 3, 458: 3, 450: 3, 429: 3, 421: 3, 417: 3, 415:
3, 409: 3, 15414: 2, 12277: 2, 11568: 2, 9080: 2, 7802: 2, 7448: 2, 7272: 2,
7169: 2, 5883: 2, 5686: 2, 5465: 2, 5159: 2, 4553: 2, 4345: 2, 4319: 2, 4284: 2,
4267: 2, 4209: 2, 4149: 2, 4054: 2, 3991: 2, 3854: 2, 3774: 2, 3742: 2, 3621: 2,
3601: 2, 3597: 2, 3510: 2, 3508: 2, 3489: 2, 3432: 2, 3412: 2, 3400: 2, 3335: 2,
3297: 2, 3278: 2, 3275: 2, 3257: 2, 3201: 2, 3193: 2, 3157: 2, 3132: 2, 3125: 2,
3111: 2, 3099: 2, 3062: 2, 3047: 2, 3027: 2, 3010: 2, 3009: 2, 2930: 2, 2871: 2,
2808: 2, 2803: 2, 2710: 2, 2615: 2, 2613: 2, 2611: 2, 2607: 2, 2586: 2, 2584: 2,
2569: 2, 2567: 2, 2557: 2, 2500: 2, 2488: 2, 2477: 2, 2444: 2, 2434: 2, 2418: 2,
2410: 2, 2373: 2, 2369: 2, 2356: 2, 2311: 2, 2302: 2, 2271: 2, 2266: 2, 2222: 2,
2199: 2, 2189: 2, 2186: 2, 2155: 2, 2154: 2, 2145: 2, 2134: 2, 2123: 2, 2099: 2,
2084: 2, 2082: 2, 2074: 2, 2068: 2, 2065: 2, 2064: 2, 2061: 2, 2060: 2, 2058: 2,
2057: 2, 2051: 2, 2044: 2, 2019: 2, 1995: 2, 1976: 2, 1964: 2, 1958: 2, 1954: 2,
1952: 2, 1938: 2, 1937: 2, 1923: 2, 1906: 2, 1889: 2, 1885: 2, 1881: 2, 1880: 2,
1872: 2, 1862: 2, 1859: 2, 1855: 2, 1853: 2, 1833: 2, 1810: 2, 1800: 2, 1799: 2,
1797: 2, 1796: 2, 1756: 2, 1739: 2, 1735: 2, 1731: 2, 1728: 2, 1725: 2, 1718: 2,
1716: 2, 1707: 2, 1704: 2, 1686: 2, 1680: 2, 1678: 2, 1667: 2, 1655: 2, 1645: 2,
1644: 2, 1641: 2, 1636: 2, 1631: 2, 1630: 2, 1627: 2, 1623: 2, 1616: 2, 1614: 2,
1604: 2, 1603: 2, 1602: 2, 1586: 2, 1582: 2, 1561: 2, 1557: 2, 1550: 2, 1547: 2,
1530: 2, 1529: 2, 1525: 2, 1496: 2, 1495: 2, 1494: 2, 1488: 2, 1481: 2, 1480: 2,
1477: 2, 1459: 2, 1458: 2, 1450: 2, 1447: 2, 1446: 2, 1445: 2, 1444: 2, 1435: 2,
1430: 2, 1422: 2, 1421: 2, 1416: 2, 1414: 2, 1408: 2, 1400: 2, 1396: 2, 1394: 2,
1392: 2, 1390: 2, 1382: 2, 1379: 2, 1374: 2, 1370: 2, 1369: 2, 1368: 2, 1357: 2,
1356: 2, 1349: 2, 1346: 2, 1327: 2, 1324: 2, 1316: 2, 1313: 2, 1294: 2, 1291: 2,
1290: 2, 1287: 2, 1284: 2, 1271: 2, 1270: 2, 1254: 2, 1251: 2, 1243: 2, 1237: 2,
1235: 2, 1226: 2, 1224: 2, 1217: 2, 1202: 2, 1196: 2, 1191: 2, 1190: 2, 1172: 2,
1166: 2, 1162: 2, 1161: 2, 1157: 2, 1149: 2, 1146: 2, 1144: 2, 1142: 2, 1138: 2,
1137: 2, 1133: 2, 1126: 2, 1119: 2, 1116: 2, 1113: 2, 1098: 2, 1096: 2, 1093: 2,
1091: 2, 1089: 2, 1085: 2, 1084: 2, 1077: 2, 1074: 2, 1070: 2, 1064: 2, 1063: 2,
1055: 2, 1054: 2, 1050: 2, 1035: 2, 1032: 2, 1030: 2, 1029: 2, 1025: 2, 1024: 2,
1023: 2, 1021: 2, 1020: 2, 1016: 2, 1010: 2, 1009: 2, 1004: 2, 1001: 2, 999: 2,
992: 2, 990: 2, 981: 2, 980: 2, 978: 2, 977: 2, 975: 2, 973: 2, 971: 2, 968: 2,
965: 2, 963: 2, 962: 2, 954: 2, 953: 2, 950: 2, 947: 2, 945: 2, 943: 2, 936: 2,
928: 2, 927: 2, 919: 2, 913: 2, 911: 2, 910: 2, 909: 2, 906: 2, 905: 2, 904: 2,

902: 2, 900: 2, 897: 2, 895: 2, 894: 2, 893: 2, 888: 2, 887: 2, 879: 2, 878: 2,
870: 2, 869: 2, 867: 2, 856: 2, 851: 2, 850: 2, 836: 2, 826: 2, 824: 2, 823: 2,
822: 2, 821: 2, 820: 2, 818: 2, 817: 2, 816: 2, 813: 2, 811: 2, 810: 2, 806: 2,
804: 2, 786: 2, 781: 2, 780: 2, 772: 2, 770: 2, 768: 2, 764: 2, 753: 2, 752: 2,
749: 2, 748: 2, 747: 2, 746: 2, 744: 2, 735: 2, 727: 2, 723: 2, 722: 2, 721: 2,
718: 2, 715: 2, 712: 2, 709: 2, 704: 2, 699: 2, 698: 2, 696: 2, 690: 2, 686: 2,
685: 2, 684: 2, 681: 2, 677: 2, 674: 2, 668: 2, 664: 2, 662: 2, 655: 2, 652: 2,
637: 2, 636: 2, 632: 2, 626: 2, 620: 2, 619: 2, 615: 2, 614: 2, 612: 2, 604: 2,
591: 2, 590: 2, 577: 2, 573: 2, 571: 2, 569: 2, 555: 2, 545: 2, 538: 2, 531: 2,
524: 2, 516: 2, 505: 2, 504: 2, 503: 2, 489: 2, 488: 2, 487: 2, 475: 2, 467: 2,
454: 2, 436: 2, 434: 2, 407: 2, 375: 2, 152853: 1, 119552: 1, 79955: 1, 67890:
1, 67711: 1, 67709: 1, 66060: 1, 63981: 1, 63581: 1, 57008: 1, 53991: 1, 49312:
1, 48722: 1, 45882: 1, 45870: 1, 44088: 1, 43762: 1, 42187: 1, 41915: 1, 41904:
1, 41285: 1, 40576: 1, 39795: 1, 38590: 1, 38383: 1, 37612: 1, 37551: 1, 36172:
1, 35696: 1, 35317: 1, 34265: 1, 33616: 1, 33179: 1, 33042: 1, 32259: 1, 31362:
1, 29439: 1, 27941: 1, 26005: 1, 25929: 1, 25653: 1, 25543: 1, 24834: 1, 24789:
1, 24682: 1, 24594: 1, 24426: 1, 24310: 1, 24182: 1, 24112: 1, 24083: 1, 23950:
1, 23140: 1, 22919: 1, 22743: 1, 22381: 1, 22050: 1, 22029: 1, 20955: 1, 20926:
1, 20740: 1, 20720: 1, 20299: 1, 20235: 1, 20132: 1, 19926: 1, 19672: 1, 19200:
1, 18998: 1, 18889: 1, 18811: 1, 18802: 1, 18686: 1, 18606: 1, 18500: 1, 18345:
1, 18256: 1, 18222: 1, 18212: 1, 18149: 1, 18015: 1, 17947: 1, 17811: 1, 17670:
1, 17556: 1, 17346: 1, 17317: 1, 17310: 1, 17177: 1, 17167: 1, 17159: 1, 17081:
1, 16873: 1, 16834: 1, 16757: 1, 16619: 1, 16452: 1, 16152: 1, 15926: 1, 15898:
1, 15809: 1, 15752: 1, 15719: 1, 15697: 1, 15605: 1, 15597: 1, 15528: 1, 15260:
1, 15102: 1, 15012: 1, 14995: 1, 14707: 1, 14678: 1, 14639: 1, 14605: 1, 14475:
1, 14335: 1, 13972: 1, 13924: 1, 13917: 1, 13891: 1, 13609: 1, 13589: 1, 13527:
1, 13487: 1, 13380: 1, 13330: 1, 13297: 1, 13169: 1, 13049: 1, 13035: 1, 12962:
1, 12947: 1, 12936: 1, 12815: 1, 12655: 1, 12649: 1, 12639: 1, 12629: 1, 12591:
1, 12580: 1, 12494: 1, 12451: 1, 12437: 1, 12377: 1, 12342: 1, 12278: 1, 12258:
1, 12246: 1, 12233: 1, 12231: 1, 12215: 1, 12214: 1, 12185: 1, 12034: 1, 12024:
1, 12021: 1, 11979: 1, 11970: 1, 11899: 1, 11765: 1, 11614: 1, 11613: 1, 11608:
1, 11599: 1, 11595: 1, 11429: 1, 11418: 1, 11399: 1, 11251: 1, 11199: 1, 11171:
1, 11165: 1, 11037: 1, 10972: 1, 10948: 1, 10940: 1, 10937: 1, 10897: 1, 10881:
1, 10807: 1, 10753: 1, 10617: 1, 10521: 1, 10446: 1, 10434: 1, 10377: 1, 10315:
1, 10257: 1, 10252: 1, 10205: 1, 10151: 1, 10124: 1, 10103: 1, 10094: 1, 10075:
1, 10043: 1, 10025: 1, 10022: 1, 9972: 1, 9933: 1, 9916: 1, 9849: 1, 9783: 1,
9740: 1, 9732: 1, 9731: 1, 9651: 1, 9590: 1, 9533: 1, 9524: 1, 9491: 1, 9448: 1,
9415: 1, 9379: 1, 9345: 1, 9310: 1, 9246: 1, 9239: 1, 9226: 1, 9200: 1, 9184: 1,
9146: 1, 9109: 1, 9040: 1, 9026: 1, 9020: 1, 8973: 1, 8954: 1, 8871: 1, 8860: 1,
8857: 1, 8837: 1, 8825: 1, 8780: 1, 8709: 1, 8707: 1, 8692: 1, 8683: 1, 8606: 1,
8568: 1, 8534: 1, 8506: 1, 8484: 1, 8457: 1, 8409: 1, 8406: 1, 8325: 1, 8312: 1,
8304: 1, 8302: 1, 8258: 1, 8227: 1, 8206: 1, 8187: 1, 8157: 1, 8150: 1, 8134: 1,
8119: 1, 8118: 1, 8101: 1, 8084: 1, 8070: 1, 8046: 1, 8031: 1, 7998: 1, 7997: 1,
7941: 1, 7909: 1, 7907: 1, 7872: 1, 7867: 1, 7858: 1, 7840: 1, 7837: 1, 7798: 1,
7709: 1, 7697: 1, 7687: 1, 7685: 1, 7629: 1, 7620: 1, 7567: 1, 7511: 1, 7504: 1,
7496: 1, 7490: 1, 7457: 1, 7453: 1, 7437: 1, 7346: 1, 7344: 1, 7342: 1, 7341: 1,
7335: 1, 7311: 1, 7305: 1, 7299: 1, 7293: 1, 7282: 1, 7227: 1, 7197: 1, 7191: 1,
7152: 1, 7130: 1, 7121: 1, 7111: 1, 7103: 1, 7086: 1, 7056: 1, 7035: 1, 7034: 1,

7015: 1, 6987: 1, 6984: 1, 6983: 1, 6970: 1, 6964: 1, 6962: 1, 6961: 1, 6937: 1,
6934: 1, 6924: 1, 6904: 1, 6876: 1, 6873: 1, 6825: 1, 6820: 1, 6804: 1, 6803: 1,
6786: 1, 6758: 1, 6730: 1, 6662: 1, 6661: 1, 6629: 1, 6529: 1, 6512: 1, 6499: 1,
6497: 1, 6489: 1, 6480: 1, 6479: 1, 6477: 1, 6476: 1, 6473: 1, 6458: 1, 6433: 1,
6432: 1, 6419: 1, 6411: 1, 6408: 1, 6383: 1, 6366: 1, 6359: 1, 6349: 1, 6346: 1,
6287: 1, 6282: 1, 6263: 1, 6259: 1, 6256: 1, 6228: 1, 6223: 1, 6203: 1, 6201: 1,
6174: 1, 6170: 1, 6154: 1, 6149: 1, 6134: 1, 6095: 1, 6092: 1, 6089: 1, 6076: 1,
6061: 1, 6055: 1, 6040: 1, 6029: 1, 6024: 1, 6016: 1, 6012: 1, 6011: 1, 6004: 1,
5993: 1, 5992: 1, 5955: 1, 5949: 1, 5908: 1, 5887: 1, 5861: 1, 5833: 1, 5814: 1,
5806: 1, 5805: 1, 5788: 1, 5787: 1, 5773: 1, 5753: 1, 5738: 1, 5736: 1, 5726: 1,
5714: 1, 5710: 1, 5681: 1, 5640: 1, 5622: 1, 5611: 1, 5580: 1, 5577: 1, 5561: 1,
5537: 1, 5531: 1, 5515: 1, 5513: 1, 5507: 1, 5492: 1, 5490: 1, 5478: 1, 5459: 1,
5453: 1, 5440: 1, 5436: 1, 5400: 1, 5388: 1, 5380: 1, 5330: 1, 5320: 1, 5318: 1,
5313: 1, 5299: 1, 5285: 1, 5284: 1, 5277: 1, 5272: 1, 5267: 1, 5265: 1, 5261: 1,
5233: 1, 5228: 1, 5205: 1, 5186: 1, 5171: 1, 5164: 1, 5149: 1, 5148: 1, 5134: 1,
5132: 1, 5129: 1, 5118: 1, 5102: 1, 5091: 1, 5090: 1, 5086: 1, 5081: 1, 5069: 1,
5064: 1, 5060: 1, 5057: 1, 5046: 1, 5037: 1, 5020: 1, 5019: 1, 4990: 1, 4979: 1,
4961: 1, 4951: 1, 4949: 1, 4938: 1, 4928: 1, 4927: 1, 4907: 1, 4895: 1, 4892: 1,
4866: 1, 4855: 1, 4854: 1, 4843: 1, 4830: 1, 4823: 1, 4815: 1, 4813: 1, 4808: 1,
4806: 1, 4803: 1, 4799: 1, 4796: 1, 4783: 1, 4779: 1, 4776: 1, 4773: 1, 4771: 1,
4770: 1, 4757: 1, 4754: 1, 4730: 1, 4728: 1, 4721: 1, 4715: 1, 4705: 1, 4703: 1,
4669: 1, 4660: 1, 4657: 1, 4615: 1, 4609: 1, 4605: 1, 4593: 1, 4581: 1, 4566: 1,
4556: 1, 4541: 1, 4538: 1, 4534: 1, 4526: 1, 4505: 1, 4501: 1, 4483: 1, 4480: 1,
4471: 1, 4465: 1, 4459: 1, 4443: 1, 4433: 1, 4425: 1, 4412: 1, 4403: 1, 4384: 1,
4379: 1, 4376: 1, 4369: 1, 4359: 1, 4355: 1, 4346: 1, 4300: 1, 4297: 1, 4296: 1,
4291: 1, 4286: 1, 4285: 1, 4280: 1, 4269: 1, 4263: 1, 4261: 1, 4256: 1, 4248: 1,
4247: 1, 4244: 1, 4239: 1, 4238: 1, 4237: 1, 4207: 1, 4202: 1, 4194: 1, 4193: 1,
4177: 1, 4169: 1, 4164: 1, 4162: 1, 4159: 1, 4147: 1, 4144: 1, 4134: 1, 4116: 1,
4110: 1, 4102: 1, 4100: 1, 4089: 1, 4087: 1, 4075: 1, 4063: 1, 4061: 1, 4060: 1,
4057: 1, 4032: 1, 4031: 1, 4030: 1, 4024: 1, 4023: 1, 4020: 1, 4009: 1, 3982: 1,
3978: 1, 3974: 1, 3965: 1, 3963: 1, 3947: 1, 3945: 1, 3943: 1, 3935: 1, 3930: 1,
3902: 1, 3900: 1, 3897: 1, 3892: 1, 3886: 1, 3881: 1, 3880: 1, 3878: 1, 3875: 1,
3853: 1, 3849: 1, 3842: 1, 3839: 1, 3834: 1, 3831: 1, 3824: 1, 3820: 1, 3814: 1,
3809: 1, 3800: 1, 3799: 1, 3797: 1, 3794: 1, 3793: 1, 3787: 1, 3777: 1, 3776: 1,
3763: 1, 3762: 1, 3761: 1, 3757: 1, 3754: 1, 3733: 1, 3725: 1, 3724: 1, 3722: 1,
3721: 1, 3708: 1, 3695: 1, 3691: 1, 3690: 1, 3686: 1, 3685: 1, 3682: 1, 3674: 1,
3669: 1, 3666: 1, 3664: 1, 3660: 1, 3656: 1, 3655: 1, 3640: 1, 3632: 1, 3625: 1,
3624: 1, 3610: 1, 3608: 1, 3595: 1, 3590: 1, 3586: 1, 3583: 1, 3582: 1, 3572: 1,
3570: 1, 3568: 1, 3561: 1, 3558: 1, 3557: 1, 3554: 1, 3549: 1, 3535: 1, 3533: 1,
3531: 1, 3529: 1, 3522: 1, 3519: 1, 3516: 1, 3512: 1, 3511: 1, 3506: 1, 3499: 1,
3495: 1, 3492: 1, 3484: 1, 3483: 1, 3481: 1, 3480: 1, 3479: 1, 3471: 1, 3470: 1,
3468: 1, 3455: 1, 3454: 1, 3447: 1, 3442: 1, 3440: 1, 3438: 1, 3434: 1, 3431: 1,
3430: 1, 3426: 1, 3420: 1, 3419: 1, 3418: 1, 3404: 1, 3388: 1, 3378: 1, 3375: 1,
3374: 1, 3367: 1, 3366: 1, 3363: 1, 3362: 1, 3361: 1, 3354: 1, 3353: 1, 3343: 1,
3341: 1, 3331: 1, 3324: 1, 3319: 1, 3312: 1, 3305: 1, 3304: 1, 3298: 1, 3295: 1,
3291: 1, 3281: 1, 3276: 1, 3274: 1, 3271: 1, 3259: 1, 3252: 1, 3250: 1, 3240: 1,
3232: 1, 3225: 1, 3223: 1, 3220: 1, 3219: 1, 3218: 1, 3212: 1, 3211: 1, 3208: 1,
3198: 1, 3194: 1, 3191: 1, 3186: 1, 3179: 1, 3160: 1, 3159: 1, 3156: 1, 3146: 1,

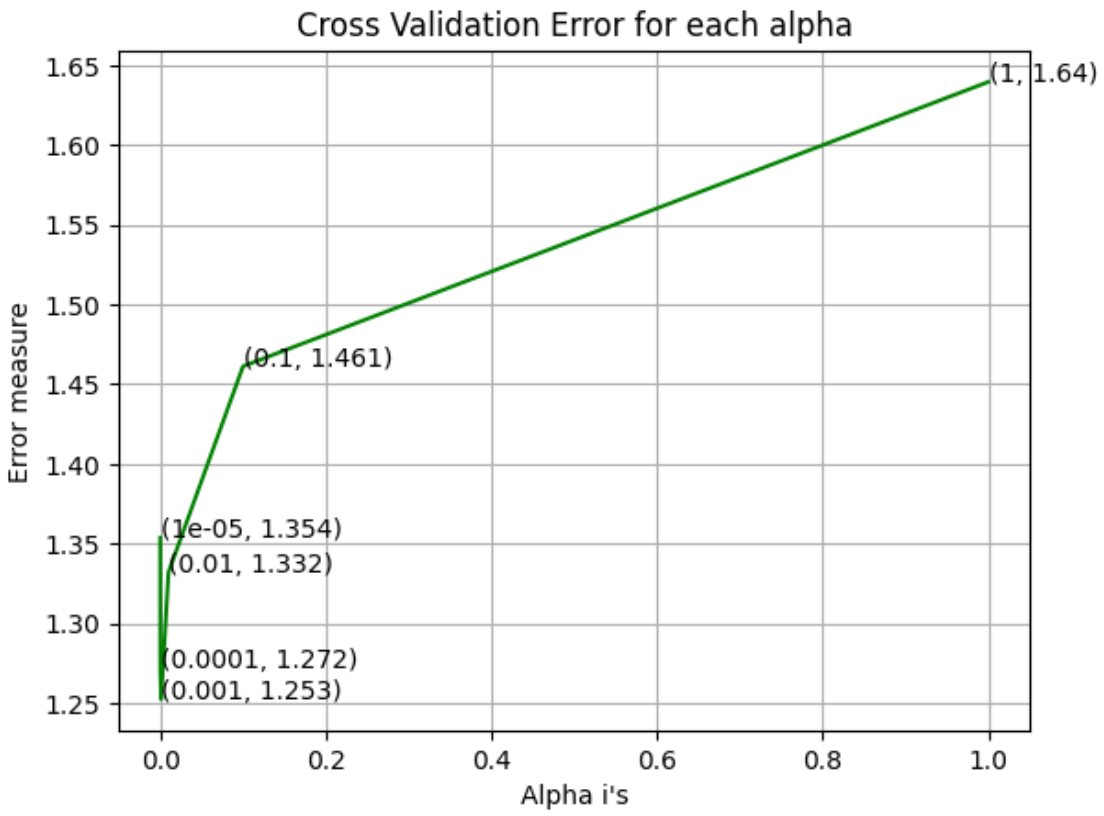
3138: 1, 3129: 1, 3123: 1, 3122: 1, 3114: 1, 3106: 1, 3105: 1, 3103: 1, 3100: 1,
3091: 1, 3088: 1, 3086: 1, 3066: 1, 3064: 1, 3056: 1, 3046: 1, 3044: 1, 3041: 1,
3037: 1, 3029: 1, 3025: 1, 3018: 1, 3016: 1, 3015: 1, 3014: 1, 3007: 1, 3004: 1,
3002: 1, 3000: 1, 2993: 1, 2989: 1, 2983: 1, 2982: 1, 2981: 1, 2977: 1, 2970: 1,
2961: 1, 2958: 1, 2950: 1, 2942: 1, 2940: 1, 2938: 1, 2922: 1, 2919: 1, 2917: 1,
2906: 1, 2905: 1, 2904: 1, 2901: 1, 2896: 1, 2892: 1, 2879: 1, 2869: 1, 2863: 1,
2856: 1, 2852: 1, 2851: 1, 2845: 1, 2843: 1, 2842: 1, 2830: 1, 2826: 1, 2823: 1,
2814: 1, 2801: 1, 2797: 1, 2791: 1, 2782: 1, 2780: 1, 2778: 1, 2773: 1, 2762: 1,
2750: 1, 2746: 1, 2743: 1, 2739: 1, 2734: 1, 2731: 1, 2727: 1, 2725: 1, 2722: 1,
2718: 1, 2711: 1, 2707: 1, 2706: 1, 2703: 1, 2702: 1, 2700: 1, 2693: 1, 2692: 1,
2691: 1, 2685: 1, 2680: 1, 2678: 1, 2675: 1, 2673: 1, 2671: 1, 2670: 1, 2668: 1,
2666: 1, 2663: 1, 2658: 1, 2656: 1, 2655: 1, 2649: 1, 2641: 1, 2633: 1, 2631: 1,
2630: 1, 2626: 1, 2618: 1, 2601: 1, 2595: 1, 2594: 1, 2585: 1, 2582: 1, 2581: 1,
2579: 1, 2578: 1, 2577: 1, 2576: 1, 2574: 1, 2573: 1, 2566: 1, 2547: 1, 2543: 1,
2540: 1, 2536: 1, 2529: 1, 2519: 1, 2514: 1, 2508: 1, 2507: 1, 2506: 1, 2505: 1,
2502: 1, 2499: 1, 2497: 1, 2496: 1, 2494: 1, 2486: 1, 2483: 1, 2482: 1, 2481: 1,
2480: 1, 2478: 1, 2473: 1, 2472: 1, 2469: 1, 2468: 1, 2466: 1, 2462: 1, 2454: 1,
2452: 1, 2451: 1, 2449: 1, 2448: 1, 2447: 1, 2443: 1, 2438: 1, 2437: 1, 2436: 1,
2428: 1, 2422: 1, 2421: 1, 2417: 1, 2416: 1, 2413: 1, 2406: 1, 2388: 1, 2384: 1,
2377: 1, 2372: 1, 2367: 1, 2364: 1, 2361: 1, 2360: 1, 2355: 1, 2353: 1, 2345: 1,
2344: 1, 2341: 1, 2338: 1, 2335: 1, 2331: 1, 2330: 1, 2323: 1, 2322: 1, 2320: 1,
2319: 1, 2317: 1, 2316: 1, 2314: 1, 2309: 1, 2305: 1, 2301: 1, 2299: 1, 2297: 1,
2295: 1, 2290: 1, 2289: 1, 2286: 1, 2284: 1, 2279: 1, 2278: 1, 2277: 1, 2276: 1,
2273: 1, 2268: 1, 2261: 1, 2254: 1, 2253: 1, 2247: 1, 2244: 1, 2243: 1, 2242: 1,
2238: 1, 2237: 1, 2233: 1, 2232: 1, 2231: 1, 2228: 1, 2227: 1, 2220: 1, 2219: 1,
2214: 1, 2201: 1, 2200: 1, 2196: 1, 2195: 1, 2188: 1, 2184: 1, 2183: 1, 2180: 1,
2176: 1, 2174: 1, 2170: 1, 2169: 1, 2168: 1, 2166: 1, 2152: 1, 2147: 1, 2146: 1,
2142: 1, 2141: 1, 2140: 1, 2132: 1, 2131: 1, 2129: 1, 2120: 1, 2119: 1, 2116: 1,
2110: 1, 2107: 1, 2105: 1, 2101: 1, 2097: 1, 2095: 1, 2089: 1, 2086: 1, 2083: 1,
2079: 1, 2078: 1, 2076: 1, 2075: 1, 2071: 1, 2069: 1, 2054: 1, 2050: 1, 2049: 1,
2046: 1, 2045: 1, 2041: 1, 2033: 1, 2032: 1, 2029: 1, 2027: 1, 2026: 1, 2021: 1,
2016: 1, 2015: 1, 2013: 1, 2005: 1, 2002: 1, 1999: 1, 1996: 1, 1994: 1, 1990: 1,
1988: 1, 1986: 1, 1983: 1, 1981: 1, 1977: 1, 1974: 1, 1970: 1, 1969: 1, 1968: 1,
1966: 1, 1961: 1, 1957: 1, 1956: 1, 1951: 1, 1950: 1, 1946: 1, 1936: 1, 1934: 1,
1931: 1, 1930: 1, 1926: 1, 1925: 1, 1919: 1, 1918: 1, 1914: 1, 1907: 1, 1905: 1,
1902: 1, 1900: 1, 1898: 1, 1896: 1, 1893: 1, 1891: 1, 1887: 1, 1882: 1, 1879: 1,
1876: 1, 1870: 1, 1869: 1, 1867: 1, 1865: 1, 1863: 1, 1860: 1, 1854: 1, 1852: 1,
1849: 1, 1847: 1, 1845: 1, 1844: 1, 1839: 1, 1837: 1, 1836: 1, 1834: 1, 1832: 1,
1830: 1, 1828: 1, 1827: 1, 1826: 1, 1825: 1, 1824: 1, 1820: 1, 1817: 1, 1815: 1,
1814: 1, 1812: 1, 1804: 1, 1803: 1, 1798: 1, 1794: 1, 1786: 1, 1783: 1, 1779: 1,
1777: 1, 1775: 1, 1773: 1, 1772: 1, 1769: 1, 1768: 1, 1767: 1, 1763: 1, 1761: 1,
1759: 1, 1757: 1, 1754: 1, 1748: 1, 1745: 1, 1744: 1, 1743: 1, 1741: 1, 1740: 1,
1738: 1, 1737: 1, 1732: 1, 1729: 1, 1726: 1, 1724: 1, 1723: 1, 1721: 1, 1720: 1,
1719: 1, 1714: 1, 1712: 1, 1705: 1, 1701: 1, 1700: 1, 1699: 1, 1695: 1, 1691: 1,
1690: 1, 1689: 1, 1682: 1, 1681: 1, 1679: 1, 1671: 1, 1663: 1, 1662: 1, 1660: 1,
1659: 1, 1653: 1, 1652: 1, 1650: 1, 1648: 1, 1647: 1, 1646: 1, 1642: 1, 1638: 1,
1633: 1, 1628: 1, 1625: 1, 1624: 1, 1613: 1, 1612: 1, 1611: 1, 1610: 1, 1609: 1,
1608: 1, 1605: 1, 1601: 1, 1594: 1, 1593: 1, 1591: 1, 1588: 1, 1584: 1, 1580: 1,

```
1578: 1, 1576: 1, 1573: 1, 1569: 1, 1568: 1, 1565: 1, 1559: 1, 1552: 1, 1545: 1,
1543: 1, 1541: 1, 1539: 1, 1537: 1, 1527: 1, 1526: 1, 1524: 1, 1522: 1, 1517: 1,
1516: 1, 1515: 1, 1513: 1, 1510: 1, 1509: 1, 1508: 1, 1505: 1, 1503: 1, 1502: 1,
1500: 1, 1499: 1, 1493: 1, 1491: 1, 1487: 1, 1485: 1, 1484: 1, 1482: 1, 1478: 1,
1474: 1, 1473: 1, 1472: 1, 1471: 1, 1469: 1, 1468: 1, 1465: 1, 1462: 1, 1461: 1,
1460: 1, 1457: 1, 1454: 1, 1452: 1, 1451: 1, 1448: 1, 1441: 1, 1439: 1, 1436: 1,
1432: 1, 1428: 1, 1426: 1, 1424: 1, 1423: 1, 1420: 1, 1419: 1, 1417: 1, 1412: 1,
1410: 1, 1409: 1, 1407: 1, 1406: 1, 1405: 1, 1404: 1, 1403: 1, 1401: 1, 1395: 1,
1389: 1, 1388: 1, 1385: 1, 1381: 1, 1380: 1, 1378: 1, 1365: 1, 1364: 1, 1363: 1,
1361: 1, 1355: 1, 1354: 1, 1350: 1, 1342: 1, 1341: 1, 1339: 1, 1338: 1, 1337: 1,
1336: 1, 1334: 1, 1333: 1, 1332: 1, 1328: 1, 1323: 1, 1320: 1, 1315: 1, 1314: 1,
1310: 1, 1308: 1, 1305: 1, 1302: 1, 1300: 1, 1298: 1, 1295: 1, 1292: 1, 1289: 1,
1282: 1, 1280: 1, 1279: 1, 1276: 1, 1272: 1, 1269: 1, 1268: 1, 1265: 1, 1264: 1,
1263: 1, 1259: 1, 1258: 1, 1256: 1, 1252: 1, 1250: 1, 1244: 1, 1242: 1, 1241: 1,
1239: 1, 1236: 1, 1234: 1, 1232: 1, 1229: 1, 1228: 1, 1227: 1, 1225: 1, 1220: 1,
1218: 1, 1216: 1, 1215: 1, 1212: 1, 1208: 1, 1200: 1, 1197: 1, 1194: 1, 1187: 1,
1186: 1, 1183: 1, 1181: 1, 1178: 1, 1173: 1, 1171: 1, 1169: 1, 1164: 1, 1153: 1,
1152: 1, 1151: 1, 1147: 1, 1145: 1, 1140: 1, 1139: 1, 1135: 1, 1134: 1, 1132: 1,
1131: 1, 1129: 1, 1123: 1, 1120: 1, 1118: 1, 1117: 1, 1115: 1, 1114: 1, 1112: 1,
1110: 1, 1109: 1, 1108: 1, 1106: 1, 1105: 1, 1103: 1, 1094: 1, 1090: 1, 1082: 1,
1076: 1, 1073: 1, 1072: 1, 1071: 1, 1067: 1, 1060: 1, 1058: 1, 1057: 1, 1056: 1,
1046: 1, 1044: 1, 1042: 1, 1041: 1, 1038: 1, 1034: 1, 1031: 1, 1028: 1, 1027: 1,
1022: 1, 1019: 1, 1008: 1, 1007: 1, 1003: 1, 1000: 1, 995: 1, 989: 1, 988: 1,
985: 1, 979: 1, 974: 1, 961: 1, 960: 1, 959: 1, 957: 1, 955: 1, 949: 1, 944: 1,
942: 1, 941: 1, 938: 1, 937: 1, 935: 1, 934: 1, 933: 1, 921: 1, 920: 1, 916: 1,
915: 1, 914: 1, 908: 1, 899: 1, 898: 1, 896: 1, 885: 1, 884: 1, 883: 1, 876: 1,
874: 1, 873: 1, 868: 1, 865: 1, 864: 1, 862: 1, 861: 1, 860: 1, 858: 1, 857: 1,
855: 1, 848: 1, 845: 1, 844: 1, 843: 1, 842: 1, 839: 1, 838: 1, 835: 1, 834: 1,
828: 1, 819: 1, 812: 1, 809: 1, 808: 1, 796: 1, 791: 1, 790: 1, 784: 1, 782: 1,
762: 1, 761: 1, 754: 1, 750: 1, 742: 1, 736: 1, 725: 1, 724: 1, 713: 1, 705: 1,
703: 1, 702: 1, 697: 1, 692: 1, 689: 1, 678: 1, 671: 1, 660: 1, 657: 1, 643: 1,
634: 1, 630: 1, 628: 1, 609: 1, 608: 1, 598: 1, 581: 1, 579: 1, 578: 1, 563: 1,
556: 1, 551: 1, 549: 1, 521: 1, 506: 1, 495: 1, 443: 1})
```

[ ]:
```
# Train a Logistic regression+Calibration model using text features which are one-hot encoded
alpha = [10 ** x for x in range(-5, 1)]


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_loss(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, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.353819061062105
For values of alpha =  0.0001 The log loss is: 1.271828336426046
For values of alpha =  0.001 The log loss is: 1.2525566201030984
For values of alpha =  0.01 The log loss is: 1.3321916970665693
For values of alpha =  0.1 The log loss is: 1.4612718437887025
For values of alpha =  1 The log loss is: 1.6395403031269695
```




```
For values of best alpha =  0.001 The train log loss is: 0.6390254563122633
For values of best alpha =  0.001 The cross validation log loss is:
1.2525566201030984
For values of best alpha =  0.001 The test log loss is: 1.1870016522468347
```

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

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

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

[ ]:
```
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")
```

```
98.088 % of word of test data appeared in train data
98.643 % of word of Cross Validation appeared in train data
```

Machine Learning Models

[ ]:

[ ]:
```
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 willl 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)
```

[ ]:
```
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)
```

Stacking the three types of features

[ ]:
```
train_gene_var_onehotCoding =␣
↪hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =␣
↪hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding =␣
↪hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,␣
↪train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding,␣
↪test_text_feature_onehotCoding)).tocsr()
```

```
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding,␣
 ↪cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.
 ↪hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.
 ↪hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.
 ↪hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,␣
 ↪train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding,␣
 ↪test_text_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding,␣
 ↪cv_text_feature_responseCoding))
```

[ ]:
```
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, 55467)
(number of data points * number of features) in test data =  (665, 55467)
(number of data points * number of features) in cross validation data = (532,
55467)
```

[ ]:
```
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)

[ ]:

Base Line Model

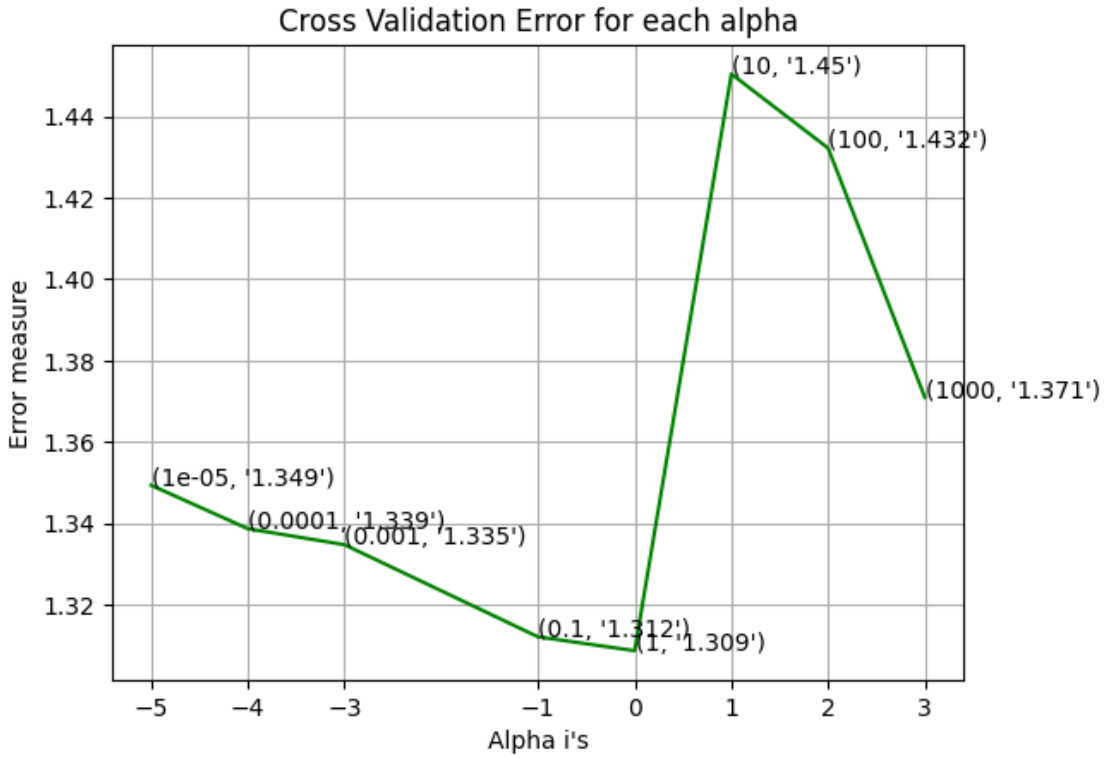
Naive Bayes

Hyper parameter tuning

[ ]:
```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    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))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
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 = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
```

```
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
  ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.349351598077089
for alpha = 0.0001
Log Loss : 1.3386705133214356
for alpha = 0.001
Log Loss : 1.3347289719449014
for alpha = 0.1
Log Loss : 1.3120884767063512
for alpha = 1
Log Loss : 1.30867195357826
for alpha = 10
Log Loss : 1.45045524960535
for alpha = 100
Log Loss : 1.4322214466994583
for alpha = 1000
Log Loss : 1.370969318343749
```




```
For values of best alpha =  1 The train log loss is: 0.8739404518769875
For values of best alpha =  1 The cross validation log loss is: 1.30867195357826
```

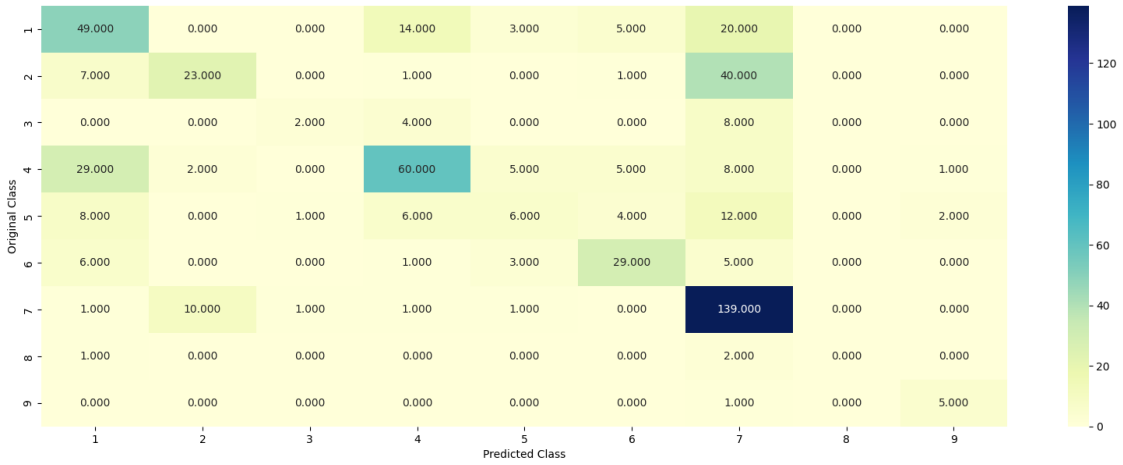
```
For values of best alpha =  1 The test log loss is: 1.287887885983238
```

[ ]:

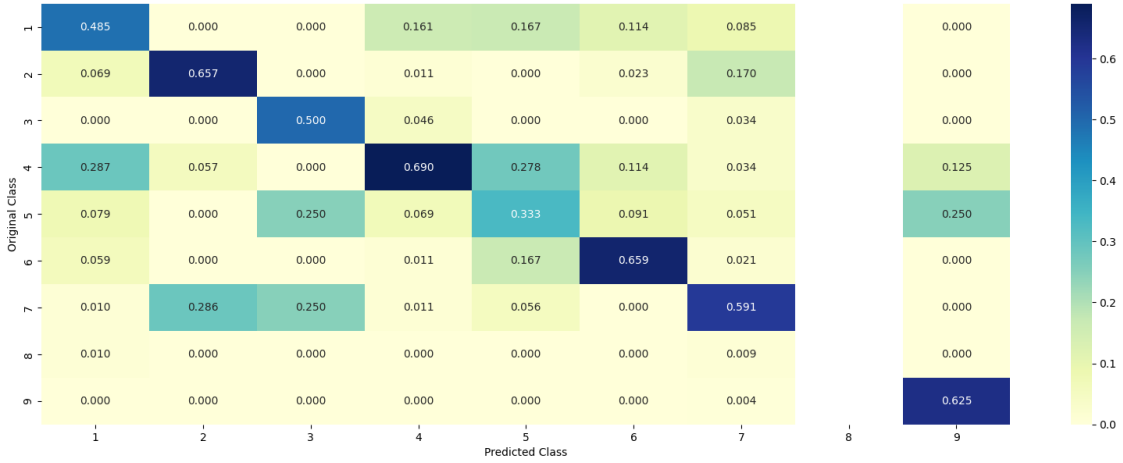
Testing the model with best hyper paramters

[ ]:
```
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)
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified 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()))
```

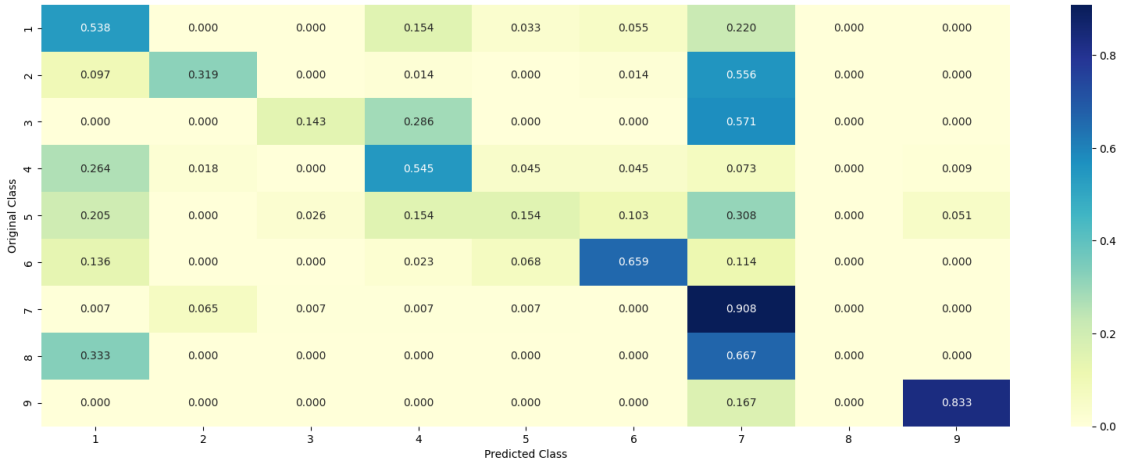
```
Log Loss : 1.30867195357826
Number of missclassified point : 0.4116541353383459
--------------------- Confusion matrix ----------------------
```




```
--------------------- Precision matrix (Columm Sum=1) --------------------
```



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



[ ]:
```
test_point_index = 1
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])
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0712 0.5165 0.024  0.0909 0.0507 0.0407
0.1936 0.007  0.0054]]
Actual Class : 2
```

[ ]:
```
test_point_index = 3
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])
```

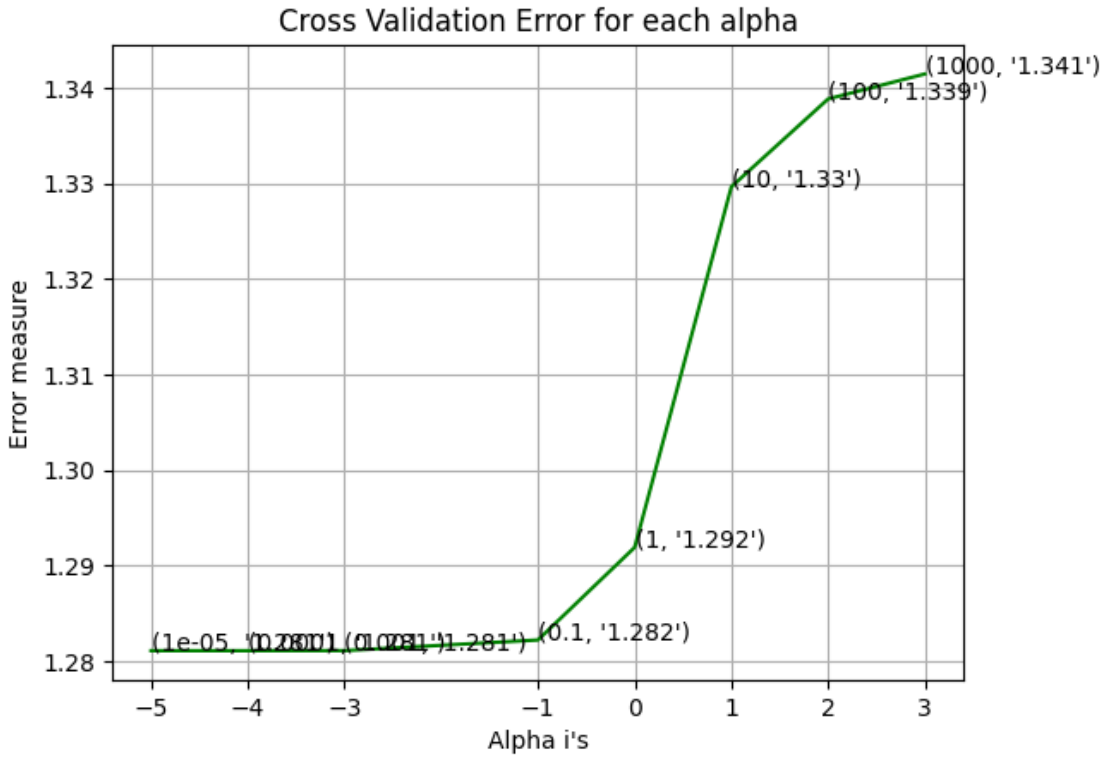
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0685 0.1678 0.0231 0.0873 0.0489 0.0389
0.5537 0.0067 0.0051]]
Actual Class : 6
```

[ ]:
```
```

[ ]:
```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    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))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
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 = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_responseCoding)
```

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

```
for alpha = 1e-05
Log Loss : 1.2810704016940393
for alpha = 0.0001
Log Loss : 1.2810712385737504
for alpha = 0.001
Log Loss : 1.2810796326570222
for alpha = 0.1
Log Loss : 1.2821925930851614
for alpha = 1
Log Loss : 1.291892771697416
for alpha = 10
Log Loss : 1.329666860200851
for alpha = 100
Log Loss : 1.3388597000346156
for alpha = 1000
Log Loss : 1.3414766214022258
```

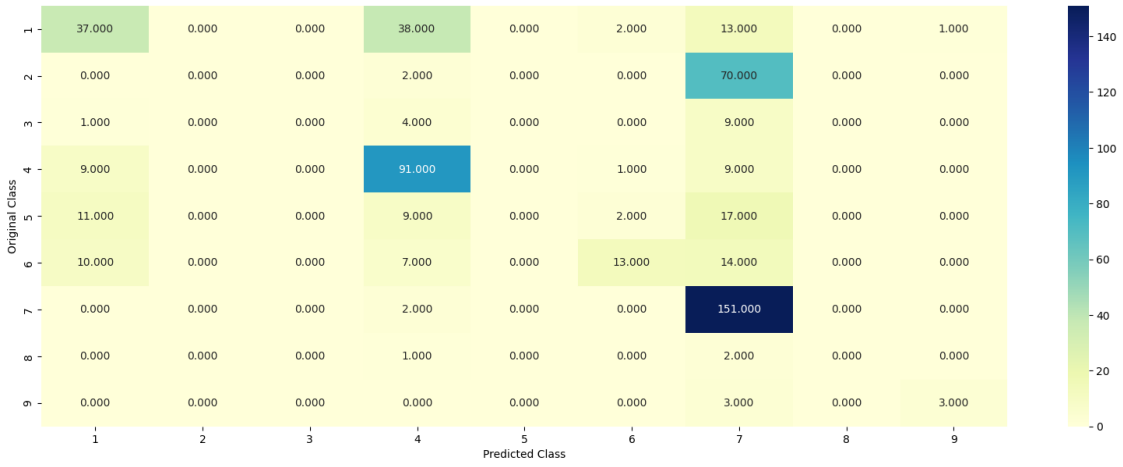


```
For values of best alpha =  1e-05 The train log loss is: 1.1906045128374896
For values of best alpha =  1e-05 The cross validation log loss is:
1.2810704016940393
For values of best alpha =  1e-05 The test log loss is: 1.2554490510644907
```

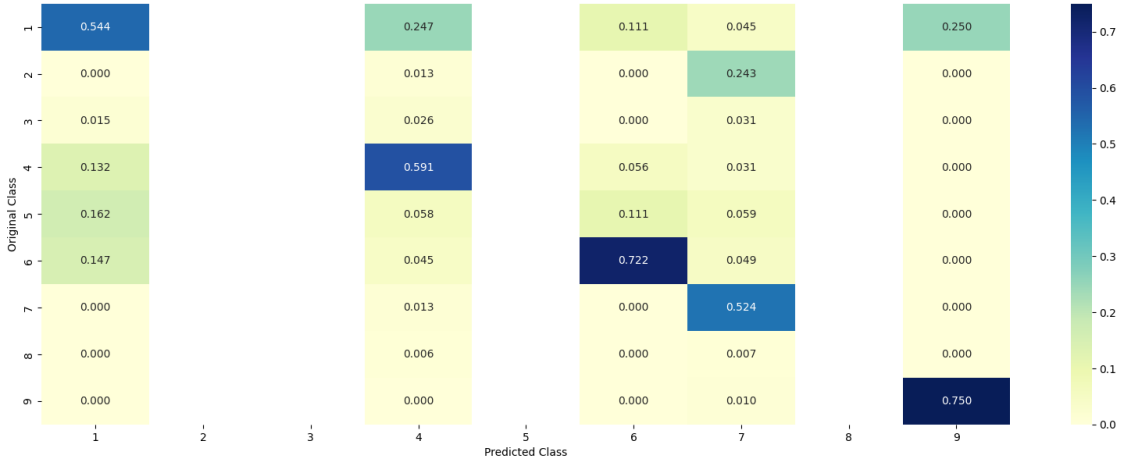
[ ]:

[ ]:
```
clf = MultinomialNB(alpha=alpha[best_alpha])
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)
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.
 ↪predict(cv_x_responseCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_responseCoding) )
```

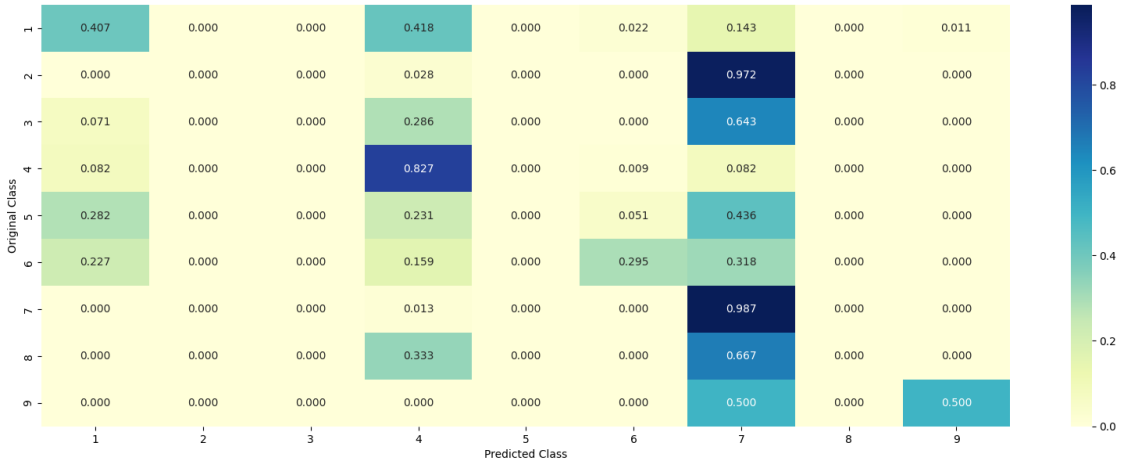
```
Log Loss : 1.2810704016940393
Number of missclassified point : 0.44548872180451127
--------------------- Confusion matrix ----------------------
```




```
---------------------- Precision matrix (Columm Sum=1) --------------------
```



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



K Nearest Neighbour Classification

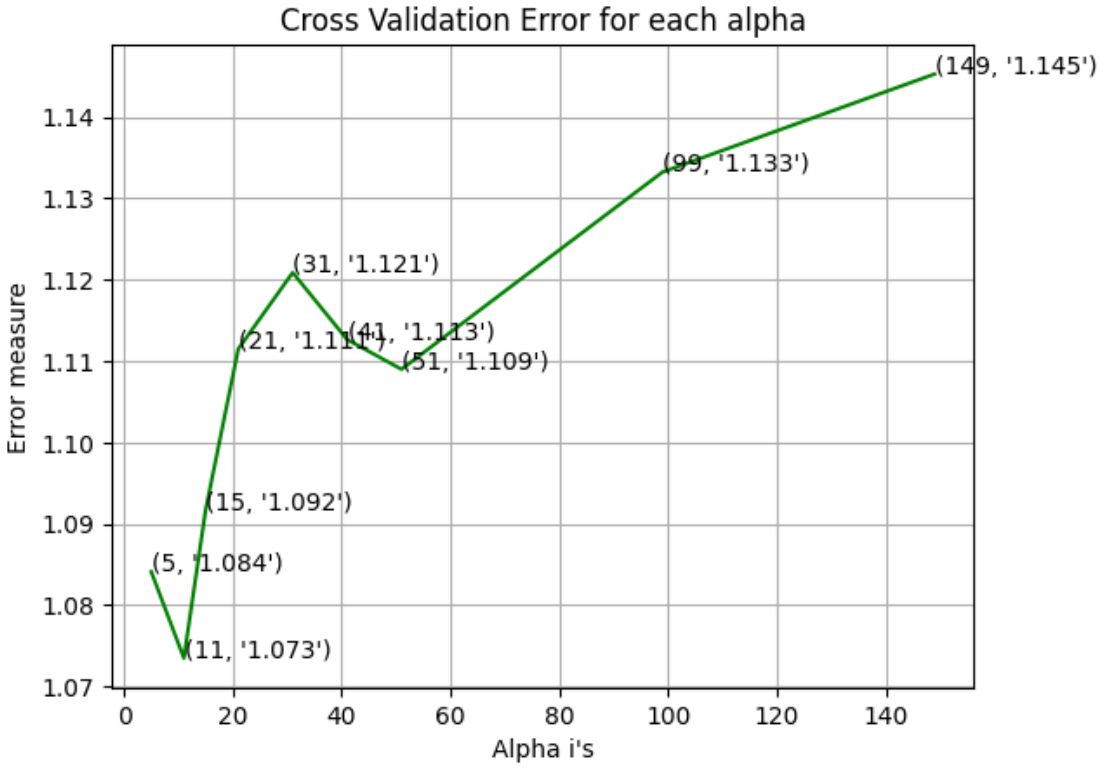
Hyper parameter tuning

[ ]:
```
alpha = [5, 11, 15, 21, 31, 41, 51, 99, 149]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    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))

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

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

```
for alpha = 5
Log Loss : 1.0841160228129867
for alpha = 11
Log Loss : 1.073497139588747
for alpha = 15
Log Loss : 1.09172680535336
for alpha = 21
Log Loss : 1.1114920692571573
for alpha = 31
Log Loss : 1.120880746938688
for alpha = 41
Log Loss : 1.1126578021128717
for alpha = 51
Log Loss : 1.1090016799367606
for alpha = 99
Log Loss : 1.1332401949701956
```

```
for alpha = 149
Log Loss : 1.1452801101299286
```

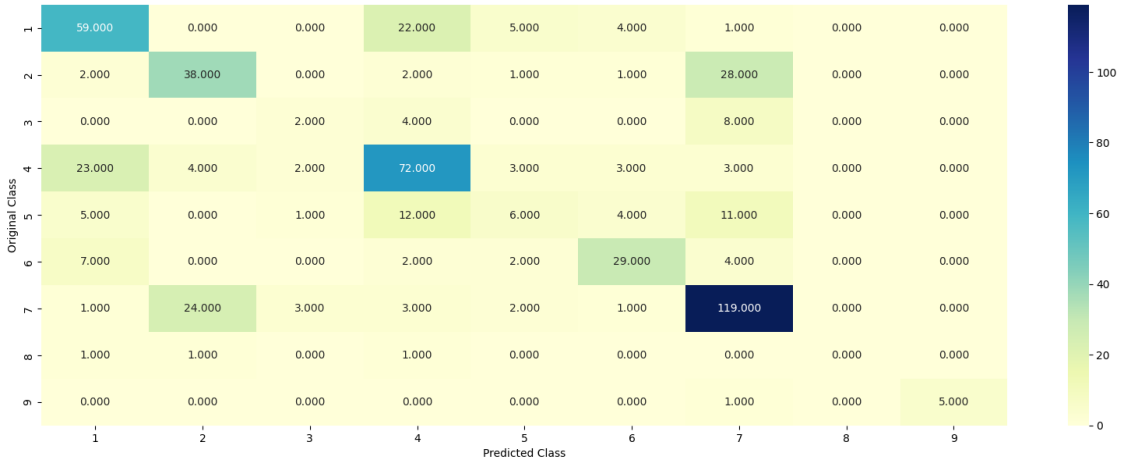



```
For values of best alpha =  11 The train log loss is: 0.6051422727794165
For values of best alpha =  11 The cross validation log loss is:
1.073497139588747
For values of best alpha =  11 The test log loss is: 1.0556543349150707
```

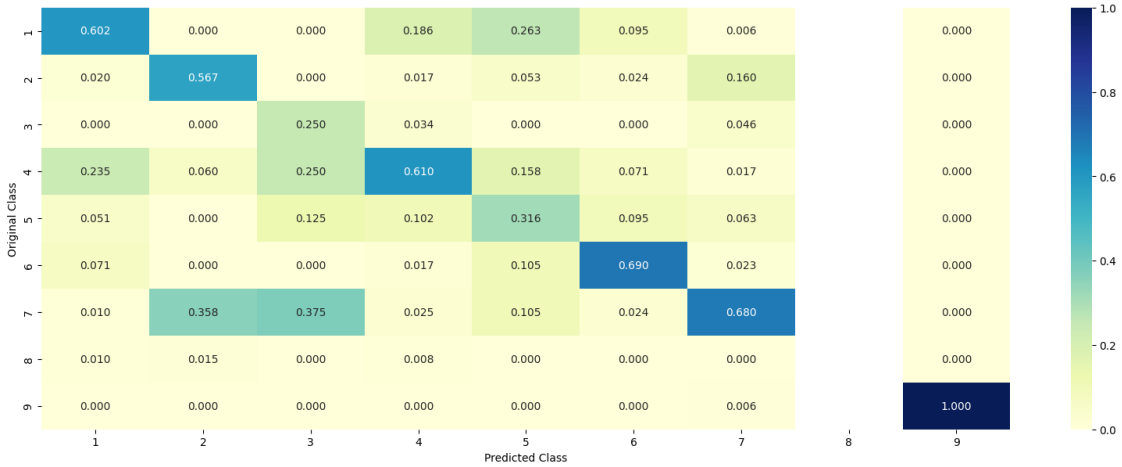
Testing the model with best hyper paramters

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

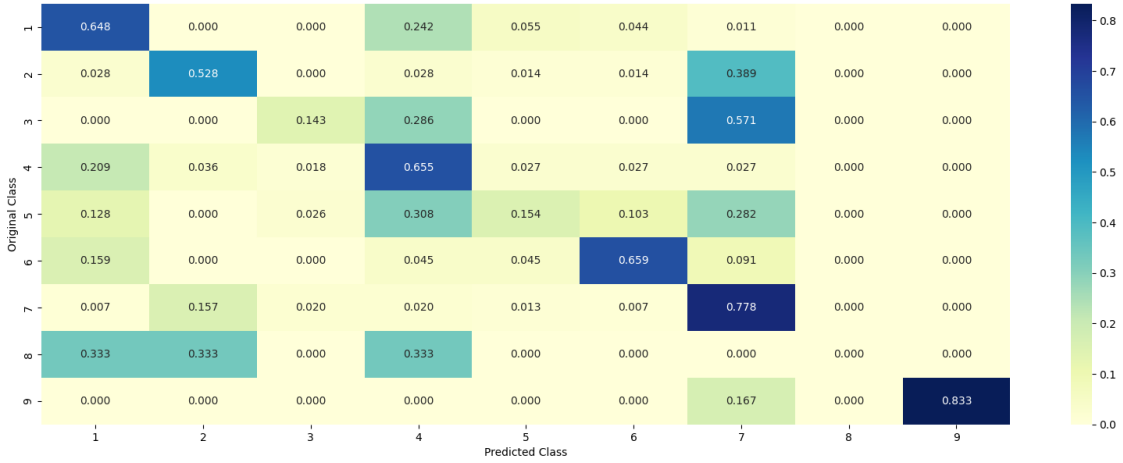
```
Log loss : 1.073497139588747
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```



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



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



Sample Query point -1

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

test_point_index = 40

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

```
Predicted Class : 7
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [2 7 7 7 6 6 2
6 7 7 2]
Fequency of nearest points : Counter({7: 5, 2: 3, 6: 3})
```

Sample Query Point-2

[ ]:
```
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 = 143
```

```
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("Fequency 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 [4 4 1 4 4 4 3 4 4 4 1]
Fequency of nearest points : Counter({4: 8, 1: 2, 3: 1})
```

8.0.1 KNN WITH ONEHOT ENCODING

[ ]:
```
alpha = [5, 11, 15, 21, 31, 41, 51, 99, 149]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    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))

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

```
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 5
```