Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

```
ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for
```

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

```
# Importing necessary Libraries
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import pickle
import os
import math
import sqlite3
import string
import itertools
from collections import Counter, defaultdict, OrderedDict
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from mpl_toolkits.mplot3d import Axes3D
```

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import contractions

from sklearn.decomposition import TruncatedSVD
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn import model_selection
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer
from sklearn import metrics
from sklearn.model_selection import train_test_split, cross_val_score, RandomizedSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc, roc_auc_score, normalized_mutual_info_score
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.ensemble import RandomForestClassifier,VotingClassifier

from scipy.sparse import hstack
from mlxtend.classifier import StackingClassifier
from imblearn.over_sampling import SMOTE

from tqdm import tqdm
from prettytable import PrettyTable
```

```
Using TensorFlow backend.
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
df_1 = pd.read_csv(r'training_variants')
print(df_1.shape)
df_1.head()
```

```
(3321, 4)
```

Out[2]:

	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

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

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

3.1.2. Reading Text Data

In [3]:

```
df_2 = pd.read_csv(r'training_text', sep= '\|\|', names= ['ID', 'Text'], skiprows= 1) # check the seperator already given in sample text in 2.1.2
print(df_2.shape)
df_2.head()
```

(3321, 2)

Out[3]:

	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...

In [4]:

```
df_2[df_2['Text'].isnull()]
```

Out[4]:

	ID	Text
1109	1109	NaN
1277	1277	NaN
1407	1407	NaN
1639	1639	NaN
2755	2755	NaN

In [5]:

```
df_2.dropna(inplace = True)
```

In [6]:

```
df_2[df_2['Text'].isnull()]
```

Out[6]:

ID	Text

3.1.3. Preprocessing of text

In [7]:

```
df_2['Text'][0]
```

Out[7]:

"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 erythro blastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The pre cise 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 mut ations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malfor mations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 sile ncing 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 rol e 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 discovere d 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 explo ration 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 u nknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK path way, and induces resistance of MCF7 cells to tamoxifen (6).Here, we deorphanize CDK10 by identifying cyclin M, the product of FAM58A, as a bindi ng partner. Mutations in this gene that predict absence or truncation of cyclin M are associated with STAR syndrome, whose features include toe syn dactyly, telecanthus, and anogenital and renal malformations in heterozygous females (10). However, both the functions of cyclin M and the pathoge

nesis 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 re sistance in estrogen receptor (ER)+ breast cancer cells. We show that CDK10/cyclin M positively controls ETS2 degradation by the proteasome, thro ugh 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 hom ology 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 interactio n phenotype with full-length cyclin M (expressed as a prey protein) but no detectable phenotype with cyclin D1, p21 (CIP1), and Cdi1 (KAP), which ar e known binding partners of other CDKs (Fig. 1B). CDK1 and CDK3 also produced Y2H signals with cyclin M, albeit notably weaker than that observe d with CDK10 (Fig. 1B). An interaction phenotype was also observed between full-length cyclin M and CDK10 proteins expressed as bait and prey, re spectively (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 iso forms 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 inter action complex. (A) Schematic representation of the different protein isoforms analyzed by Y2H assays. Amino acid numbers are indicated. Black bo xes indicate internal deletions. The red box indicates a differing amino acid sequence compared with CDK10 P1. (B) Y2H assay between a set of CD K proteins expressed as baits (in fusion to the LexA DNA binding domain) and CDK interacting proteins expressed as preys (in fusion to the B42 tran scriptional 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-te rminal 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 e xpression levels in transfected HEK293 cells. (E) Western blot analysis of Myc-CDK10 (wt or kd) immunoprecipitates obtained using the anti-Myc anti body. “Inputs” correspond to 10 µg total lysates obtained from HEK293 cells coexpressing Myc-CDK10 (wt or kd) and CycM-V5-6His. (F) Western blo t analysis of immunoprecipitates obtained using the anti-CDK10 antibody or a control goat antibody, from human breast cancer MCF7 cells. “Input” c orresponds to 30 µg 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 signifi cantly increased upon coexpression with Myc-CDK10 (wt or kd) and, to a lesser extent, that of Myc-CDK10 (wt or kd) was increased upon coexpressi on 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.P 2 isoform and cyclin M (Fig. S1C). To detect the interaction between endogenous proteins, we performed immunoprecipitations on nontransfected M CF7 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 c opurified, thus confirming their interaction in yet another cellular model (Fig. 2A). We then performed in vitro kinase assays with purified proteins, usin g histone H1 as a generic substrate. Histone H1 phosphorylation was detected only from lysates of cells coexpressing GST-CDK10 and StrepII-cycli n 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 o f CDK10/cyclin M. We detected strong phosphorylation of ETS2 by the GST-CDK10/StrepII-cyclin M purified heterodimer, whereas no phosphorylatio n 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.CD K10 is a cyclin M-dependent protein kinase. (A) In vitro protein kinase assay on histone H1. Lysates from insect cells expressing different proteins we re purified on a glutathione Sepharose matrix to capture GST-CDK10(wt or kd) fusion proteins alone, or in complex with STR-CycM fusion protein. P urified 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 differ ent 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 resu lts, 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 l evels (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) W estern blot analysis of endogenous CDK10 and cyclin M expression levels in MCF7 cells, in response to siRNA-mediated gene silencing. (B) Wester n 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. S3 A. (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 th e proteasome (13). A straightforward hypothesis is that CDK10/cyclin M positively controls ETS2 degradation. We thus examined the impact of CDK 10 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 e xogenously expressed Flag-ETS2 protein (Fig. S4A), as well as of the endogenous ETS2 protein (Fig. 4A). This increase is not attributable to increa sed 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-ET S2 was readily detected when expressed alone or, to a lesser extent, when coexpressed with CDK10(kd). However, its expression level was dramati cally 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 coe xpression involves CDK10 kinase activity. Treatment of cells coexpressing Flag-ETS2, CDK10, and cyclin M with the proteasome inhibitor MG132 lar gely rescued Flag-ETS2 expression levels (Fig. 4B).Fig. 4.In a new window Download PPTFig. 4.CDK10/cyclin M controls ETS2 stability in human c ancer derived cells. (A) Western blot analysis of endogenous ETS2 expression levels in MCF7 cells, in response to siRNA-mediated CDK10 and/or c yclin M silencing. A quantification is shown in Fig. S4B. (B) Western blot analysis of exogenously expressed Flag-ETS2 protein levels in MCF7 cells c otransfected 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 bl ot analysis of expression levels of exogenously expressed Flag-ETS2 wild-type or mutant proteins in MCF7 cells, in the absence of (C) or in respons e to (D) Myc-CDK10/CycM-V5-6His expression. Quantifications are shown in Fig. S4 C and D.A mass spectrometry analysis of recombinant ETS2 ph osphorylated by CDK10/cyclin M in vitro revealed the existence of multiple phosphorylated residues, among which are two neighboring phospho-seri nes (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 ETS 2wt (Fig. S9), thus confirming that Ser220/Ser225 are phosphorylated by CDK10/cyclin M. To establish a direct link between ETS2 phosphorylation b y CDK10/cyclin M and degradation, we examined the expression levels of Flag-ETS2SASA. In the absence of CDK10/cyclin M coexpression, it did n

y CDK10/cyclin M and degradation, we examined the expression levels of Flag-ETS2SASA. In the absence of CDK10/cyclin M coexpression, it did n
ot 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 expr
ession 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 p
atient with STAR syndrome, bearing FAM58A mutation c.555+1G>A, predicted to result in aberrant splicing (10). In accordance with incomplete ske
wing of X chromosome inactivation previously found in this patient, we detected a decreased expression level of cyclin M protein in the STAR cell lin
e, compared with a control lymphoblastoid cell line. In line with our preceding observations, we detected an increased expression level of ETS2 prot
ein 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 sim
ilar 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 ET
S2 protein levels (Fig. 5C).Fig. 5.In a new window Download PPTFig. 5.Decreased cyclin M expression in STAR patient-derived cells results in incre
ased 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 co
ntrol 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 Se
ctionDiscussionIn this work, we unveil the interaction between CDK10, the last orphan CDK discovered in the pregenomic era (2), and cyclin M, the o
nly cyclin associated with a human genetic disease so far, and whose functions remain unknown (10). The closest paralogs of CDK10 within the CD
K family are the CDK11 proteins, which interact with L-type cyclins (15). Interestingly, the closest paralog of these cyclins within the cyclin family is cy
clin M (Fig. S11). The fact that none of the shorter CDK10 isoforms interact robustly with cyclin M suggests that alternative splicing of the CDK10 gen
e (16, 17) plays an important role in regulating CDK10 functions.The functional relevance of the interaction between CDK10 and cyclin M is supporte
d 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 d
egradation 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 t
o CDK10, independently from its kinase activity, as seen for cyclin C and CDK8 (19). We uncover a cyclin M-dependent CDK10 protein kinase activit
y 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-dep
endent kinase. Our Y2H assays reveal that truncated cyclin M proteins corresponding to the hypothetical products of two STAR syndrome-associate
d 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 devel
opmental 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 judge
d from the sensitivity of ETS2 levels to partially decreased CDK10 and cyclin M levels, achieved in MCF7 cells and observed in STAR cells, respectiv
ely. These findings offer a straightforward explanation for the already reported up-regulation of ETS2-driven transcription of c-RAF in response to CD
K10 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 β-TRCP phosphodegron (DSMCPAS) (14). Because none of these two serines precede a proline, they do not confo
rm 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 a
llow docking-mediated phosphorylation of atypical sites. The control of ETS2 degradation involves a number of players, including APC-Cdh1 (13) an
d 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 b
iological and medical implications. First, they shed light on the regulation of ETS2, which plays an important role in development (22) and is frequentl
y 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 horm
one therapy response of ERα-positive breast cancer patients. Third, our findings offer an interesting hypothesis on the molecular mechanisms underl
ying 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 ETS
2 expression levels would be a decreased risk to develop certain types of cancers and an increased risk to develop others. Studies on various mous
e 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 h
as 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-suppressiv
e activity in some cancers (7, 8). The severe growth retardation exhibited by STAR patients strongly suggests that CDK10/cyclin M plays an importan
t role in the control of cell proliferation.Previous SectionNext SectionMaterials and MethodsCloning of CDK10 and cyclin M cDNAs, plasmid constructi
ons, tamoxifen response analysis, quantitative RT-PCR, mass spectrometry experiments, and antibody production are detailed in SI Materials and M
ethods.Yeast Two-Hybrid Interaction Assays. We performed yeast interaction mating assays as previously described (27).Mammalian Cell Cultures a
nd Transfections. We grew human HEK293 and MCF7 cells in DMEM supplemented with 10% (vol/vol) FBS (Invitrogen), and we grew lymphoblastoi
d cells in RPMI 1640 GlutaMAX supplemented with 15% (vol/vol) FBS. We transfected HEK293 and MCF7 cells using Lipofectamine 2000 (Invitroge
n) for plasmids, Lipofectamine RNAiMAX (Invitrogen) for siRNAs, and Jetprime (Polyplus) for plasmids/siRNAs combinations according to the manuf
acturers' instructions. We transfected lymphoblastoid cells by electroporation (Neon, Invitrogen). For ETS2 stability studies we treated MCF7 cells 32
h after transfection with 10 μM MG132 (Fisher Scientific) for 16 h.Coimmunoprecipitation and Western Blot Experiments. We collected cells by scrapi
ng in PBS (or centrifugation for lymphoblastoid cells) and lysed them by sonication in a lysis buffer containing 60 mM β-glycerophosphate, 15 mM p-n
itrophenylphosphate, 25 mM 3-(N-morpholino)propanesulfonic acid (Mops) (pH 7.2), 15 mM EGTA, 15 mM MgCl2, 1 mM Na vanadate, 1 mM NaF, 1
mM 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 μg of total prot
eins, in lysis buffer. We precleared the lysates with 20 μL of protein A or G-agarose beads, incubated 1 h 4 °C on a rotating wheel. We added 5 μg o
f antibody to the supernatants, incubated 1 h 4 °C on a rotating wheel, added 20 μL of protein A or G-agarose beads, and incubated 1 h 4 °C on a r
otating 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 (p
H 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 a
ntibodies: 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 (Co
valab 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-acti
n (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, d
ilution 1:3,000) or anti-rabbit (Bio-Rad 172–1019, 1:5,000) as secondary antibodies. We revealed the blots by enhanced chemiluminescence (SuperS
ignal West Femto, Thermo Scientific).Production and Purification of Recombinant Proteins.GST-CDK10(kd)/StrepII-CycM. We generated recombinan
t bacmids in DH10Bac Escherichia coli and baculoviruses in Sf9 cells using the Bac-to-Bac system, as described by the provider (Invitrogen). We infe
cted 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 Na Cl, 0.5% Nonidet P-40, 50 mM NaF, 10 mM β-glycerophosphate, and 0.3 mM Na-vanadate) containing a protease inhibitor mixture (Roche). We lyse d 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. W e 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 µL kinase buffer A containing 10% (vol/vol) glycerol for storage.6His-ETS2. We transformed Origami2 DE3 (Novagen) with the 6His-ETS2 expression v ector. We induced expression with 0.2 mM isopropyl-β-d-1-thiogalactopyranoside for 3 h at 22 °C. To purify 6His-ETS2, we spun 50 mL cells and res uspended 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 m ixture 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 fr action and added it to 200 µL Cobalt beads (Thermo Scientific). After 1 h incubation at 4 °C on a rotating wheel, we washed four times with lysis buff er. 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 1 0,000 × 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 µM BSA, 15 mM DTT, 100 µM ATP, 5 µCi ATP[γ-32P], 7.75 µM histone H1, or 1 µM 6His-ETS2 a nd added kinase buffer A (25 mM Tris·HCl, 10 mM MgCl2, 1 mM EGTA, 1 mM DTT, and 3.7 µM heparin, pH 7.5) up to a total volume of 30 µL. We in cubated 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 slice s 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–daug hter pair1 (Table 1 and Supplementary Note online). These authors noted clinical overlap with Townes-Brocks syndrome but suggested that the phen otype represented a separate autosomal dominant entity (MIM601446). Here we define the cardinal features of this syndrome as a characteristic faci al 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 syndro me. Figure 1 : Clinical and molecular characterization of STAR syndrome. (a–f) Facial appearances of cases 1–3 (apparent telecanthus, dysplastic e ars 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 p arental consent has been obtained for publication of these photographs). Anal atresia and hypoplastic labia are not shown. (g,h) X-ray films of the fee t 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 delet ion 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 exon s 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 h orizontal bars above the boxes indicate the amplicons used for qPCR and sequencing (one alternative exon 5 amplicon is not indicated because of s pace 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 (9 7 KB) Table 1: Clinical features in STAR syndrome cases Table 1 - Clinical features in STAR syndrome cases Full table On the basis of the phenot ypic overlap with Townes-Brocks, Okihiro and Feingold syndromes, we analyzed SALL1 (ref. 2), SALL4 (ref. 3) and MYCN4 but found no mutations i n any of these genes (Supplementary Methods online). Next, we carried out genome-wide high-resolution oligonucleotide array comparative genomi c hybridization (CGH)5 analysis (Supplementary Methods) of genomic DNA from the most severely affected individual (case 1, with lower lid colobom a, 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 breakp oint 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 Supplem entary 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 o f 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 t he 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 no vo 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 re moved 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 mut ations and deletions by independent rounds of PCR and sequencing or by qPCR. We confirmed paternity and de novo status of the point mutations a nd deletions in all sporadic cases. None of the mutations were seen in the DNA of 60 unaffected female controls, and no larger deletions involving FA M58A 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 informat ive SNPs. Case 5, the mother of case 6, gave birth to two boys, both clinically unaffected (samples not available). We cannot exclude that the conditi on 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 e xons, 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 (N M_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 (NCB I expression viewer, UniGene Hs.496943). Expression was also noted in 24 of 26 listed tumor tissues as well as in embryo and fetus. Genes homolo gous to FAM58A (NCBI HomoloGene: 13362) are found on the X chromosome in the chimpanzee and the dog. The zebrafish has a similar gene on c hromosome 23. However, in the mouse and rat, there are no true homologs. These species have similar but intronless genes on chromosomes 11 (m ouse) and 10 (rat), most likely arising from a retrotransposon insertion event. On the murine X chromosome, the flanking genes Atp2b3 and Dusp9 ar e conserved, but only remnants of the FAM58A sequence can be detected. FAM58A contains a cyclin-box-fold domain, a protein-binding domain fou nd in cyclins with a role in cell cycle and transcription control. No human phenotype resulting from a cyclin gene mutation has yet been reported. Ho mozygous knockout mice for Ccnd1 (encoding cyclin D1) are viable but small and have reduced lifespan. They also have dystrophic changes of the r etina, 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 w ith FAM58A in a manner similar to that observed for SALL4 and cyclin D1. We found that FAM58A interacts with SALL1 but not with SALL4 (Supplem entary Fig. 5 online), supporting the hypothesis that FAM58A and SALL1 participate in the same developmental pathway. How do FAM58A mutation s lead to STAR syndrome? Growth retardation (all cases; Table 1) and retinal abnormalities (three cases) are reminiscent of the reduced body size a nd 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 (Suppleme ntary Methods and Supplementary Fig. 6 online), supporting the link between FAM58A and cell proliferation. We found that loss-of-function mutation s 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 del etion 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 ca using the overlapping phenotypes of STAR syndrome and Townes-Brocks syndrome seem to act in the same pathway. Of note, MYCN, a gene muta ted in Feingold syndrome, is a direct regulator of cyclin D2 (refs. 10,11); thus, it is worth exploring whether the phenotypic similarities between Feing

old and STAR syndrome might be explained by direct regulation of FAM58A by MYCN. FAM58A is located approximately 0.56 Mb centromeric to M
ECP2 on Xq28. Duplications overlapping both MECP2 and FAM58A have been described and are not associated with a clinical phenotype in females
12, but no deletions overlapping both MECP2 and FAM58A have been observed to date13. Although other genes between FAM58A and MECP2 hav
e been implicated in brain development, FAM58A and MECP2 are the only genes in this region known to result in X-linked dominant phenotypes; thu
s, deletion of both genes on the same allele might be lethal in both males and females."

In [8]:

```
item = ['fig', 'pptfig' ,'kb', 'mm'] # After some observation of data

def clean_sentence(text):
    text = text.lower() # converts text to lower case
    text = contractions.fix(text) # converts (don't) to (do not)
    text = re.sub('\W+',' ',text) # removes all special chars, punc
    text = ''.join([i for i in text if not i.isdigit()]) # removes numbers
    for i in item:
        text = text.replace(i, '')
    text = text.split()
    text1 = []
    for i in text:
        if "_" in i:
            del i # removes string if it contains '_'
        else:
            text1.append(i)
    text = " ".join(text1)
    text = ' '.join([i for i in text.split() if len(i)>1]) # https://stackoverflow.com/a/32705991/10219869
    return text
```

In [9]:

```
df_2['TEXT'] = [clean_sentence(i) for i in df_2['Text']]
```

In [10]:

```
df_2['TEXT'][0]
```

Out[10]:

'cyclin dependent kinases cdks regulate variety of fundamental cellular processes cdk stands out as one of the last orphan cdks for which no activatin
g cyclin has been identified and no kinase activity revealed previous work has shown that cdk silencing increases ets ets erythroblastosis virus oncog
ene homolog driven activation of the mapk pathway which confers tamoxifen resistance to breast cancer cells the precise mechanisms by which cdk
modulates ets activity and more generally the functions of cdk remain elusive here we demonstrate that cdk is cyclin dependent kinase by identifying
cyclin as an activating cyclin cyclin an orphan cyclin is the product of fama whose mutations because star syndrome human developmental anomaly
whose features include toe syndactyly telecanthus and anogenital and renal malformations we show that star syndrome associated cyclin mutants ar
e unable to interact with cdk cyclin silencing phenocopies cdk silencing in increasing raf and in conferring tamoxifen resistance to breast cancer cells
cdk cyclin phosphorylates ets in vitro and in cells it positively controls ets degradation by the proteasome ets protein levels are increased in cells deri
ved from star patient and this increase is attributable to decreased cyclin levels altogether our results reveal an additional regulatory mechanism for e
ts which plays key roles in cancer and development they also she would light on the molecular mechanisms underlying star syndrome cyclin depende
nt kinases cdks play pivotal role in the control of number of fundamental cellular processes the human genome contains genes encoding proteins tha
t 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 altho
ugh discovered almost ago cdk remains one of the two cdks without an identified cyclin partner this knowledge gap has largely impeded the explorati
on of its biological functions cdk can act as positive cell cycle regulator in some cells or as tumor suppressor in others cdk interacts with the ets ets e
rythroblastosis virus oncogene homolog transcription factor and inhibits its transcriptional activity through an unknown mechanism cdk knockdown de
represses ets which increases the expression of the raf protein kinase activates the mapk pathway and induces resistance of mcf cells to tamoxifen h
ere we deorphanize cdk by identifying cyclin the product of fama as binding partner mutations in this gene that predict absence or truncation of cyclin
are associated with star syndrome whose features include toe syndactyly telecanthus and anogenital and renal malformations in heterozygous female
s however both the functions of cyclin and the pathogenesis of star syndrome remain unknown we show that recombinant cdk cyclin heterodimer is a
n active protein kinase that phosphorylates ets in vitro cyclin silencing phenocopies cdk silencing in increasing raf and phospho erk expression levels
and in inducing tamoxifen resistance in estrogen receptor er breast cancer cells we show that cdk cyclin positively controls ets degradation by the pr
oteasome through the phosphorylation of two neighboring serines finally we detect an increased ets expression level in cells derived from star patien
t and we demonstrate that it is attributable to the decreased cyclin expression level observed in these cells previous sectionnext sectionresultsa yeas
t two hybrid yh screen unveiled an interaction signal between cdk and mouse protein whose terminal half presents strong sequence homology with th
e human fama gene product whose proposed name is cyclin we thus performed yh mating assays to determine whether human cdk interacts with hu
man cyclin the longest cdk isoform expressed as bait protein produced strong interaction phenotype with full length cyclin expressed as prey protein
but no detectable phenotype with cyclin cip and cdi kap which are known binding partners of other cdks cdk and cdk also produced yh signals with cy
clin albeit notably weaker than that observed with cdk an interaction phenotype was also observed between full length cyclin and cdk proteins expres
sed as bait and prey respectively sa we then tested different isoforms of cdk and cyclin originating from alternative gene splicing and two truncated cy
clin proteins corresponding to the hypothetical products of two mutated fama genes found in star syndrome patients none of these shorter isoforms p
roduced interaction phenotypes and and sa in new window download ppt cdk and cyclin form an interaction complex schematic representation of the
different protein isoforms analyzed by yh assays amino acid numbers are indicated black boxes indicate internal deletions the red box indicates differ
ing amino acid sequence compared with cdk yh assay between set of cdk proteins expressed as baits in fusion to the lexa dna binding domain and c
dk interacting proteins expressed as preys in fusion to the transcriptional activator peg and pjg are the empty bait and prey plasmids expressing lexa
and respectively lacz was used as reporter gene and blue yeast are indicative of yh interaction phenotype yh assay between the different cdk and cyc
lin isoforms the amino terminal region of ets known to interact with cdk was also assayed western blot analysis of myc cdk wt or kd and cycm his exp
ression levels in transfected hek cells western blot analysis of myc cdk wt or kd iunoprecipitates obtained using the anti myc antibody inputs correspo
nd to µg total lysates obtained from hek cells coexpressing myc cdk wt or kd and cycm his western blot analysis of iunoprecipitates obtained using th
e anti cdk antibody or control goat antibody from human breast cancer mcf cells input corresponds to µg mcf total cell lysates the lower band of the d
oublet observed on the upper panel comigrates with the exogenously expressed untagged cdk and thus corresponds to endogenous cdk the upper b

and of the doublet corresponds to nonspecific signal as demonstrated by it insensitivity to either overexpression of cdk as seen on the left lane or silencing of cdk sb another experiment with longer gel migration is shown in sd next we examined the ability of cdk and cyclin to interact when expressed in human cells and we tested wild type cdk wt and kinase dead kd mutant bearing da amino acid substitution that abolishes atp binding we expressed cyclin his and or myc cdk wt or kd in human embryonic kidney cell line hek the expression level of cyclin his was significantly increased upon coexpression with myc cdk wt or kd and to lesser extent that of myc cdk wt or kd was increased upon coexpression with cyclin his we then iunoprecipitated myc cdk proteins and detected the presence of cyclin in the cdk wt and kd iunoprecipitates only when these proteins were coexpressed pair wise we confirmed these observations by detecting the presence of myc cdk in cyclin his iunoprecipitates sb these experiments confirmed the lack of robust interaction between the cdk isoform and cyclin sc to detect the interaction between endogenous proteins we performed iunoprecipitations on nontransfected mcf cells derived from human breast cancer cdk and cyclin antibodies detected their cognate endogenous proteins by western blotting we readily detected cyclin in iunoprecipitates obtained with the cdk antibody but not with control antibody these results confirm the physical interaction between cdk and cyclin in human cells to unveil hypothesized cdk cyclin protein kinase activity we produced gst cdk and strepii cyclin fusion proteins in insect cells either individually or in combination we observed that gst cdk and strepii cyclin copurified thus confirming their interaction in yet another cellular model we then performed in vitro kinase assays with purified proteins using histone as generic substrate histone phosphorylation was detected only from lysates of cells coexpressing gst cdk and strepii cyclin no phosphorylation was detected when gst cdk or strepii cyclin were expressed alone or when strepii cyclin was coexpressed with gst cdk kd next we investigated whether ets which is known to interact with cdk is phosphorylation substrate of cdk cyclin we detected strong phosphorylation of ets by the gst cdk strepii cyclin purified heterodimer whereas no phosphorylation was detected using gst cdk alone or gst cdk kd strepii cyclin heterodimer in new window download ppt cdk is cyclin dependent protein kinase in vitro protein kinase assay on histone lysates from insect cells expressing different proteins were purified on glutathione sepharose matrix to capture gst cdk 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 whose added amounts were visualized by coomassie staining lower middle and bottom same as in using purified recombinant his ets as substrate cdk silencing has been shown to increase ets driven raf transcription and to activate the mapk pathway we investigated whether cyclin is also involved in this regulatory pathway to aim at highly specific silencing we used sirna pools mix of four different sirnas at low final concentration nm both cdk and cyclin sirna pools silenced the expression of their cognate targets and and and interestingly the cyclin sirna pool also caused marked decrease in cdk protein level and sb these results and those shown in suggest that cyclin binding stabilizes cdk cyclin silencing induced an increase in raf protein and mrna levels and and in phosphorylated erk and erk protein levels sb similarly to cdk silencing as expected from these effects cdk and cyclin silencing both decreased the sensitivity of er mcf cells to tamoxifen to similar extent the combined silencing of both genes did not result in higher resistance to the drug sc altogether these observations demonstrate functional interaction between cyclin and cdk which negatively controls ets in new window download ppt cyclin silencing up regulates raf expression western blot analysis of endogenous cdk and cyclin expression levels in mcf cells in response to sirna mediated gene silencing western blot analysis of endogenous raf expression levels in mcf cells in response to cdk or cyclin silencing quantification is shown in sa quantitative rt pcr analysis of cdk cyclin and raf mrna levels in response to cdk upper or cyclin lower silencing we then wished to explore the mechanism by which cdk cyclin controls ets ets is short lived protein degraded by the proteasome straightforward hypothesis is that cdk cyclin positively controls ets degradation we thus examined the impact of cdk or cyclin silencing on ets expression levels the silencing of cdk and that of cyclin caused an increase in the expression levels of an exogenously expressed flag ets protein sa as well as of the endogenous ets protein this increase is not attributable to increased ets mrna levels which marginally fluctuated in response to cdk or cyclin silencing sb we then examined the expression levels of the flag tagged ets protein when expressed alone or in combination with myc cdk or cdk kd with or without cyclin his flag ets was readily detected when expressed alone or to lesser extent when coexpressed with cdk kd however its expression level was dramatically decreased when coexpressed with cdk alone or with cdk and cyclin these observations suggest that endogenous cyclin levels are in excess compared with those of cdk in mcf cells and they show that the major decrease in ets levels observed upon cdk coexpression involves cdk kinase activity treatment of cells coexpressing flag ets cdk and cyclin with the proteasome inhibitor mg largely rescued flag ets expression levels in new window download ppt cdk cyclin controls ets stability in human cancer derived cells western blot analysis of endogenous ets expression levels in mcf cells in response to sirna mediated cdk and or cyclin silencing quantification is shown in sb western blot analysis of exogenously expressed flag ets protein levels in mcf cells cotransfected with empty vectors or coexpressing myc cdk wt or kd or myc cdk cycm his the latter cells were treated for with the mg proteasome inhibitor proper expression of cdk and cyclin tagged proteins was verified by western blot analysis and western blot analysis of expression levels of exogenously expressed flag ets wild type or mutant proteins in mcf cells in the absence of or in response to myc cdk cycm his expression quantifications are shown in and mass spectrometry analysis of recombinant ets phosphorylated by cdk cyclin in vitro revealed the existence of multiple phosphorylated residues among which are two neighboring phospho serines at positions and that may form phosphodegron to confirm this finding we compared the phosphorylation level of recombinant etswt with that of etssasa protein mutant bearing alanine substitutions of these two serines as expected from the existence of multiple phosphorylation sites we detected small but reproducible significant decrease of phosphorylation level of etssasa compared with etswt thus confirming that ser ser are phosphorylated by cdk cyclin to establish direct link between ets phosphorylation by cdk cyclin and degradation we examined the expression levels of flag etssasa in the absence of cdk cyclin coexpression it did not differ significantly from that of flag ets this is contrary to that of flag etsdbm bearing deletion of the terminal destruction box that was previously shown to be involved in apc cdh mediated degradation of ets however contrary to flag ets wild type the expression level of flag etssasa remained insensitive to cdk cyclin coexpression altogether these results suggest that cdk cyclin directly controls ets degradation through the phosphorylation of these two serines finally we studied lymphoblastoid cell line derived from patient with star syndrome bearing fama mutation predicted to result in aberrant splicing in accordance with incomplete skewing of chromosome inactivation previously found in this patient we detected decreased expression level of cyclin protein in the star cell line compared with control lymphoblastoid cell line in line with our preceding observations we detected an increased expression level of ets protein in the star cell line compared with the control and sa we then examined by quantitative rt pcr the mrna expression levels of the corresponding genes the star cell line showed decreased expression level of cyclin mrna but an expression level of ets mrna similar to that of the control cell line to demonstrate that the increase in ets protein expression is indeed result of the decreased cyclin expression observed in the star patient derived cell line we expressed cyclin his in this cell line this expression caused decrease in ets protein levels in new window download ppt decreased cyclin expression in star patient derived cells results in increased ets protein level western blot analysis of cyclin and ets protein levels in star patient derived lymphoblastoid cell line and in control lymphoblastoid cell line derived from healthy individual quantification is shown in sa quantitative rt pcr analysis of cyclin and ets mrna levels in the same cells western blot analysis of ets protein levels in the star patient derived lymphoblastoid cell line transfected with an empty vector or vector directing the expression of cyclin his another western blot revealing endogenously and exogenously expressed cyclin levels is shown in sb quantification of ets protein levels is shown in sc previous sectionnext sectiondiscussionin this work we unveil the interaction between cdk the last orphan cdk discovered in the pregenomic era and cyclin the only cyclin associated with human genetic disease so far and whose functions remain unknown the closest paralogs of cdk within the cdk family are the cdk proteins which interact with type cyclins interestingly the closest paralog of these cyclins within the cyclin family is cyclin the fact that none of the shorter cdk isoforms interact robustly with cyclin suggests that alternative splicing of the cdk gene plays an important role in regulating cdk functions the functional relevance of the interaction between cdk and cyclin is supported by different observations both proteins seem to enhance each other stability as judged from their increased expression levels when their partner is exogenously coexpressed and from the much reduced endogenous cdk expression level observed in response to cyclin silencing and sb cdk is subject to ubiquitin mediated degradation our observations suggest that cyclin protects cdk from such degradation and that it is the only cyclin partner of cdk at least in mcf cells they also suggest that cyclin stability is enhanced upon binding to cdk independently from its kinase activity as seen for cyclin and cdk we uncover cyclin dependent cdk protein kinase activity in vitro thus demonstrating that this protein which was named cdk on the sole basis of its amino acid sequence is indeed genuine cyclin dependent kinase our yh assays reveal that truncated cyclin proteins corresponding to the hypothetical products of two star syndrome associated fama mutations do not produce an interaction phenotype with cdk hence regardless of whether these mutated mrnas undergo nonsense mediated decay as suggested from the decreased cyclin mrna levels in star cells shown in or give rise to truncated cyclin proteins females affected by the star syndrome must exhibit compromised cdk cyclin kinase activity at least in some tissues and during specific developmental stages we show that ets known interactor of cdk is phosphorylation substrate of cdk cyclin in vitro and that cdk cyclin kinase activity positively controls ets degradation by the proteasome this control seems to be exerted through very fine mechanism as judged from the sensitivity of ets levels to partially decreased cdk and cyclin levels achieved in mcf cells and observed in star cells respectively these findings offer straightforward explanation for the already reported up regulation of ets driven transcription of raf in response to cdk silencing we bring evidence that cdk cyclin directly controls ets degradation through the phosphorylation of two neighboring serines which

may form noncanonical trcp phosphodegron dsmcpas because none of these two serines precede proline they do not conform to usual cdk phosphor
ylation sites however multiple so called transcriptional cdks cdk and to which cdk may belong have been shown to phosphorylate variety of motifs in
non proline directed fashion especially in the context of molecular docking with the substrate here it can be hypothesized that the high affinity interacti
on between cdk and the pointed domain of ets would allow docking mediated phosphorylation of atypical sites the control of ets degradation involves
number of players including apc cdh and the cullin ring ligase crl the formal identification of the ubiquitin ligase involved in the cdk cyclin pathway and
the elucidation of its concerted action with the other ubiquitin ligases to regulate ets degradation will require further studies our results present numbe
r of significant biological and medical implications first they she would light on the regulation of ets which plays an important role in development and i
s frequently deregulated in many cancers second our results contribute to the understanding of the molecular mechanisms causing tamoxifen resista
nce associated with reduced cdk expression levels and they suggest that like cdk cyclin could also be predictive clinical marker of hormone therapy r
esponse of erα positive breast cancer patients third our findings offer an interesting hypothesis on the molecular mechanisms underlying star syndro
me ets transgenic mice showing less than twofold overexpression of ets present severe cranial abnormalities and those observed in star patients cou
ld thus be caused at least in part by increased ets protein levels another expected consequence of enhanced ets expression levels would be decreas
ed risk to develop certain types of cancers and an increased risk to develop others studies on various mouse models including models of down syndr
ome in which three copies of ets exist have revealed that ets dosage can repress or promote tumor growth and hence that ets exerts noncell autono
mous functions in cancer intringuingly one of the very few star patients identified so far has been diagnosed with nephroblastoma finally our findings
will facilitate the general exploration of the biological functions of cdk and in particular its role in the control of cell division previous studies have sugg
ested either positive role in cell cycle control or tumor suppressive activity in some cancers the severe growth retardation exhibited by star patients st
rongly suggests that cdk cyclin plays an important role in the control of cell proliferation previous sectionnext sectionmaterials and methodscloning of
cdk and cyclin 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 m
aalian cell cultures and transfections we grew human hek and mcf cells in dmem supplemented with vol vol fbs invitrogen and we grew lymphoblastoi
d cells in rpmi glutamax supplemented with vol vol fbs we transfected hek and mcf cells using lipofectamine invitrogen for plasmids lipofectamine rnai
max invitrogen for sirnas and jetprime polyplus for plasmids sirnas combinations according to the manufacturers instructions we transfected lymphobl
astoid cells by electroporation neon invitrogen for ets stability studies we treated mcf cells after transfection with µm mg fisher scientific for coiunoprec
ipitation and western blot experiments we collected cells by scraping in pbs or centrifugation for lymphoblastoid cells and lysed them by sonication in
lysis buffer containing glycerophosphate nitrophenylphosphate morpholino propanesulfonic acid mops ph egta mgcl na vanadate naf phenylphosphat
e nonidet and protease inhibitor mixture roche we spun the lysates min at at collected the supernatants and determined the protein content using bra
dford assay we performed the iunoprecipitation experiments on µg of total proteins in lysis buffer we precleared the lysates with µl of protein or agaro
se beads incubated on rotating wheel we added µg of antibody to the supernatants incubated on rotating wheel added µl of protein or agarose beads
and incubated on rotating wheel we collected the beads by centrifugation at at and washed three times in bead buffer containing tris ph naf nacl edta
egta nonidet and protease inhibitor coktail roche we directly added sample buffer to the washed pellets heat denatured the proteins and ran the samp
les on 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 ab anti invitrogen anti tubulin santa cruz biotechnology anti cd
k covalab pabp or santa cruz biotechnology anti cycm home made dilution or covalab pab dilution anti raf santa cruz biotechnology anti ets santa cruz
biotechnology anti flag sigma and anti actin sigma we used hrp coupled anti goat santa cruz biotechnology sc dilution anti mouse bio rad dilution or ar
ti rabbit bio rad as secondary antibodies we revealed the blots by enhanced chemiluminescence supersignal west femto thermo scientific production
and purification of recombinant proteins gst cdk kd strepii cycm we generated recombinant bacmids in dhbac escherichia coli and baculoviruses in sf
cells using the bac to bac system as described by the provider invitrogen we infected sf cells with gst cdk or gst cdkkd producing viruses or coinfecte
d the cells with strepii cycm producing viruses and we collected the cells after infection to purify gst fusion proteins we spun ml cells and resuspende
d the pellet in ml lysis buffer pbs nacl nonidet naf glycerophosphate and na vanadate containing protease inhibitor mixture roche we lysed the cells b
y sonication spun the lysate min at collected the soluble fraction and added it to ml glutathione sepharose matrix we incubated at washed four times
with lysis buffer one time with kinase buffer see below and finally resuspended the beads in µl kinase buffer containing vol vol glycerol for storage his
ets we transformed origami de novagen with the his ets expression vector we induced expression with isopropyl thiogalactopyranoside for at to purif
y his ets we spun ml cells and resuspended the pellet in ml lysis buffer pbs nacl imidazole dtt and nonidet containing protease inhibitor mixture witho
ut edta roche we lysed the cells at bar using cell disruptor and spun the lysate min at we collected the soluble fraction and added it to µl cobalt beads
thermo scientific after incubation at on rotating wheel we washed four times with lysis buffer to elute we incubated beads min with elution buffer pbs i
midazole ph containing the protease inhibitor mixture spun at and collected the eluted protein protein kinase assays we mixed glutathione sepharose
beads harboring gst cdk wt or kd either monomeric or complexed with strepii cycm µm bsa dtt µm atp µci atp µm histone or µm his ets and added kin
ase buffer tris hcl mgcl egta dtt and µm heparin ph up to total volume of µl we incubated the reactions min at added laemli sample buffer heat denatur
ed the samples and ran bis tris sds page we cut gel slices to detect gst cdk and strepii cycm by western blotting we stained the gel slices containing
the substrate with coomassie 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 similar combination of features had been report
ed previously in mother daughter pair table and supplementary note online these authors noted clinical overlap with townes brocks syndrome but sug
gested that the phenotype represented separate autosomal dominant entity mim here we define the cardinal features of this syndrome as characterist
ic facial appearance with apparent telecanthus and broad tripartite nasal tip variable syndactyly of toes hypoplastic labia anal atresia and urogenital
malformations we also observed variety of other features table ure clinical and molecular characterization of star syndrome ure clinical and molecular
characterization of star syndrome facial appearances of cases apparent telecanthus dysplastic ears and thin upper lips and toe syndactyly or 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 ray films of the feet of case showing only four rays on the left and delta shaped th and th metatarsals on the right
compare to clinical picture in array cgh data log ratio represents copy number loss of six probes spanning between and with one probe positioned wit
hin fama the deletion does not remove parts of other functional genes schematic structure of fama and position of the mutations fama has five coding
exons boxes the cyclin domain green is encoded by exons the horizontal arrow indicates the deletion extending in case which includes exons and wh
ereas the horizontal line below exon indicates the deletion found in case which removes exon and some sequence the pink horizontal bars above the
boxes indicate the amplicons used for qpcr and sequencing one alternative exon amplicon is not indicated because of space constraints the mutation
dupt case results in an iediate stop codon and the and splice mutations in cases and are predicted to be deleterious because they alter the conserved
splice donor and acceptor site of intron respectively full size image table clinical features in star syndrome cases table clinical features in star syndro
me cases full table on the basis of the phenotypic overlap with townes brocks okihiro and feingold syndromes we analyzed sall ref sall ref and mycn b
ut found no mutations in any of these genes supplementary methods online next we carried out genome wide high resolution oligonucleotide array co
mparative genomic hybridization cgh analysis supplementary methods of genomic dna from the most severely affected individual case with lower lid c
oloboma epilepsy and syringomyelia and identified heterozygous deletion of on xq which removed exons and of fama using real time pcr we confirme
d the deletion in the child and excluded it in her unaffected parents supplementary online supplementary methods and supplementary table online thr
ough cgh with customized oligonucleotide array enriched in probes for xq followed by breakpoint cloning we defined the exact deletion size as bp del
chromosome ncbi build and supplementary online the deletion removes the coding regions of exons and as well as intron bp bp of intron and bp of se
quence including the utr and the entire krtp pseudogene ncbi gene id paternity was proven using routine methods we did not find deletions overlappin
g fama in the available copy number variation cnv databases subsequently we carried out qpcr analysis of the three other affected individuals cases
and and the mother daughter pair from the literature cases and in case we detected de novo heterozygous deletion of overlapping exon supplementa
y online using xq targeted array cgh and breakpoint cloning we identified deletion of bp del chromosome ncbi build and supplementary which remove
d bp of intron all of exon including the utr and bp of sequence we found heterozygous fama point mutations in the remaining cases supplementary su
pplementary methods and supplementary table in case we identified the mutation affecting the splice donor site of intron in case we identified the fra
meshift mutation dupt which iediately results in premature stop codon nxfsx in cases and we detected the mutation which alters the splice acceptor si
te of intron we validated the point mutations and deletions by independent rounds of pcr and sequencing or by qpcr we confirmed paternity and de no
vo status of the point mutations and deletions in all sporadic cases none of the mutations were seen in the dna of unaffected female controls and no l

arger deletions involving fama were found in unrelated array cgh investigations by analyzing chromosome inactivation supplementary methods and s
upplementary online we found complete skewing of inactivation in cases and and almost complete skewing in case suggesting that cells carrying the
mutation on the active chromosome have growth disadvantage during fetal development using rt pcr on rna from lymphoblastoid cells of case supple
mentary we did not find any aberrant splice products as additional evidence that the mutated allele is inactivated furthermore fama is subjected to ina
ctivation in cases and the parental origin of the deletions could not be determined as result of lack of informative snps case the mother of case gave b
irth to two boys both clinically unaffected samples not available we cannot exclude that the condition is lethal in males no fetal losses were reported fr
om any of the families the function of fama is unknown the gene consists of five coding exons and the bp coding region encodes protein of amino aci
ds genbank lists mrna length of bp for the reference sequence expression of the gene by est data was found in of adult tissues including kidney colo
n cervix and uterus but not heart ncbi expression viewer unigene hs expression was also noted in of listed tumor tissues as well as in embryo and fet
us genes homologous to fama ncbi homologene are found on the chromosome in the chimpanzee and the dog the zebrafish has similar gene on chro
mosome however in the mouse and rat there are no true homologs these species have similar but intronless genes on chromosomes mouse and rat
most likely arising from retrotransposon insertion event on the murine chromosome the flanking genes atpb and dusp are conserved but only remnant
s of the fama sequence can be detected fama contains cyclin box fold domain protein binding domain found in cyclins with role in cell cycle and trans
cription control no human phenotype resulting from cyclin gene mutation has yet been reported homozygous knockout mice for ccnd encoding cyclin
are viable but small and have reduced lifespan they also have dystrophic changes of the retina likely as result of decreased cell proliferation and deg
eneration of photoreceptor cells during embryogenesis cyclin colocalizes with sall in the nucleus and both proteins cooperatively mediate transcriptio
nal repression as the phenotype of our cases overlaps considerably with that of townes brocks syndrome caused by sall mutations we carried out co
iunoprecipitation to find out if sall or sall would interact with fama in manner similar to that observed for sall and cyclin we found that fama interacts wi
th sall but not with sall supplementary online supporting the hypothesis that fama and sall participate in the same developmental pathway how do fam
a mutations lead to star syndrome growth retardation all cases table and retinal abnormalities three cases are reminiscent of the reduced body size a
nd retinal anomalies in cyclin knockout mice therefore proliferation defect might be partly responsible for star syndrome to address this question we c
arried out knockdown of fama mrna followed by proliferation assay transfection of hek cells with three different fama specific rnai oligonucleotides res
ulted in significant reduction of both fama mrna expression and proliferation of transfected cells supplementary methods and supplementary online s
upporting the link between fama and cell proliferation we found that loss of function mutations of fama result in rather homogeneous clinical phenotyp
e the additional anomalies in case are likely to result from an effect of the deletion on expression of neighboring gene possibly atpb or dusp however
we cannot exclude that the homogeneous phenotype results from an ascertainment bias and that fama mutations including missense changes could
result in broader spectrum of malformations the genes causing the overlapping phenotypes of star syndrome and townes brocks syndrome seem to a
ct in the same pathway of note mycn gene mutated in feingold syndrome is direct regulator of cyclin refs thus it is worth exploring whether the phenot
ypic similarities between feingold and star syndrome might be explained by direct regulation of fama by mycn fama is located approximately mb centr
omeric to mecp on xq duplications overlapping both mecp and fama have been described and are not associated with clinical phenotype in females b
ut no deletions overlapping both mecp and fama have been observed to date although other genes between fama and mecp have been implicated in
brain development fama and mecp are the only genes in this region known to result in linked dominant phenotypes thus deletion of both genes on the
same allele might be lethal in both males and females'

In [11]:

```
a = df_2['Text'][0].split()
print('No. of words before preprocessing:', len(a))
```

No. of words before preprocessing: 6089

In [12]:

```
b = df_2['TEXT'][0].split()
print('No. of words before preprocessing:', len(b))
```

No. of words before preprocessing: 5541

In [13]:

```
# merging both the dataframes based on 'ID'
df = pd.merge(left = df_1, right = df_2, on = 'ID')

df = df.drop('Text', axis=1)
df.head()
```

Out[13]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	oncogenic mutations in the monomeric casitas l...

In [14]:

```
df[df.isnull()].count()
```

Out[14]:

ID 0

```
Gene        0
Variation   0
Class       0
TEXT        0
dtype: int64
```

In [15]:

```
df.shape
```

Out[15]:

```
(3316, 5)
```

In [16]:

```
df['Variation'] = df['Variation'].str.replace('*', '')
```

In [17]:

```
df['Variation'].head()
```

Out[17]:

```
0    Truncating Mutations
1                    W802
2                   Q249E
3                   N454D
4                   L399V
Name: Variation, dtype: object
```

3.1.4. Test, Train and Cross Validation Split

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

In [18]:

```
x = df
y = df['Class']

# split the data into test and train by maintaining same distribution of output varaible 'y_true' [stratify=y_true]
x1, x_test, y1, y_test = train_test_split(x, y, test_size = 0.2, stratify = y)
x_train, x_cv, y_train, y_cv = train_test_split(x1, y1, test_size = 0.2, stratify = y1)
```

In [19]:

```
print('Datapoints in train dataset', x_train.shape[0])
print('Datapoints in cv dataset', x_cv.shape[0])
print('Datapoints in test dataset', x_test.shape[0])
```

```
Datapoints in train dataset 2121
Datapoints in cv dataset 531
Datapoints in test dataset 664
```

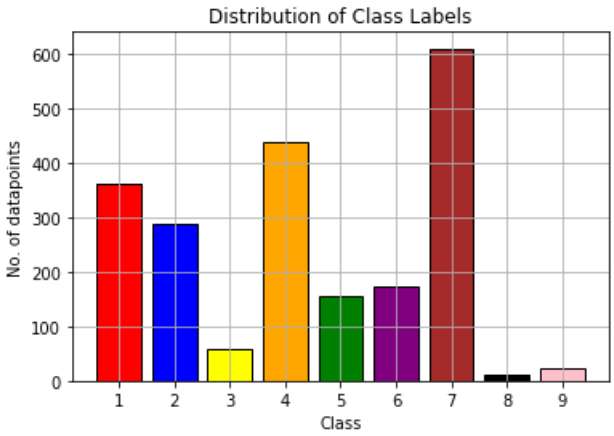
In [20]:

```
# We provide the class label 'y' and the function returns the number of individual class labels along with their percentages

def bar_plot(y): # y = y_train, y_cv, y_test
    yplot = Counter(y)
    key = yplot.keys()

    key = list(key)
    keys=[]
    for i in key:
        j = str(i)
        keys.append(j)

    value = yplot.values()
    value = list(value)
    y = str()

    # https://python-graph-gallery.com/3-control-color-of-barplots/
```

```
    plt.bar(x = np.arange(len(key)), height = value, edgecolor = 'black',
        color=['red', 'blue', 'yellow', 'orange', 'green', 'purple', 'brown', 'black', 'pink'])
    plt.xticks(np.arange(len(key)), keys)
    plt.xlabel('Class')
    plt.ylabel('No. of datapoints')
    plt.title('Distribution of Class Labels')
    plt.grid(b = True)
    plt.show()

    #https://stackoverflow.com/a/33216978/10219869
    ypts = OrderedDict(yplot.most_common())

    # https://stackoverflow.com/a/15785761/10219869
    for i, j in ypts.items():
        k = j/sum(ypts.values())
        print('The number of datapoints in Class {} : {} ({:.2f}%)'.format(i, j, k*100))
```

In [21]:

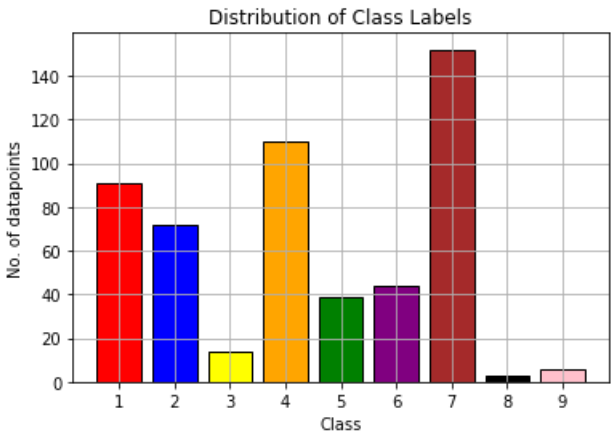
```
# Train
bar_plot(y_train)
```




```
The number of datapoints in Class 7 : 609 (28.71%)
The number of datapoints in Class 4 : 439 (20.70%)
The number of datapoints in Class 1 : 362 (17.07%)
The number of datapoints in Class 2 : 289 (13.63%)
The number of datapoints in Class 6 : 174 (8.20%)
The number of datapoints in Class 5 : 155 (7.31%)
The number of datapoints in Class 3 : 57 (2.69%)
The number of datapoints in Class 9 : 24 (1.13%)
The number of datapoints in Class 8 : 12 (0.57%)
```

In [22]:

```
# CV
bar_plot(y_cv)
```

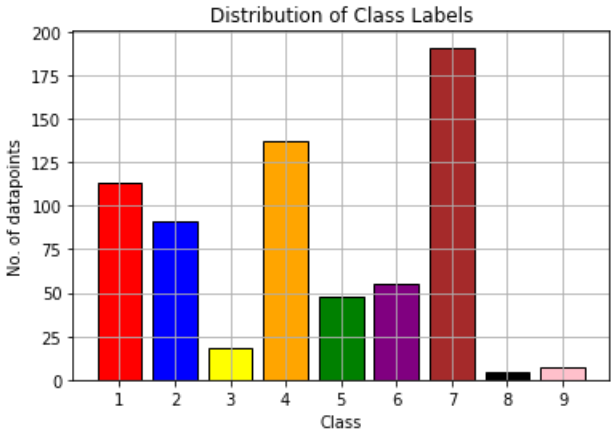



```
The number of datapoints in Class 7 : 152 (28.63%)
The number of datapoints in Class 4 : 110 (20.72%)
The number of datapoints in Class 1 : 91 (17.14%)
The number of datapoints in Class 2 : 72 (13.56%)
The number of datapoints in Class 6 : 44 (8.29%)
The number of datapoints in Class 5 : 39 (7.34%)
The number of datapoints in Class 3 : 14 (2.64%)
```

The number of datapoints in Class 3 : 14 (2.64%)
The number of datapoints in Class 9 : 6 (1.13%)
The number of datapoints in Class 8 : 3 (0.56%)

In [23]:

```
# Test
bar_plot(y_test)
```



The number of datapoints in Class 7 : 191 (28.77%)
The number of datapoints in Class 4 : 137 (20.63%)
The number of datapoints in Class 1 : 113 (17.02%)
The number of datapoints in Class 2 : 91 (13.70%)
The number of datapoints in Class 6 : 55 (8.28%)
The number of datapoints in Class 5 : 48 (7.23%)
The number of datapoints in Class 3 : 18 (2.71%)
The number of datapoints in Class 9 : 7 (1.05%)
The number of datapoints in Class 8 : 4 (0.60%)

3.2 Prediction using a 'Random' Model

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

In [24]:

```
# 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
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data

# CV
cv_pred = np.zeros((y_cv.shape[0], 9)) # initialization
for i in range(y_cv.shape[0]):        # no. of datapoints in cv
    rand_probs = np.random.rand(1,9)
    # random 'prediction' without using 'CLASSIFIER/ALGORITHM' and adding values to cv_pred
    cv_pred[i] = (rand_probs / sum(sum(rand_probs)))[0]
print('Log Loss on CV using Random Model:', log_loss(y_cv, cv_pred, eps = 1e-15))

# Test
test_pred = np.zeros((y_test.shape[0], 9)) # initialization
for i in range(y_test.shape[0]):          # no. of datapoints in test
    rand_probs = np.random.rand(1,9)
    # random 'prediction' without using 'CLASSIFIER/ALGORITHM' and adding values to cv_pred
    test_pred[i] = (rand_probs / sum(sum(rand_probs)))[0]
print('Log Loss on TEST using Random Model:', log_loss(y_test, test_pred, eps = 1e-15))
```

Log Loss on CV using Random Model: 2.5104090524768936
Log Loss on TEST using Random Model: 2.4735151649898794

In [25]:

```
pred_y = np.argmax(test_pred, axis=1)
c_m = confusion_matrix(y_test, pred_y)
c_m = c_m[1:,:-1]
```

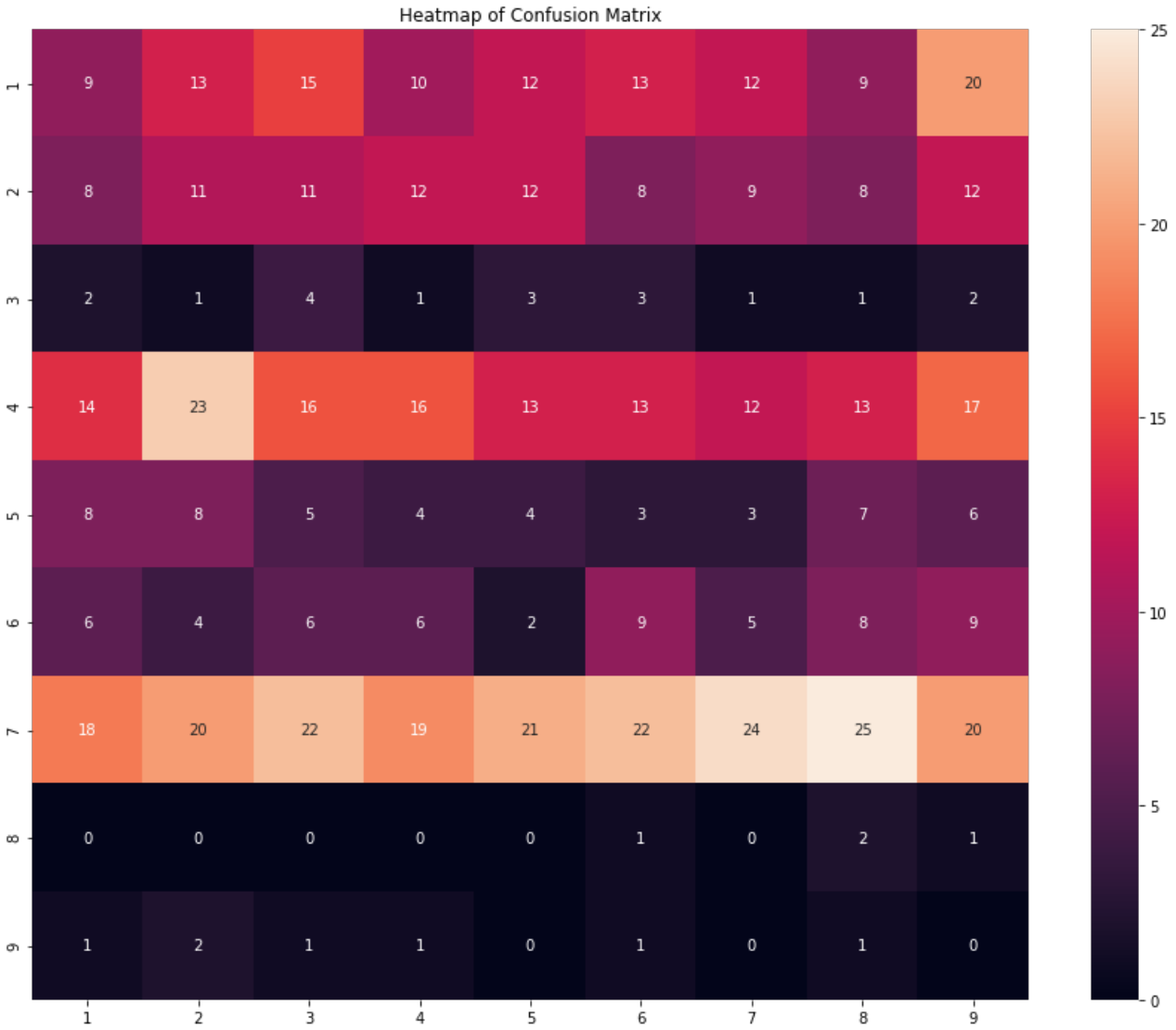
In [103]:

```
# We write a function for heatmap
```

```
def heatmap(cm):
    plt.figure(figsize = (15, 12))
    labels = [1,2,3,4,5,6,7,8,9]
    sns.heatmap(cm, annot = True, xticklabels = labels, yticklabels = labels)
    plt.show()
```

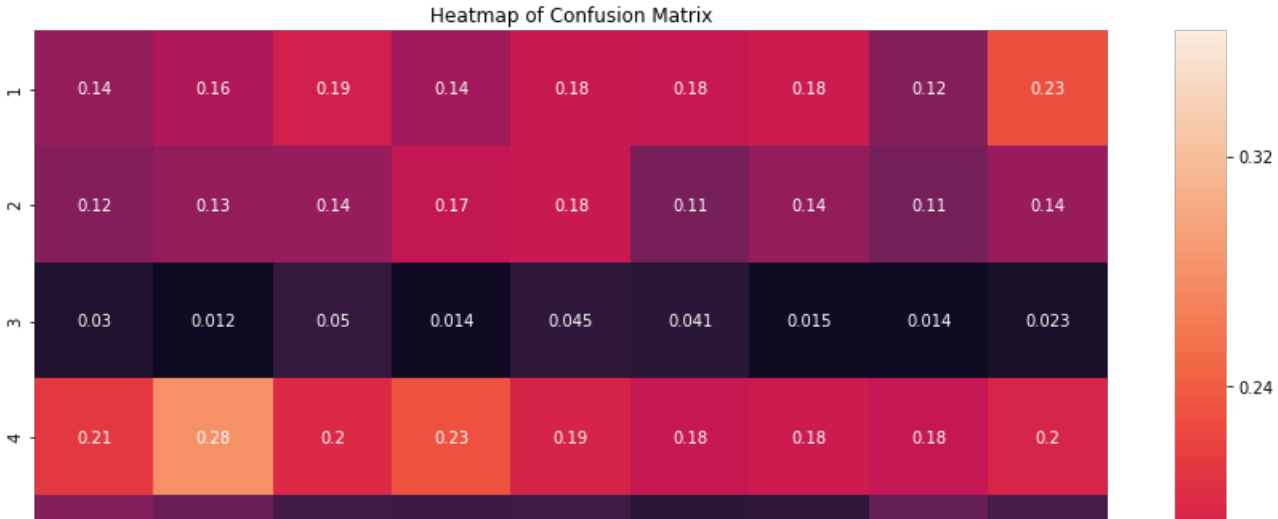
In [27]:

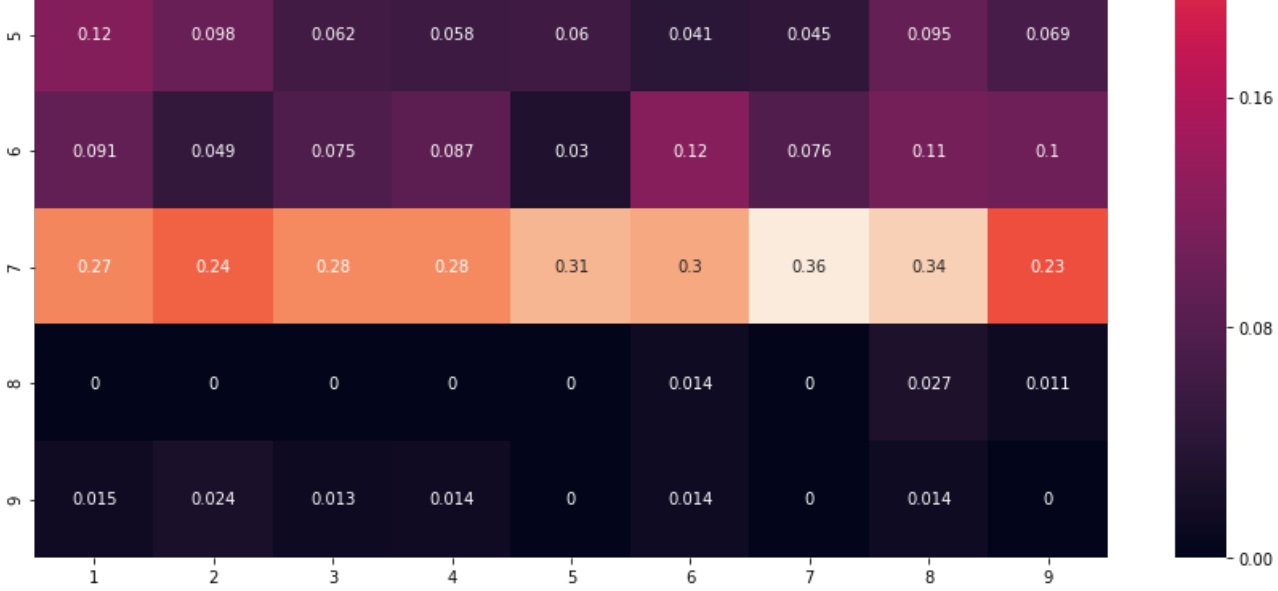
```
# Confusion Matrix
heatmap(c_m)
```



In [28]:

```
# Precision
precision = c_m / c_m.sum(axis=0)
heatmap(precision)
```





In [29]:

```
# Recall
recall = (c_m.T / c_m.sum(axis=0)).T
heatmap(recall)
```

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

In [30]:

```
x_train['Gene'].value_counts()
```

Out[30]:

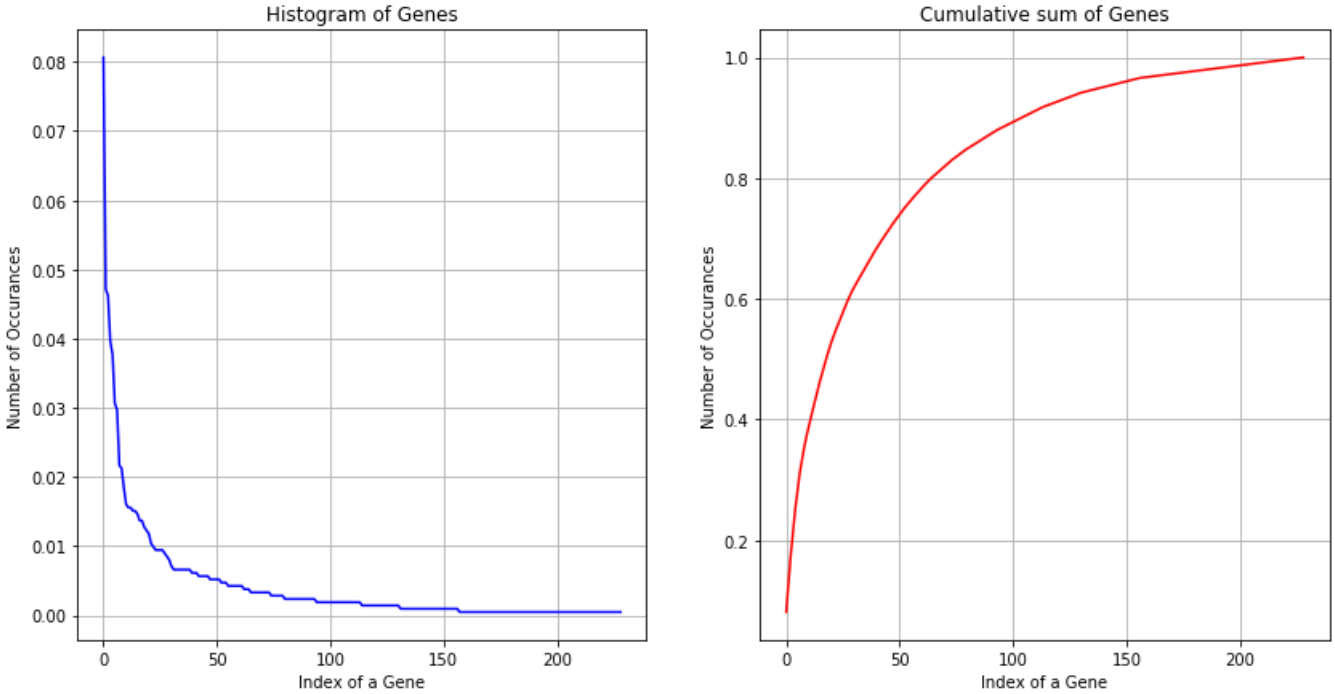
```
BRCA1     171
TP53      100
EGFR       98
PTEN       84
BRCA2      80
        ...
GNA11       1
PTCH1       1
INPP4B      1
MDM4        1
ACVR1       1
Name: Gene, Length: 229, dtype: int64
```

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

In [31]:

```
print("Ans: There are", len(x_train['Gene'].value_counts()) ,"different categories of genes in the train data,"
      "and they are distributed as follows:")
Genes = (x_train['Gene'].value_counts()).values / sum((x_train['Gene'].value_counts()).values)
cumsum_genes = np.cumsum(Genes)

# adding plots side by side
# https://towardsdatascience.com/subplots-in-matplotlib-a-guide-and-tool-for-planning-your-plots-7d63fa632857
fig = plt.figure(figsize = (14, 7))
fig.add_subplot(1, 2, 1) # 1 row 2 columns 1st plot
plt.plot(Genes, 'b')
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.title('Histogram of Genes')
plt.grid()

fig.add_subplot(1, 2, 2) # 1 row 2 columns 1st plot
plt.plot(cumsum_genes, 'r')
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.title('Cumulative sum of Genes')
plt.grid()
plt.show()
```

Ans: There are 229 different categories of genes in the train data,and they are distributed as follows:



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

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

In [32]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# 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)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional 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 = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.06818181818181818177, 0.13636363636363635, 0.25, 0.1931818181818181
8, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #     'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.0
66326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #     'EGFR': [0.05681818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181818177, 0.06818181818181818177, 0.0625, 0.3465909
0909090912, 0.0625, 0.05681818181818181816],
    #     '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, 0.075471698113207544, 0.
062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #     'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.06
6225165562913912, 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],
    #     ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = x_train[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
```

In [33]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
gene_train_rc = np.array(get_gv_feature(alpha, "Gene", x_train))
# cross validation gene feature
gene_cv_rc = np.array(get_gv_feature(alpha, "Gene", x_cv))
# test gene feature
gene_test_rc = np.array(get_gv_feature(alpha, "Gene", x_test))
```

In [34]:

```
print("The Gene feature is converted using respone coding method. The shape of gene feature:", gene_train_rc.shape)
```

The Gene feature is converted using respone coding method. The shape of gene feature: (2121, 9)

In [35]:

```
# OHE

gene_vect= CountVectorizer()

# Train
gene_train= gene_vect.fit_transform(x_train['Gene'])
# CV
gene_cv= gene_vect.transform(x_cv['Gene'])
# Test
gene_test= gene_vect.transform(x_test['Gene'])
```

In [36]:

```
print("Using CountVectorizer method, the shape of gene feature is converted to OHE:", gene_train.shape)
```

Using CountVectorizer method, the shape of gene feature is converted to OHE: (2121, 229)

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

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

In [37]:

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

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

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

cv_log_error= []
for i in tqdm(alpha):
    sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= i, random_state= 42)
    sgdc.fit(gene_train, y_train)
    cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
    cccv.fit(gene_train, y_train)
    y_pred= cccv.predict_proba(gene_cv)
    cv_log_error.append(log_loss(y_cv, y_pred, labels= sgdc.classes_, eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, y_pred, labels=sgdc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42)
sgdc.fit(gene_train, y_train)
cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
cccv.fit(gene_train, y_train)

y_pred= cccv.predict_proba(gene_train)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
      log_loss(y_train, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(gene_cv)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
      log_loss(y_cv, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(gene_test)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
      log_loss(y_test, y_pred, labels= sgdc.classes_, eps = 1e-15))
```

```
17%|▉         | 1/6 [00:00<00:01,  2.63it/s]

For values of alpha =  1e-05 The log loss is: 1.2316958191398661

33%|▉▉▉▉      | 2/6 [00:00<00:01,  2.93it/s]

For values of alpha =  0.0001 The log loss is: 1.2132069870497812

50%|▉▉▉▉▉▉    | 3/6 [00:00<00:00,  3.32it/s]

For values of alpha =  0.001 The log loss is: 1.256503935558334

67%|▉▉▉▉▉▉▉▉  | 4/6 [00:01<00:00,  3.55it/s]

For values of alpha =  0.01 The log loss is: 1.3637348422806232

100%|▉▉▉▉▉▉▉▉▉▉▉▉| 6/6 [00:01<00:00,  3.99it/s]
```

```
For values of alpha =  0.1 The log loss is: 1.4637013147986746
For values of alpha =  1 The log loss is: 1.5022278439856172
```


```
For values of best alpha:  0.0001 The train log loss is:  1.0065019048970862
For values of best alpha:  0.0001 The cv log loss is:  1.2132069870497812
For values of best alpha:  0.0001 The test log loss is:  1.1699099995665336
```

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

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

Q6. How many data points in Test and CV datasets are covered by the 230 genes in train dataset?

In [38]:

```
words = x_train['Gene'].values
words = set(words)

cv_words = x_cv['Gene'].isin(words)
test_words = x_test['Gene'].isin(words)

print('Datapoints in CV which are also in Train are in {:.2f}%'.format((len(x_cv['Gene'][cv_words]) / x_cv.shape[0])* 100))
print('Datapoints in Test which are also in Train are in {:.2f}%'.format((len(x_test['Gene'][test_words]) / x_test.shape[0])* 100))
```

```
Datapoints in CV which are also in Train are in 97.93%
Datapoints in Test which are also in Train are in 95.78%
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

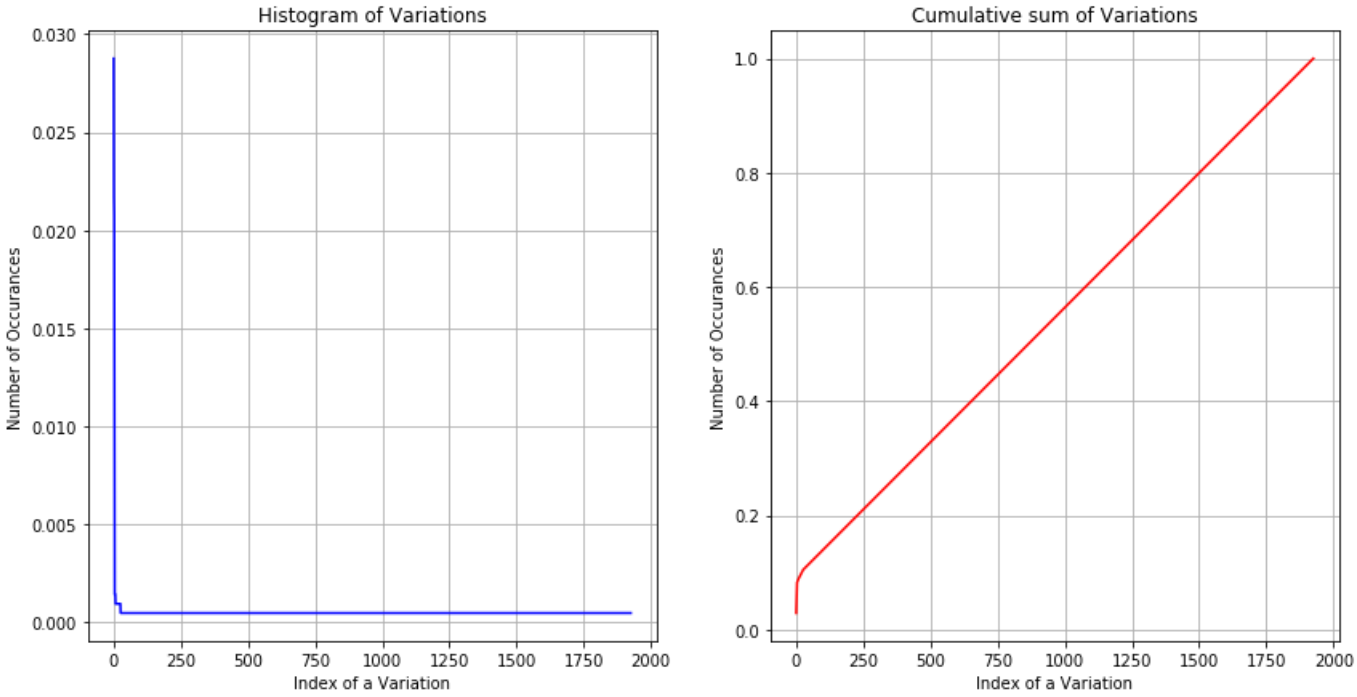
Q8. How many categories are there and how are they distributed?

In [39]:

```
print("Ans: There are", x_train['Variation'].value_counts().shape[0] ,"different categories of Variation in the train data,"
    "and they are distributed as follows:")
Variations = (x_train['Variation'].value_counts()).values / sum((x_train['Variation'].value_counts()).values)
cumsum_variations = np.cumsum(Variations)

# adding plots side by side
# https://towardsdatascience.com/subplots-in-matplotlib-a-guide-and-tool-for-planning-your-plots-7d63fa632857
fig = plt.figure(figsize = (14, 7))
fig.add_subplot(1, 2, 1) # 1 row 2 columns 1st plot
plt.plot(Variations, 'b')
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.title('Histogram of Variations')
```

```
plt.grid()

fig.add_subplot(1, 2, 2) # 1 row 2 columns 1st plot
plt.plot(cumsum_variations, 'r')
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.title('Cumulative sum of Variations')
plt.grid()
plt.show()
```

Ans: There are 1926 different categories of Variation in the train data,and they are distributed as follows:



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

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

In [40]:

```
#response-coding of the Variation feature
# alpha is used for laplace smoothing
alpha = 1

# train
variation_train_rc = np.array(get_gv_feature(alpha, "Variation", x_train))
# cv
variation_cv_rc = np.array(get_gv_feature(alpha, "Variation", x_cv))
# test
variation_test_rc = np.array(get_gv_feature(alpha, "Variation", x_test))
```

In [41]:

```
print("Variation feature is converted using respone coding method. The shape of Variation feature:", variation_train_rc.shape)
```

Variation feature is converted using respone coding method. The shape of Variation feature: (2121, 9)

In [42]:

```
# OHE
variation_vect= CountVectorizer()

# Train
variation_train= variation_vect.fit_transform(x_train['Variation'])
# CV
variation_cv= variation_vect.transform(x_cv['Variation'])
```

```
variation_cv= variation_vect.transform(x_cv['Variation'])
# Test
variation_test= variation_vect.transform(x_test['Variation'])
```

In [43]:

```
print("Using CountVectorizer method, the shape of Variation feature is converted to OHE:", variation_train.shape)
```

Using CountVectorizer method, the shape of Variation feature is converted to OHE: (2121, 1957)

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

Let's build a model just like the earlier!

In [44]:

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

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

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

cv_log_error= []
for i in tqdm(alpha):
    sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= i, random_state= 42)
    sgdc.fit(variation_train, y_train)
    cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
    cccv.fit(variation_train, y_train)
    y_pred= cccv.predict_proba(variation_cv)
    cv_log_error.append(log_loss(y_cv, y_pred, labels= sgdc.classes_, eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, y_pred, labels=sgdc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42)
sgdc.fit(variation_train, y_train)
cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
cccv.fit(variation_train, y_train)

y_pred= cccv.predict_proba(variation_train)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
      log_loss(y_train, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(variation_cv)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
      log_loss(y_cv, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(variation_test)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
      log_loss(y_test, y_pred, labels= sgdc.classes_, eps = 1e-15))
```

```
17%|▉         | 1/6 [00:00<00:01,  4.46it/s]
```

```
For values of alpha =  1e-05 The log loss is: 1.682001368508244
```

```
33%|▉▉▉       | 2/6 [00:00<00:00,  4.51it/s]
```

```
For values of alpha =  0.0001 The log loss is: 1.6695088440775319
```

```
50%|▉▉▉▉▉     | 3/6 [00:00<00:00,  4.53it/s]
```

```
For values of alpha =  0.001 The log loss is: 1.670808174821951
```

```
 67%|██████     | 4/6 [00:00<00:00,  4.30it/s]
```

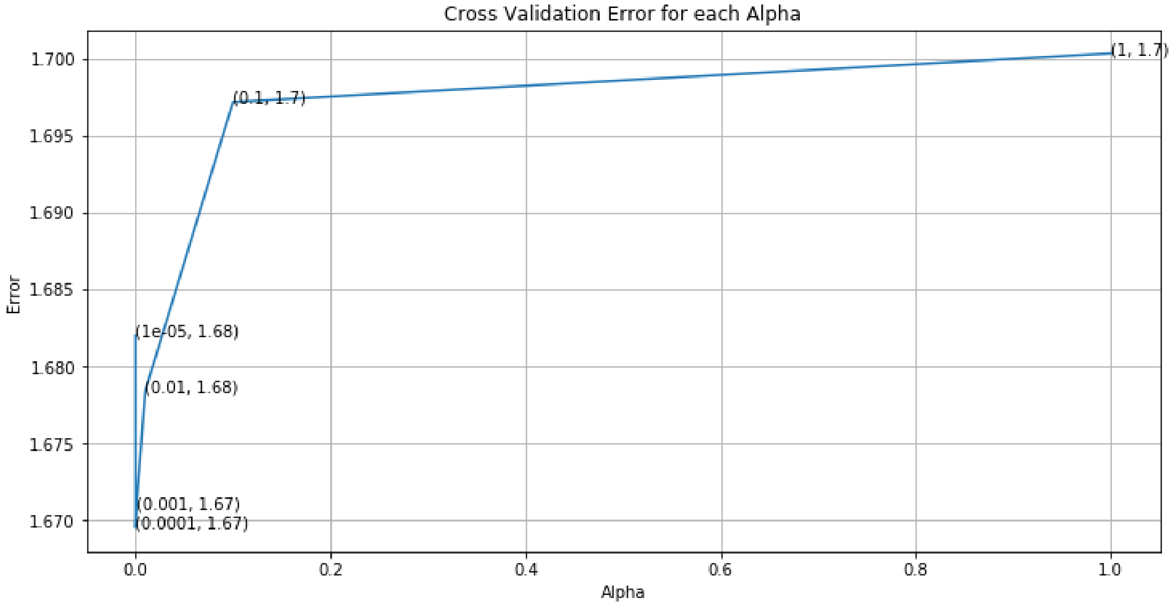
```
For values of alpha =  0.01 The log loss is: 1.6783932196829212
```

```
 83%|████████  | 5/6 [00:01<00:00,  4.19it/s]
```

```
For values of alpha =  0.1 The log loss is: 1.69716694217203
```

```
100%|██████████| 6/6 [00:01<00:00,  4.33it/s]
```

```
For values of alpha =  1 The log loss is: 1.700319587823367
```



```
For values of best alpha:  0.0001 The train log loss is:  0.776129115044374
For values of best alpha:  0.0001 The cv log loss is:  1.6695088440775319
For values of best alpha:  0.0001 The test log loss is:  1.692506405376072
```

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

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

In [45]:

```python
words = x_train['Variation'].values
words = set(words)

cv_words = x_cv['Variation'].isin(words)
test_words = x_test['Variation'].isin(words)
```

Q12. How many data points are covered by total?

In [46]:

```python
print('Ans:')
print('Datapoints in CV which are also in Train are in {:.2f}%'.format((len(x_cv['Variation'][cv_words]) / x_cv.shape[0])* 100))
print('Datapoints in Test which are also in Train are in {:.2f}%'.format((len(x_test['Variation'][test_words]) / x_test.shape[0])* 100))
```

```
Ans:
Datapoints in CV which are also in Train are in 10.17%
Datapoints in Test which are also in Train are in 10.24%
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?

2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

In [47]:

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

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

In [48]:

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

In [70]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in train data
text_vect= TfidfVectorizer(min_df= 3, ngram_range= (1, 2), max_features = 5000)
text_train = text_vect.fit_transform(x_train['TEXT'])

# getting all the feature names (words)
text_train_vocab= text_vect.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_train_feat_dict = dict(zip(list(text_train_vocab), text_train_feat_counts))

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

```
Total number of unique words in train data :  5000
```

In [50]:

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['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(x_train)


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

In [51]:

```
#response coding of text features
text_train_rc= get_text_responsecoding(x_train)
text_cv_rc= get_text_responsecoding(x_cv)
text_test_rc= get_text_responsecoding(x_test)
```

In [52]:

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

In [71]:

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

# CV
text_cv= text_vect.transform(x_cv['TEXT'])
# don't forget to normalize every feature
text_cv= normalize(text_cv, axis=0)

# Test
text_test= text_vect.transform(x_test['TEXT'])
# don't forget to normalize every feature
text_test= normalize(text_test, axis=0)
```

In [72]:

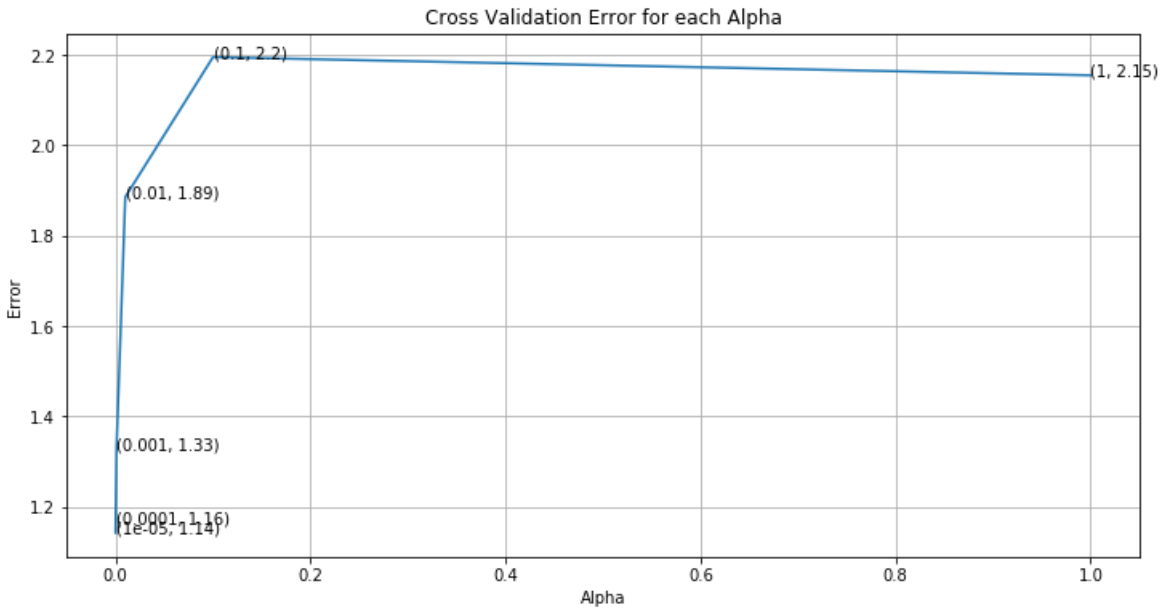
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text= dict(sorted(text_train_feat_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text.values()))
```

In [74]:

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

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

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

cv_log_error= []
for i in tqdm(alpha):
    sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= i, random_state= 42)
    sgdc.fit(text_train, y_train)
    cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
    cccv.fit(text_train, y_train)
    y_pred= cccv.predict_proba(text_cv)
    cv_log_error.append(log_loss(y_cv, y_pred, labels= sgdc.classes_, eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, y_pred, labels=sgdc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42)
sgdc.fit(text_train, y_train)
cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
```

```
cccv.fit(text_train, y_train)

y_pred= cccv.predict_proba(text_train)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
    log_loss(y_train, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(text_cv)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
    log_loss(y_cv, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(text_test)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
    log_loss(y_test, y_pred, labels= sgdc.classes_, eps = 1e-15))
```

```
17%|▉         | 1/6 [00:11<00:56, 11.32s/it]

For values of alpha =  1e-05 The log loss is: 1.1418491288157646

33%|▉▉▉▉      | 2/6 [00:17<00:38,  9.70s/it]

For values of alpha =  0.0001 The log loss is: 1.161822515342003

50%|▉▉▉▉▉▉    | 3/6 [00:21<00:23,  7.97s/it]

For values of alpha =  0.001 The log loss is: 1.3278244445387088

67%|▉▉▉▉▉▉▉▉  | 4/6 [00:24<00:13,  6.55s/it]

For values of alpha =  0.01 The log loss is: 1.8853271636467424

83%|▉▉▉▉▉▉▉▉▉▉ | 5/6 [00:27<00:05,  5.59s/it]

For values of alpha =  0.1 The log loss is: 2.195107286992929

100%|▉▉▉▉▉▉▉▉▉▉▉▉| 6/6 [00:30<00:00,  5.07s/it]

For values of alpha =  1 The log loss is: 2.154836727318279
```




```
For values of best alpha:  1e-05 The train log loss is:  0.6762806418765435
For values of best alpha:  1e-05 The cv log loss is:  1.1418491288157646
For values of best alpha:  1e-05 The test log loss is:  1.0420133908817764
```

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

Ans. Yes, it seems like!

In [57]:

```
def get_intersec_text(df):
```

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df= 3, ngram_range= (1, 2))
    df_text = df_text_vec.fit_transform(df['TEXT'])
    df_text_vocab = df_text_vec.get_feature_names()

    df_text_counts = df_text.sum(axis=0).A1
    df_text_dict = dict(zip(list(df_text_vocab), df_text_counts))
    len1 = len(set(df_text_vocab))
    len2 = len(set(text_train_vocab) & set(df_text_vocab))
    return len1,len2
```

In [58]:

```
# CV
len1,len2 = get_intersec_text(x_cv)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")

# Test
len1,len2 = get_intersec_text(x_test)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
```

```
96.462 % of word of Cross Validation appeared in train data
94.677 % of word of test data appeared in train data
```

4. Machine Learning Models

In [105]:

```
#Data preparation for ML models.

#Misc. functionns for ML models

def predict_and_plot_confusion_matrix(x_train_ohe, ytrain, x_test_ohe, ytest, clf):
    clf.fit(x_train_ohe, ytrain)
    cccv = CalibratedClassifierCV(clf, method="sigmoid")
    cccv.fit(x_train_ohe, ytrain)
    y_pred = cccv.predict(x_test_ohe)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(ytest, cccv.predict_proba(x_test_ohe)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((y_pred - ytest))/ytest.shape[0])
    print()
    print('*'*35 +' Confusion Matrix ' + '*'*35)
    c_m = confusion_matrix(ytest, y_pred)
    heatmap(c_m)
    # Precision
    print('*'*35 +' Precision Matrix ' + '*'*35)
    precision = c_m / c_m.sum(axis=0)
    heatmap(precision)
    # Recall
    print('*'*35 +' Recall Matrix ' + '*'*35)
    recall = (c_m.T / c_m.sum(axis=0)).T
    heatmap(recall)
```

In [60]:

```
def report_log_loss(x_train, y_train, x_test, y_test, clf):
    clf.fit(x_train, y_train)
    cccv = CalibratedClassifierCV(clf, method="sigmoid")
    cccv.fit(x_train, y_train)
    cccv_probs = cccv.predict_proba(x_test)
    return log_loss(y_test, cccv_probs, eps=1e-15) # log-loss between predicted prob label and test label
```

In [61]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3, ngram_range= (1, 2))

    gene_vec = gene_count_vec.fit(x_train['Gene'])
    var_vec  = var_count_vec.fit(x_train['Variation'])
```

```
text_vec = text_count_vec.fit(x_train['TEXT'])

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

word_present = 0
for i,v in enumerate(indices):
    if (v < fea1_len):
        word = gene_vec.get_feature_names()[v]
        yes_no = True if word == gene else False
        if yes_no:
            word_present += 1
            print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
    elif (v < fea1_len+fea2_len):
        word = var_vec.get_feature_names()[v-(fea1_len)]
        yes_no = True if word == var else False
        if yes_no:
            word_present += 1
            print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_no))
    else:
        word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
        yes_no = True if word in text.split() else False
        if yes_no:
            word_present += 1
            print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

print("Out of the top ",no_features," features ", word_present, "are present in query point")
```

Stacking the three types of features

In [75]:

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

# Y- Label
ytrain = np.array(y_train)
ycv = np.array(y_cv)
ytest = np.array(y_test)

# OHE
x_train_ohe = hstack((gene_train, variation_train, text_train)).tocsr()
x_cv_ohe = hstack((gene_cv, variation_cv, text_cv)).tocsr()
x_test_ohe = hstack((gene_test, variation_test, text_test)).tocsr()

# Response coding
x_train_rc = np.hstack((gene_train_rc, variation_train_rc, text_train_rc))
x_cv_rc = np.hstack((gene_cv_rc, variation_cv_rc, text_cv_rc))
x_test_rc = np.hstack((gene_test_rc, variation_test_rc, text_test_rc))
```

In [77]:

```
print("One hot encoding features & Tfidf:")
print("(number of data points * number of features) in Train data = ", x_train_ohe.shape)
print("(number of data points * number of features) in CV data = ", x_cv_ohe.shape)
print("(number of data points * number of features) in Test data =", x_test_ohe.shape)
```

```
One hot encoding features & Tfidf:
(number of data points * number of features) in Train data =  (2121, 7186)
(number of data points * number of features) in CV data =  (531, 7186)
(number of data points * number of features) in Test data = (664, 7186)
```

In [64]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", x_train_rc.shape)
print("(number of data points * number of features) in test data = ", x_cv_rc.shape)
print("(number of data points * number of features) in cross validation data =", x_test_rc.shape)
```

```
Response encoding features :
(number of data points * number of features) in train data =  (2121, 27)
(number of data points * number of features) in test data =  (531, 27)
(number of data points * number of features) in cross validation data = (664, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [78]:

```
alpha = [10 ** x for x in range(-5, 4)]

# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification


cv_log_error= []
for i in tqdm(alpha):
    mnb= MultinomialNB(alpha= i)
    mnb.fit(x_train_ohe, ytrain)
    cccv= CalibratedClassifierCV(base_estimator= mnb, method= 'sigmoid')
    cccv.fit(x_train_ohe, ytrain)
    y_pred= cccv.predict_proba(x_cv_ohe)
    cv_log_error.append(log_loss(ycv, y_pred, labels= mnb.classes_, eps = 1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, y_pred, labels=mnb.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

mnb= MultinomialNB(alpha= alpha[best_alpha])
mnb.fit(x_train_ohe, ytrain)
cccv= CalibratedClassifierCV(base_estimator= mnb, method= 'sigmoid')
cccv.fit(x_train_ohe, ytrain)

y_pred= cccv.predict_proba(x_train_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
      log_loss(ytrain, y_pred, labels= mnb.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
      log_loss(ycv, y_pred, labels= mnb.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
      log_loss(ytest, y_pred, labels= mnb.classes_, eps = 1e-15))
```

```
11%|▉         | 1/9 [00:00<00:02,  2.78it/s]
```

```
For values of alpha =  1e-05 The log loss is: 1.3047805919441118
```

```
22%|▉▉▉       | 2/9 [00:00<00:02,  2.84it/s]
```

```
For values of alpha =  0.0001 The log loss is: 1.3077504640656623
```

```
33%|▉▉▉▉      | 3/9 [00:01<00:02,  2.86it/s]
```

```
For values of alpha =  0.001 The log loss is: 1.3113847659614661
```

```
44%|▉▉▉▉▉▉    | 4/9 [00:01<00:01,  2.88it/s]
```

```
For values of alpha =  0.01 The log loss is: 1.3214269663648144
```

```
56%|▉▉▉▉▉▉▉   | 5/9 [00:01<00:01,  2.89it/s]
```

```
For values of alpha =  0.1 The log loss is: 1.3441880895425748
```

```
67%|▉▉▉▉▉▉▉▉  | 6/9 [00:02<00:01,  2.74it/s]
```

```
For values of alpha =  1 The log loss is: 1.3708175852898348
```

```
78%|▉▉▉▉▉▉▉▉▉ | 7/9 [00:02<00:00,  2.68it/s]
```

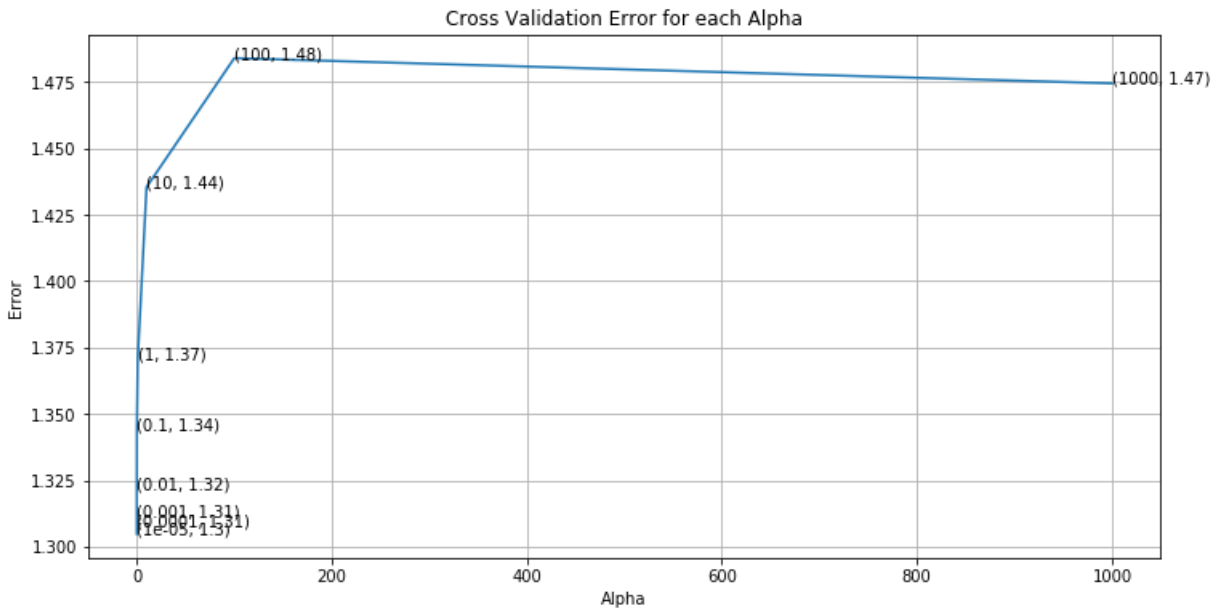
```
For values of alpha =  10 The log loss is: 1.4353242825886094
```

```
89%|▉▉▉▉▉▉▉▉▉▉| 8/9 [00:02<00:00,  2.65it/s]
```

```
For values of alpha =  100 The log loss is: 1.4839085861316486
```

```
100%|▉▉▉▉▉▉▉▉▉▉▉| 9/9 [00:03<00:00,  2.71it/s]
```

```
For values of alpha =  1000 The log loss is: 1.474392350284852
```




```
For values of best alpha:  1e-05 The train log loss is:  0.6622901424638779
For values of best alpha:  1e-05 The cv log loss is:  1.3047805919441118
For values of best alpha:  1e-05 The test log loss is:  1.2154567029971501
```

Observations:

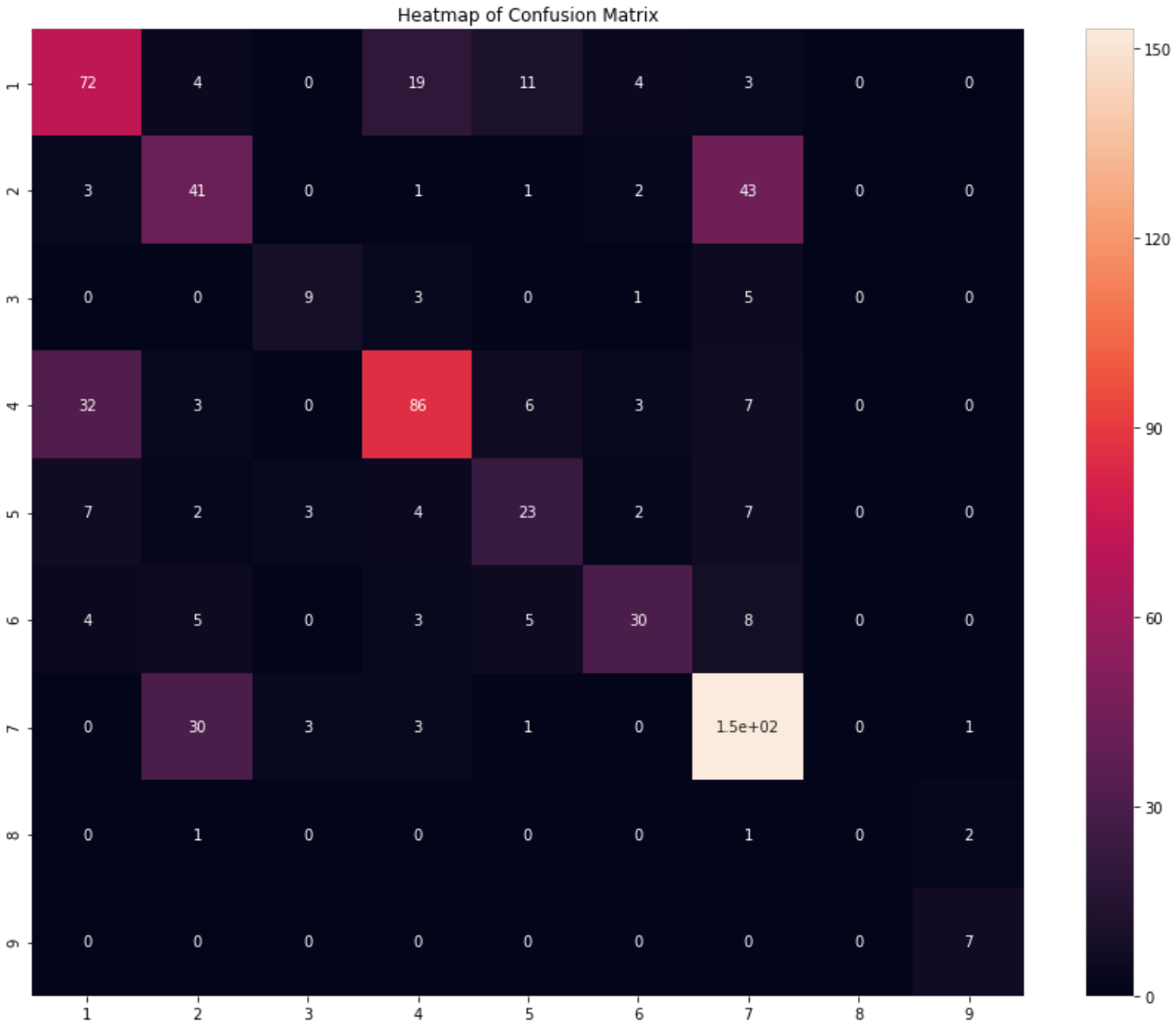
- This is Imbalanced dataset
- As I have considered bigrams in text vectorizer, the features (Top 5000):- 7186, and the log loss:- 1.21

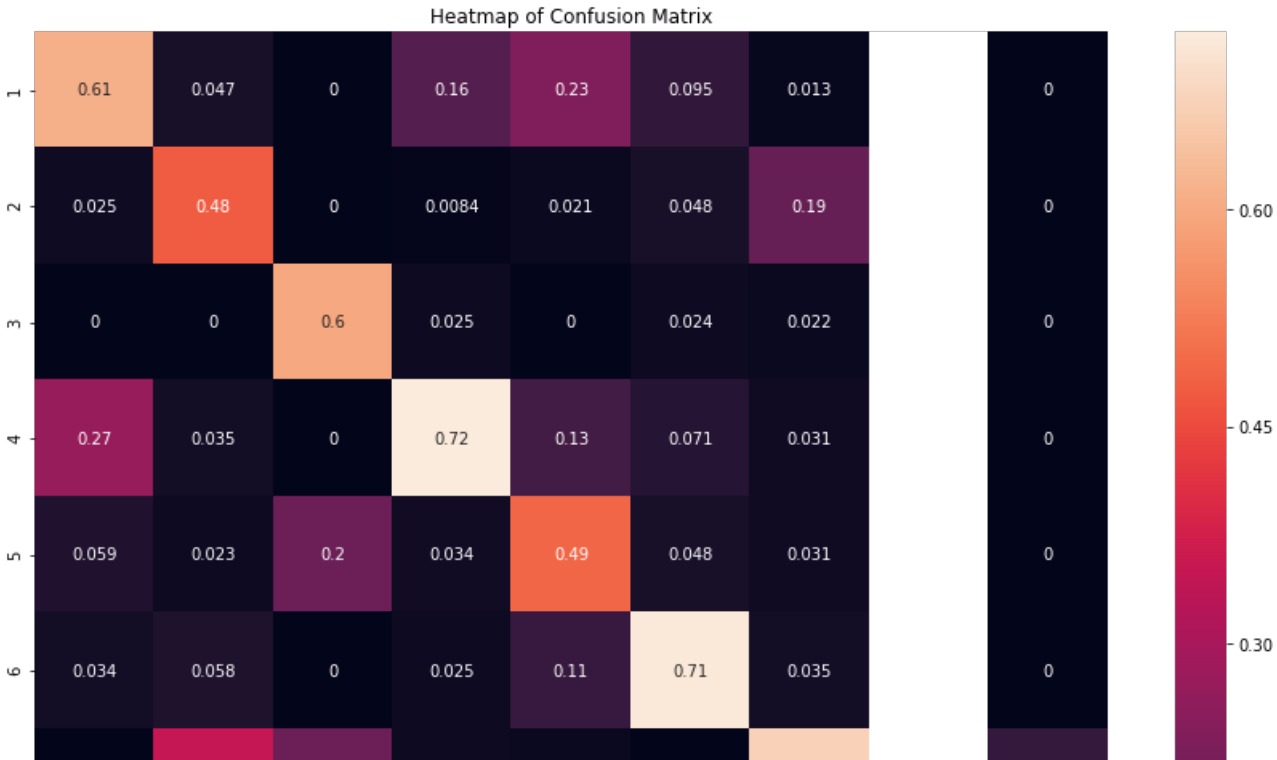
4.1.1.2. Testing the model with best hyper paramters

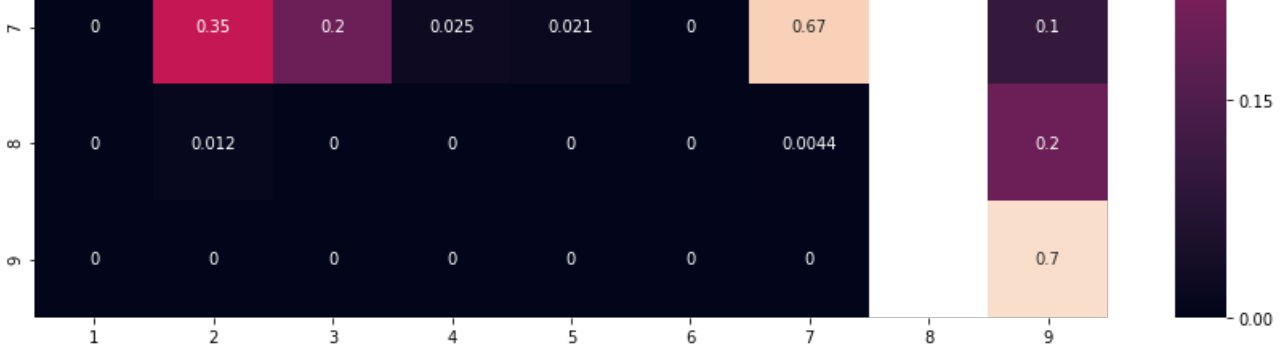
In [80]:

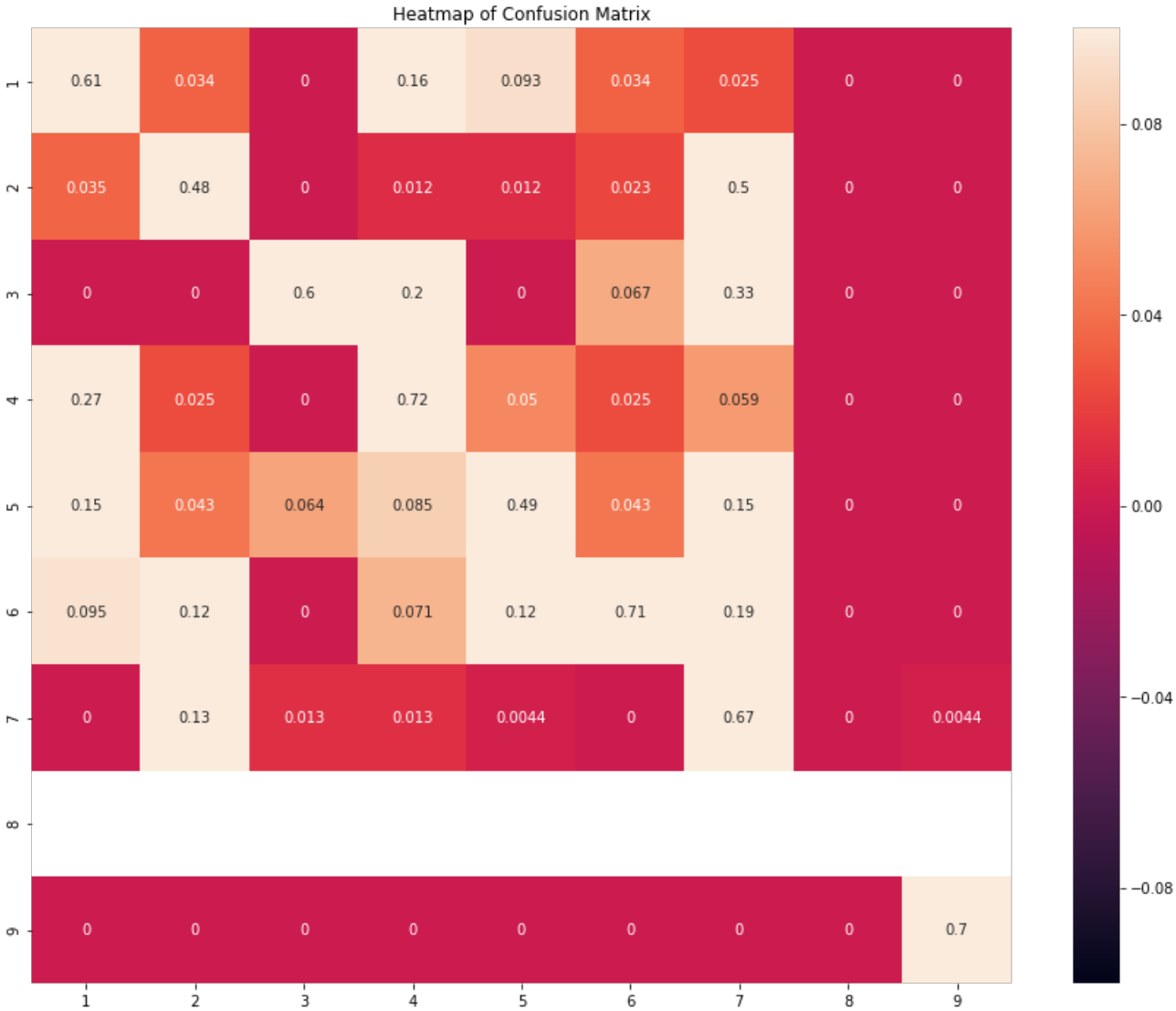
```
predict_and_plot_confusion_matrix(x_train_ohe, ytrain, x_test_ohe, ytest, MultinomialNB(alpha = alpha[best_alpha]))
```

Log loss : 1.2154567029971501
Number of mis-classified points : 0.36596385542168675









4.1.1.3. Feature Importance, Correctly classified point

In [84]:

```
test_point_index = 2
no_feature = 100
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]), 4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-mnb.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0578 0.0568 0.015  0.0729 0.0388 0.6604 0.0914 0.0045 0.0024]]
Actual Class : 6
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [88]:

```
test_point_index = 90
no_feature = 100
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-mnb.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0681 0.0675 0.0254 0.0868 0.0459 0.039  0.6591 0.0053 0.0029]]
Actual Class : 3
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

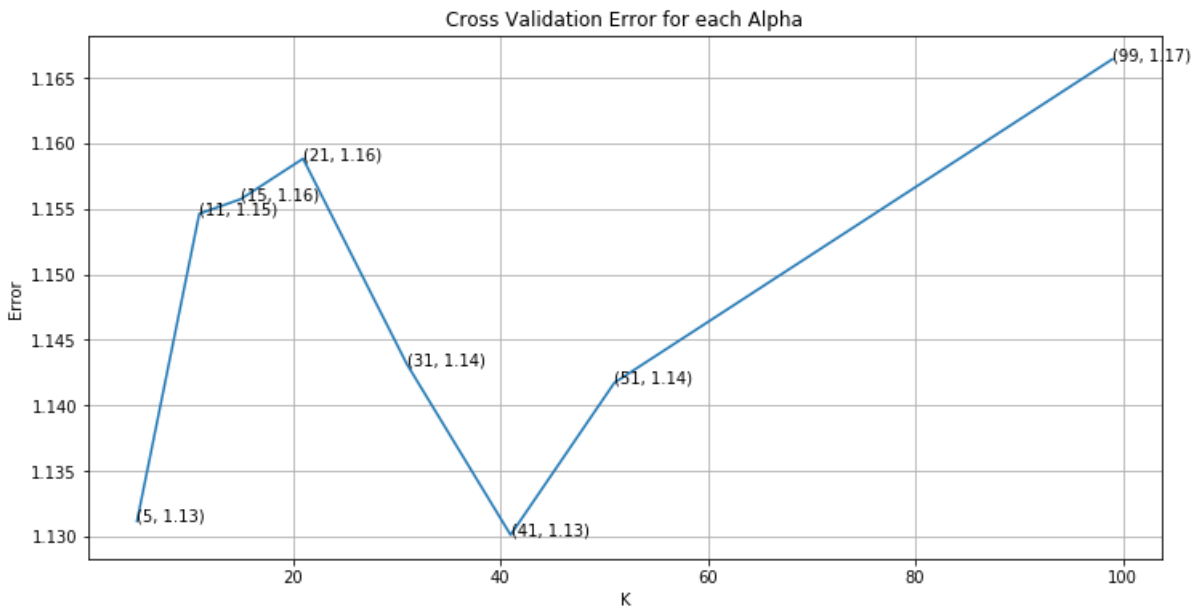
4.2.1. Hyper parameter tuning

In [89]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification

k= [5, 11, 15, 21, 31, 41, 51, 99]

cv_log_error= []
for i in tqdm(k):
    knn= KNeighborsClassifier(n_neighbors= i, n_jobs= -1)
    knn.fit(x_train_rc, ytrain) # Response coding
    cccv= CalibratedClassifierCV(base_estimator= knn, method= 'sigmoid')
    cccv.fit(x_train_rc, ytrain)
    y_pred= cccv.predict_proba(x_cv_rc)
    cv_log_error.append(log_loss(ycv, y_pred, labels= knn.classes_, eps = 1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print('For values of K= ', i, "The log loss is:",log_loss(ycv, y_pred, labels=knn.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(k, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((k[i], np.round(j, 2)), (k[i], cv_log_error[i]))
plt.title('Cross Validation Error for each K')
plt.xlabel('K')
plt.ylabel('Error')
plt.grid()
plt.show()

best_k= np.argmin(cv_log_error)

knn= KNeighborsClassifier(n_neighbors= k[best_k], n_jobs= -1)
knn.fit(x_train_rc, ytrain)
cccv= CalibratedClassifierCV(base_estimator= knn, method= 'sigmoid')
cccv.fit(x_train_rc, ytrain)
```

```
y_pred= cccv.predict_proba(x_train_rc)
print('For values of best k: ', k[best_k], "The train log loss is: ",
    log_loss(ytrain, y_pred, labels= knn.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_rc)
print('For values of best k: ', k[best_k], "The cv log loss is: ",
    log_loss(ycv, y_pred, labels= knn.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_rc)
print('For values of best k: ', k[best_k], "The test log loss is: ",
    log_loss(ytest, y_pred, labels= knn.classes_, eps = 1e-15))
```

```
12%|          | 1/8 [00:00<00:05,  1.25it/s]

For values of alpha =  5 The log loss is: 1.1311676344071935

25%|          | 2/8 [00:01<00:04,  1.28it/s]

For values of alpha =  11 The log loss is: 1.1546262072100628

38%|          | 3/8 [00:02<00:03,  1.30it/s]

For values of alpha =  15 The log loss is: 1.1557390757419992

50%|          | 4/8 [00:03<00:03,  1.31it/s]

For values of alpha =  21 The log loss is: 1.1588252832343167

62%|          | 5/8 [00:03<00:02,  1.33it/s]

For values of alpha =  31 The log loss is: 1.1431537432680594

75%|          | 6/8 [00:04<00:01,  1.33it/s]

For values of alpha =  41 The log loss is: 1.1301152119603657

88%|          | 7/8 [00:05<00:00,  1.34it/s]

For values of alpha =  51 The log loss is: 1.1417246619464323

100%|          | 8/8 [00:05<00:00,  1.34it/s]

For values of alpha =  99 The log loss is: 1.166420946945628
```



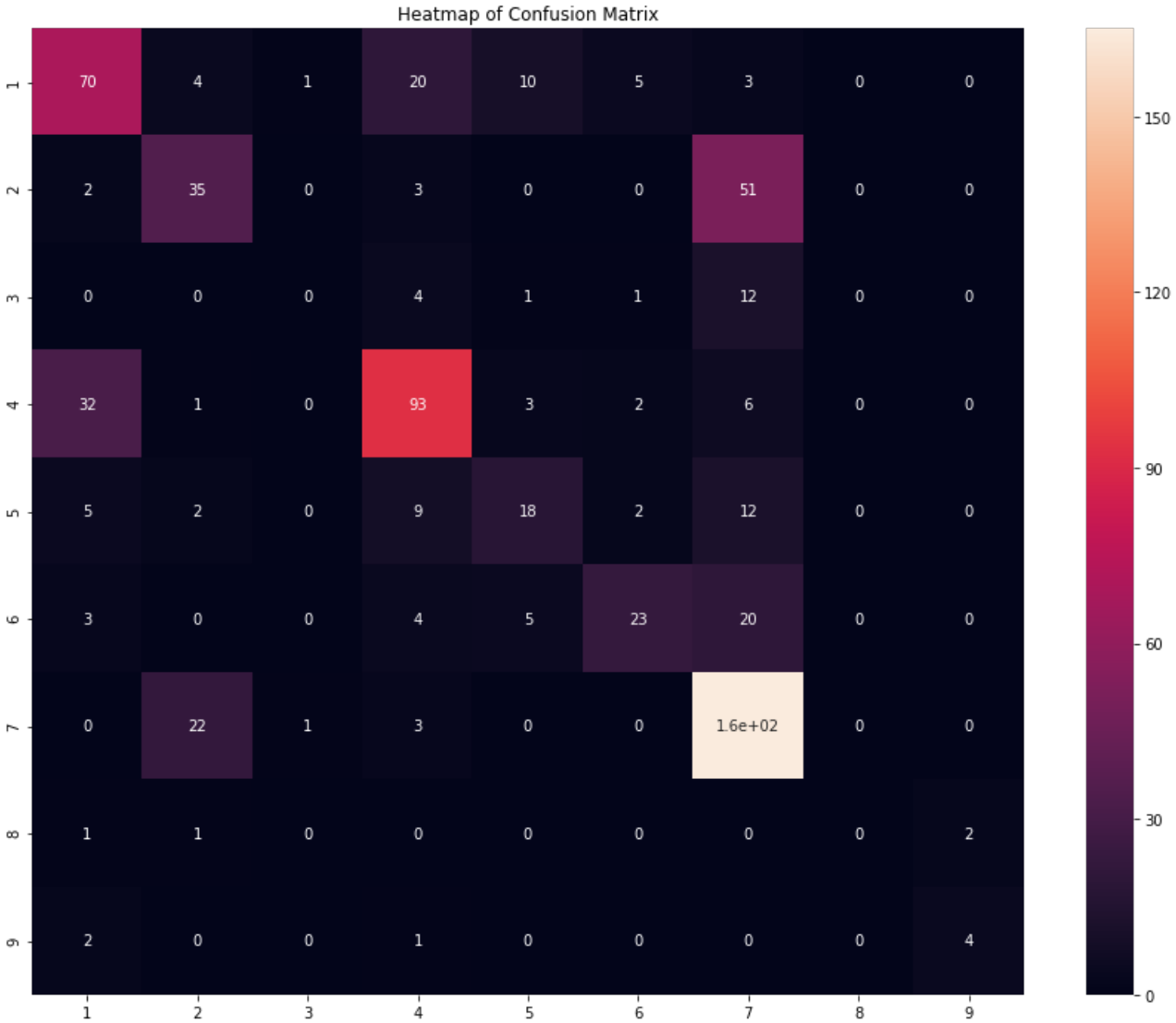

```
For values of best alpha:  41 The train log loss is:  0.8485923706654264
For values of best alpha:  41 The cv log loss is:  1.1301152119603657
For values of best alpha:  41 The test log loss is:  1.0839157361302618
```

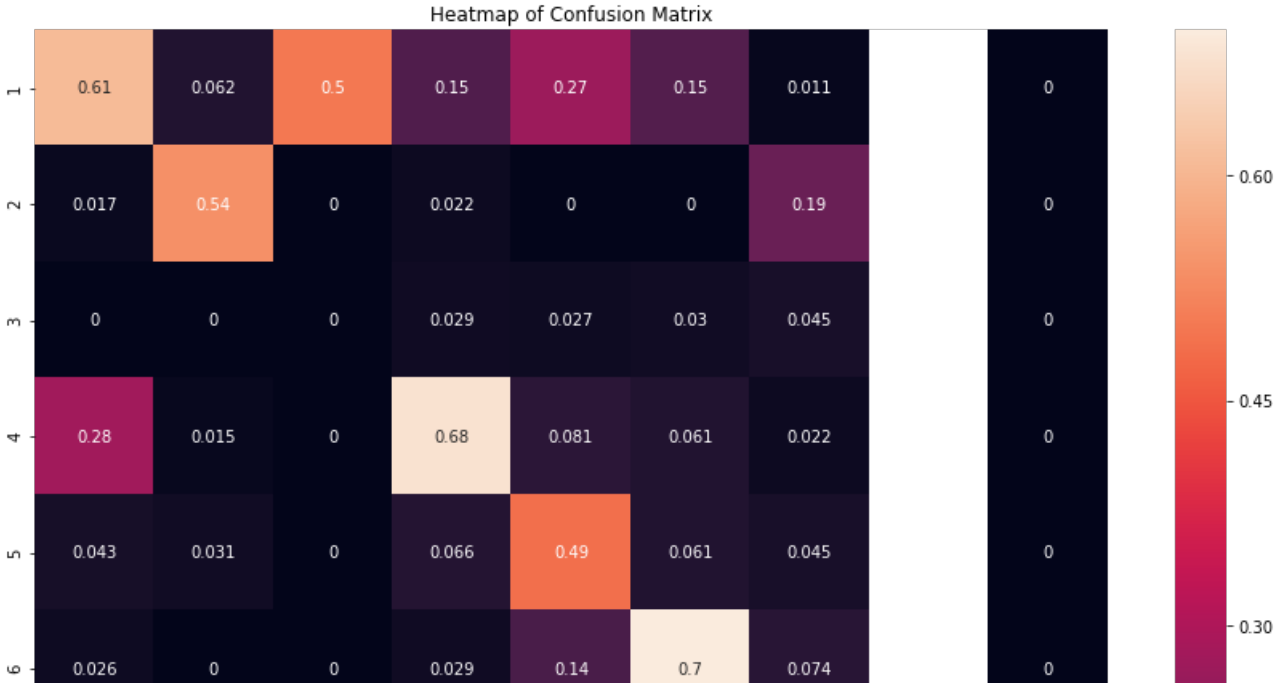
4.2.2. Testing the model with best hyper paramters

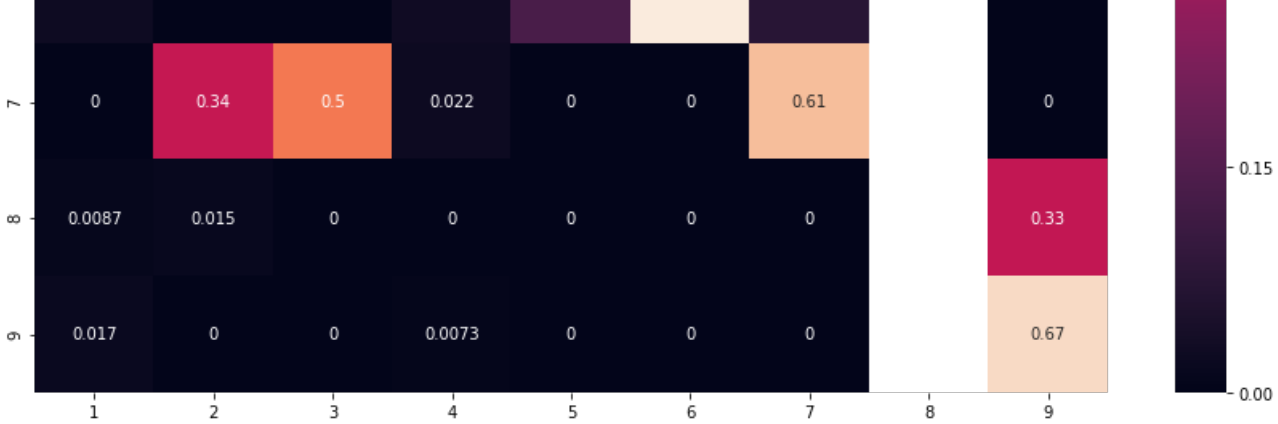
In [90]:

```
predict_and_plot_confusion_matrix(x_train_rc, ytrain, x_test_rc, ytest,
                                  KNeighborsClassifier(n_neighbors= k[best_k], n_jobs= -1))
```

Log loss : 1.0839157361302618
Number of mis-classified points : 0.3855421686746988







4.2.3.Sample Query point -1

In [92]:

```
test_point_index = 1
predicted_cls = cccv.predict(x_test_rc[0].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", ytest[test_point_index])
neighbors = knn.kneighbors(x_test_rc[test_point_index].reshape(1, -1), k[best_k])
print("The ",k[best_k]," nearest neighbours of the test points belongs to classes",ytrain[neighbors[1][0]])
print("Fequency of nearest points :",Counter(ytrain[neighbors[1][0]]))
```

```
Predicted Class : 6
Actual Class : 1
The  41  nearest neighbours of the test points belongs to classes [1 1 1 1 4 1 1 1 1 1 4 4 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 1 1 1 6 1 1 1 1 1 4
 1 4 1 1]
Fequency of nearest points : Counter({1: 34, 4: 6, 6: 1})
```

4.2.4.Sample Query point -2

In [97]:

```
test_point_index = 5
predicted_cls = cccv.predict(x_test_rc[0].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", ytest[test_point_index])
neighbors = knn.kneighbors(x_test_rc[test_point_index].reshape(1, -1), k[best_k])
print("The ",k[best_k]," nearest neighbours of the test points belongs to classes",ytrain[neighbors[1][0]])
print("Fequency of nearest points :",Counter(ytrain[neighbors[1][0]]))
```

```
Predicted Class : 6
Actual Class : 6
The  41  nearest neighbours of the test points belongs to classes [1 5 6 7 1 4 2 7 1 7 6 6 4 7 1 6 1 5 1 1 3 1 5 9 1 2 1 4 1 6 7 5 1 8 1 4 6
 4 8 2 8]
Fequency of nearest points : Counter({1: 13, 6: 6, 4: 5, 7: 5, 5: 4, 2: 3, 8: 3, 3: 1, 9: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

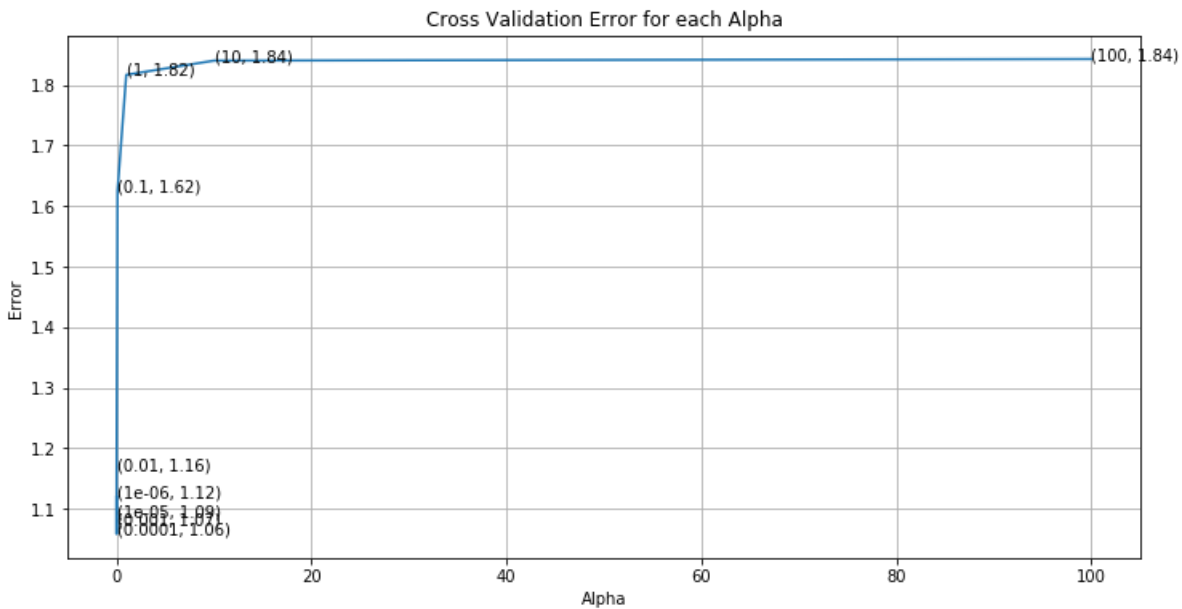
4.3.1.1. Hyper paramter tuning

In [98]:

```
alpha = [10 ** x for x in range(-6, 3)]

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification


cv_log_error= []
for i in tqdm(alpha):
    sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= i, random_state= 42, class_weight= 'balanced')
    sgdc.fit(x_train_ohe, ytrain)
    cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
    cccv.fit(x_train_ohe, ytrain)
    y_pred= cccv.predict_proba(x_cv_ohe)
    cv_log_error.append(log_loss(ycv, y_pred, labels= sgdc.classes_, eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(ycv, y_pred, labels=sgdc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42, class_weight= 'balanced')
sgdc.fit(x_train_ohe, ytrain)
cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
cccv.fit(x_train_ohe, ytrain)
```

```
y_pred= cccv.predict_proba(x_train_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
    log_loss(ytrain, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
    log_loss(ycv, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
    log_loss(ytest, y_pred, labels= sgdc.classes_, eps = 1e-15))
```

```
11%|          | 1/9 [00:04<00:33,  4.24s/it]

For values of alpha =  1e-06 The log loss is: 1.1195204491654112

22%|          | 2/9 [00:07<00:28,  4.08s/it]

For values of alpha =  1e-05 The log loss is: 1.0870685464183478

33%|          | 3/9 [00:11<00:22,  3.82s/it]

For values of alpha =  0.0001 The log loss is: 1.0583082152823189

44%|          | 4/9 [00:15<00:19,  3.90s/it]

For values of alpha =  0.001 The log loss is: 1.073150383317793

56%|          | 5/9 [00:18<00:14,  3.66s/it]

For values of alpha =  0.01 The log loss is: 1.1645269072596407

67%|          | 6/9 [00:21<00:10,  3.44s/it]

For values of alpha =  0.1 The log loss is: 1.6249840712964545

78%|          | 7/9 [00:23<00:06,  3.13s/it]

For values of alpha =  1 The log loss is: 1.8168337979221387

89%|          | 8/9 [00:25<00:02,  2.82s/it]

For values of alpha =  10 The log loss is: 1.8402408805452228

100%|          | 9/9 [00:27<00:00,  3.11s/it]

For values of alpha =  100 The log loss is: 1.8429143760059246
```



For values of best alpha: 0.0001 The train log loss is: 0.4232396135673121
For values of best alpha: 0.0001 The cv log loss is: 1.0583082152823189
For values of best alpha: 0.0001 The test log loss is: 0.9621309970690596

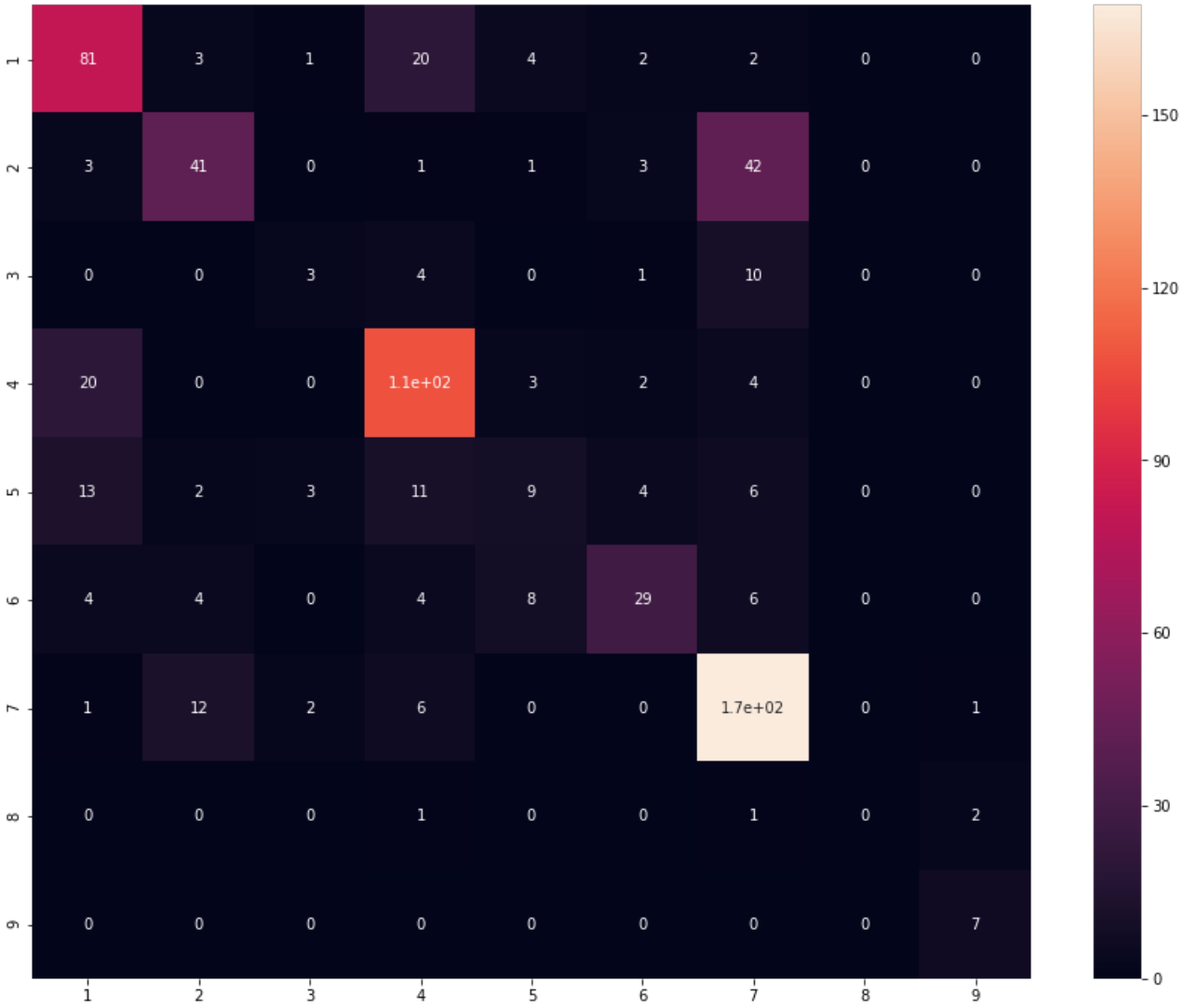
4.3.1.2. Testing the model with best hyper paramters

In [106]:

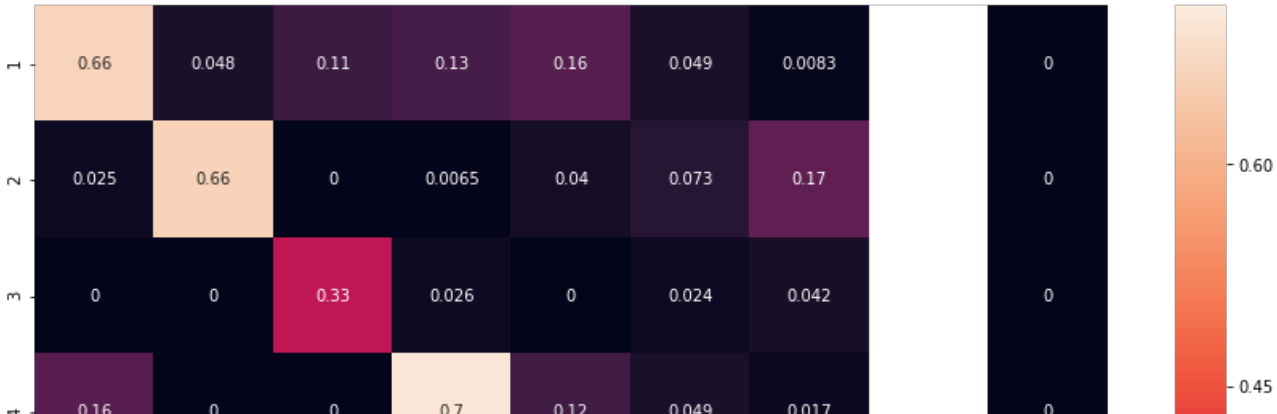
```
predict_and_plot_confusion_matrix(x_train_ohe, ytrain, x_test_ohe, ytest,
                                  SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42,
                                                class_weight= 'balanced'))
```

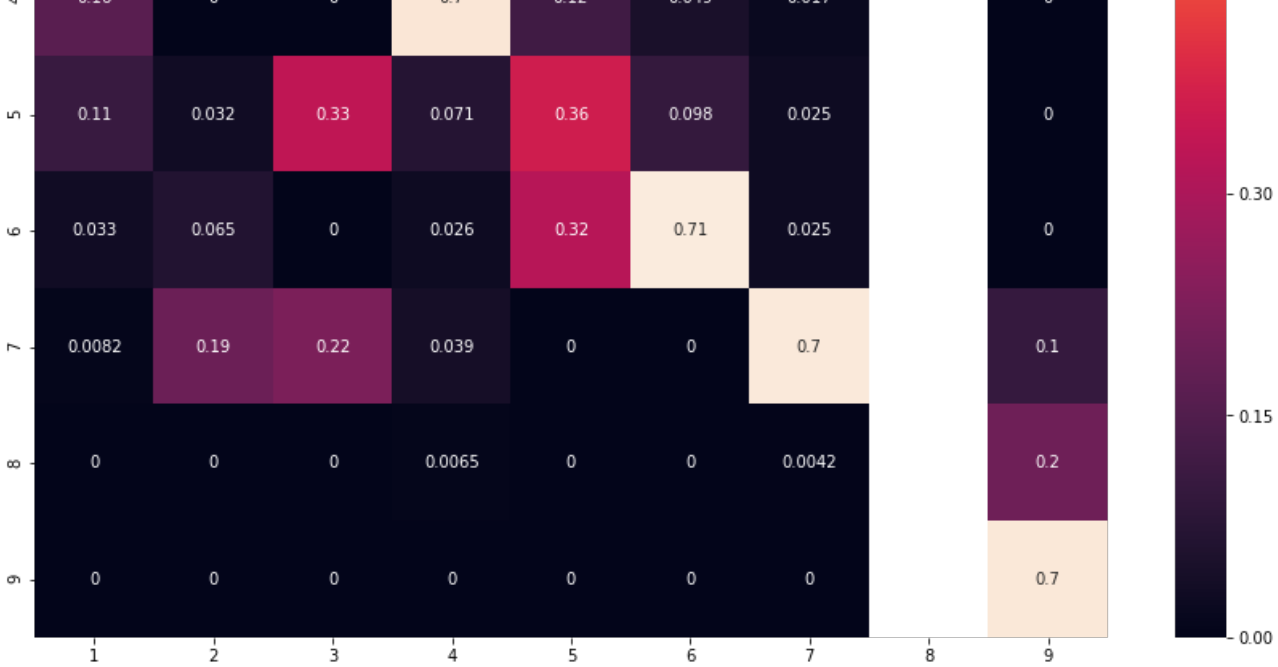
Log loss : 0.9621309970690596
Number of mis-classified points : 0.32680722891566266

********** Confusion Matrix **********



********** Precision Matrix **********





********** Recall Matrix **********

4.3.1.3. Feature Importance

In [108]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < gene_train.shape[1]:
```

```
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = text_train_vocab[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,text_train_vocab[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most important features are present in our query point")
    print("-"*50)
    print("The features that are most important of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [111]:

```
test_point_index = 1
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-sgdc.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[4.884e-01 4.700e-03 1.000e-03 4.861e-01 3.100e-03 1.600e-03 1.300e-02
  1.800e-03 3.000e-04]]
Actual Class : 1
--------------------------------------------------
146 Text feature [aa] present in test data point [True]
229 Text feature [accumulating] present in test data point [True]
416 Text feature [abnormality] present in test data point [True]
428 Text feature [acceptor] present in test data point [True]
436 Text feature [abrogating] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [118]:

```
test_point_index = 6
no_feature = 100
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-sgdc.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[4.478e-01 8.900e-03 2.100e-03 5.320e-02 2.454e-01 2.049e-01 3.190e-02
  5.700e-03 1.000e-04]]
Actual Class : 5
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [113]:

```
alpha = [10 ** x for x in range(-6, 3)]

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

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification


cv_log_error= []
for i in tqdm(alpha):
    sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= i, random_state= 42)
    sgdc.fit(x_train_ohe, ytrain)
    cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
    cccv.fit(x_train_ohe, ytrain)
    y_pred= cccv.predict_proba(x_cv_ohe)
    cv_log_error.append(log_loss(ycv, y_pred, labels= sgdc.classes_, eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(ycv, y_pred, labels=sgdc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

sgdc= SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42)
sgdc.fit(x_train_ohe, ytrain)
cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
cccv.fit(x_train_ohe, ytrain)

y_pred= cccv.predict_proba(x_train_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
      log_loss(ytrain, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
      log_loss(ycv, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
      log_loss(ytest, y_pred, labels= sgdc.classes_, eps = 1e-15))
```

```
11%|▉         | 1/9 [00:03<00:31,  3.94s/it]

For values of alpha =  1e-06 The log loss is: 1.1086756134013742

22%|▉▉         | 2/9 [00:07<00:26,  3.82s/it]

For values of alpha =  1e-05 The log loss is: 1.1119089114101985

33%|▉▉▉▉       | 3/9 [00:10<00:21,  3.56s/it]

For values of alpha =  0.0001 The log loss is: 1.0932040803184244

44%|▉▉▉▉▉▉     | 4/9 [00:13<00:17,  3.54s/it]

For values of alpha =  0.001 The log loss is: 1.1467219992884568

56%|▉▉▉▉▉▉▉▉   | 5/9 [00:16<00:13,  3.35s/it]
```

```
For values of alpha =  0.01 The log loss is: 1.4504600266775927
```

```
67%|[redacted]| 6/9 [00:19<00:09,  3.19s/it]
```

```
For values of alpha =  0.1 The log loss is: 1.9121412785363048
```

```
78%|[redacted]| 7/9 [00:21<00:05,  2.92s/it]
```

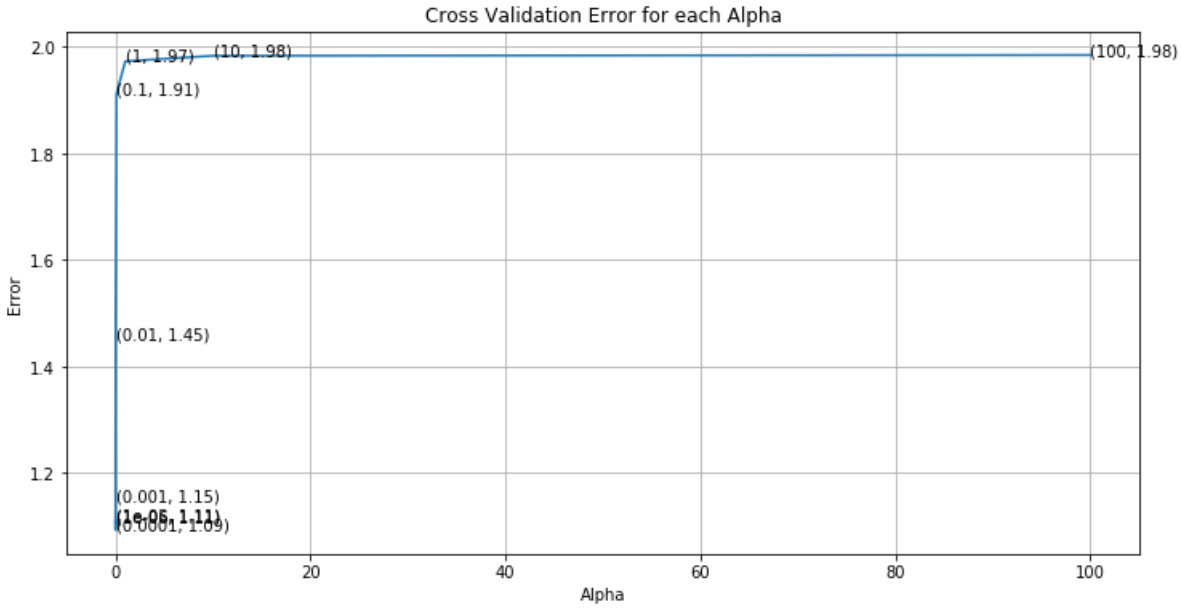
```
For values of alpha =  1 The log loss is: 1.9731543852550064
```

```
89%|[redacted]| 8/9 [00:24<00:02,  2.67s/it]
```

```
For values of alpha =  10 The log loss is: 1.9835919912561724
```

```
100%|[redacted]| 9/9 [00:26<00:00,  2.91s/it]
```

```
For values of alpha =  100 The log loss is: 1.984991548844364
```



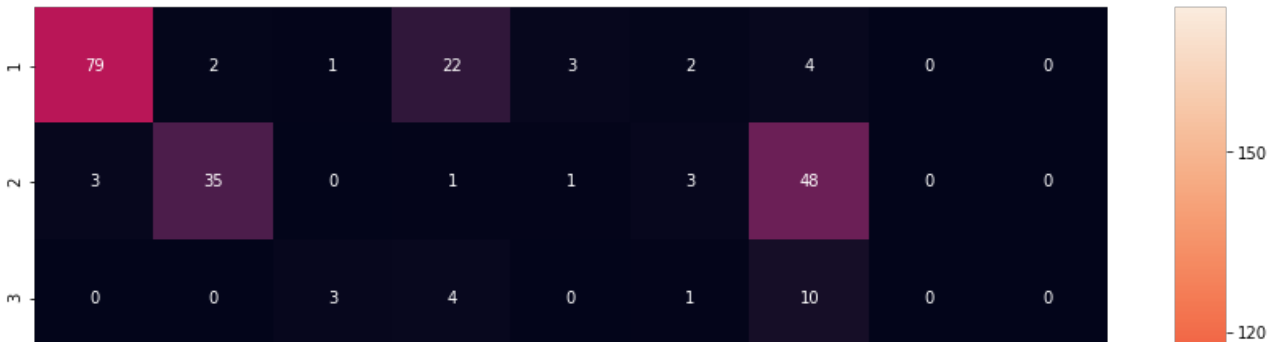

```
For values of best alpha:  0.0001 The train log loss is:  0.4209160904649432
For values of best alpha:  0.0001 The cv log loss is:  1.0932040803184244
For values of best alpha:  0.0001 The test log loss is:  0.9883086677144824
```

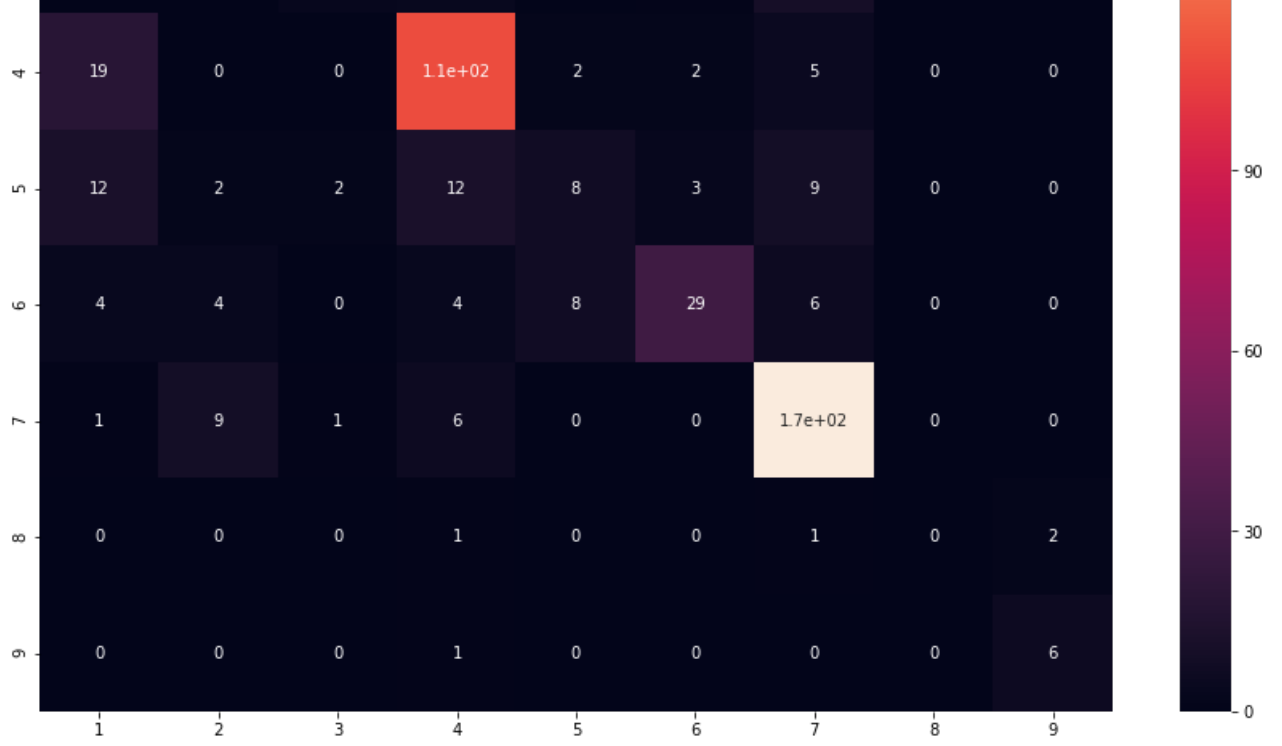
4.3.2.2. Testing model with best hyper parameters

In [114]:

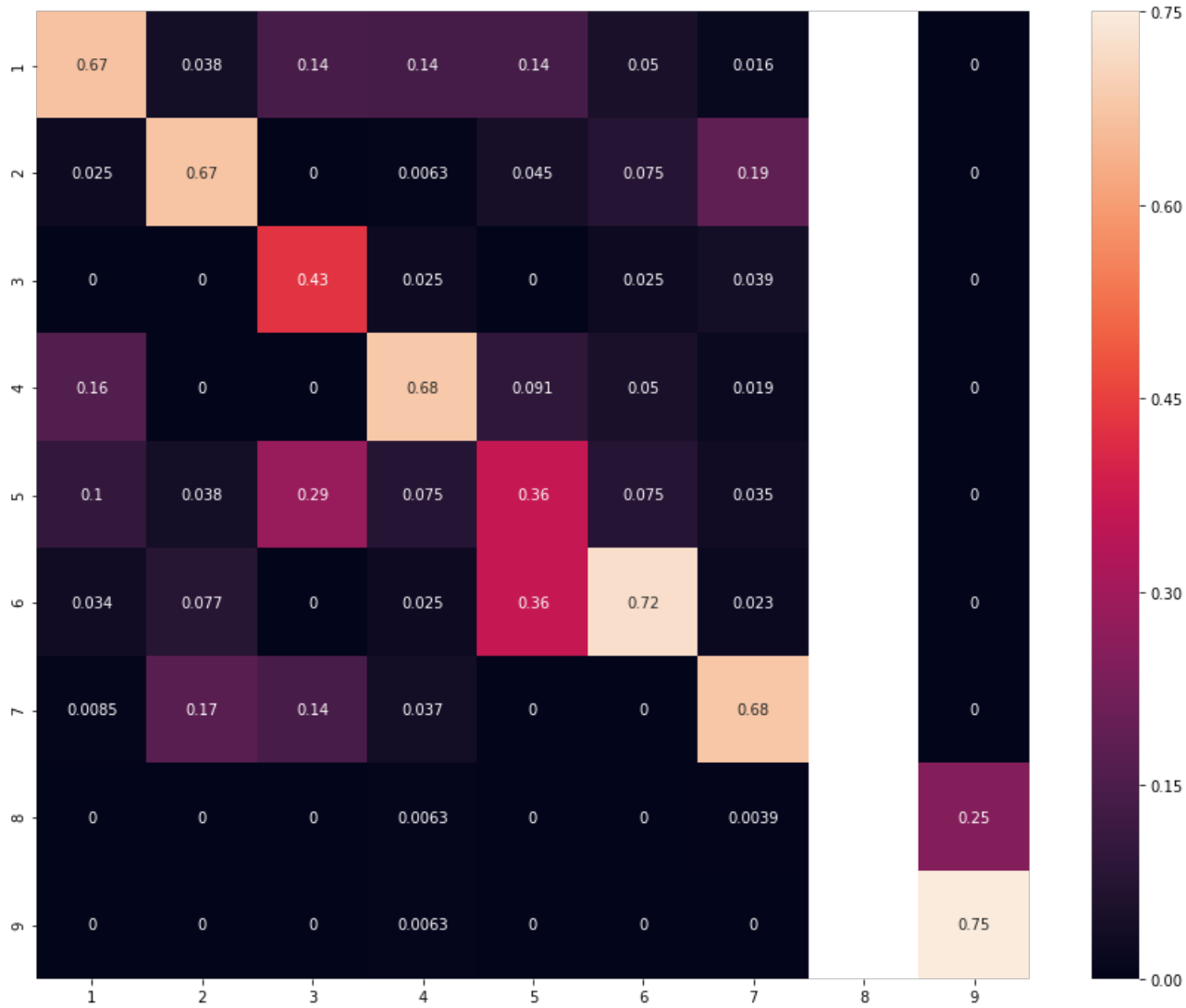
```
predict_and_plot_confusion_matrix(x_train_ohe, ytrain, x_test_ohe, ytest,
                                  SGDClassifier(loss= 'log', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42))
```

```
Log loss : 0.9883086677144824
Number of mis-classified points : 0.3328313253012048

************************************ Confusion Matrix ************************************
```

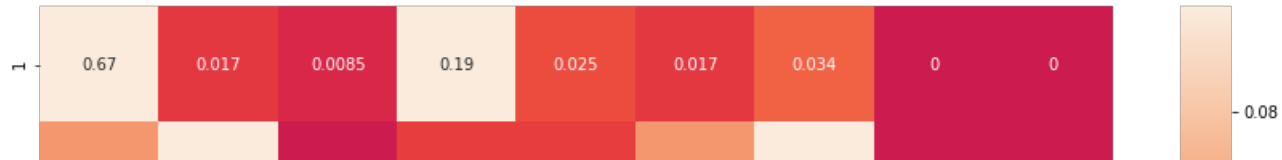




********** Precision Matrix **********



********** Recall Matrix **********



4.3.2.3. Feature Importance, Correctly Classified point

In [120]:

```
test_point_index = 2
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-sgdc.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[1.040e-02 2.200e-03 1.100e-03 4.400e-03 2.210e-02 9.565e-01 9.000e-04
  2.400e-03 0.000e+00]]
Actual Class : 6
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [115]:

```
test_point_index = 1
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-sgdc.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[4.452e-01 4.200e-03 5.000e-04 5.200e-01 2.400e-03 1.100e-03 2.250e-02
  4.100e-03 0.000e+00]]
Actual Class : 1
--------------------------------------------------
```

```
242 Text feature [aacr] present in test data point [True]
269 Text feature [achieve] present in test data point [True]
350 Text feature [achieved] present in test data point [True]
444 Text feature [acquisition] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [121]:

```
alpha = [10 ** x for x in range(-6, 3)]

# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
# ----------------------------
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification


cv_log_error= []
for i in tqdm(alpha):
    sgdc= SGDClassifier(loss= 'hinge', penalty= 'l2', alpha= i, random_state= 42, class_weight= 'balanced')
    sgdc.fit(x_train_ohe, ytrain)
    cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
    cccv.fit(x_train_ohe, ytrain)
    y_pred= cccv.predict_proba(x_cv_ohe)
    cv_log_error.append(log_loss(ycv, y_pred, labels= sgdc.classes_, eps = 1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(ycv, y_pred, labels=sgdc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (12, 6))
ax.plot(alpha, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((alpha[i], np.round(j, 2)), (alpha[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Alpha')
plt.xlabel('Alpha')
plt.ylabel('Error')
plt.grid()
plt.show()

best_alpha= np.argmin(cv_log_error)

sgdc= SGDClassifier(loss= 'hinge', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42, class_weight= 'balanced')
sgdc.fit(x_train_ohe, ytrain)
cccv= CalibratedClassifierCV(base_estimator= sgdc, method= 'sigmoid')
cccv.fit(x_train_ohe, ytrain)

y_pred= cccv.predict_proba(x_train_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The train log loss is: ",
      log_loss(ytrain, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The cv log loss is: ",
      log_loss(ycv, y_pred, labels= sgdc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_ohe)
print('For values of best alpha: ', alpha[best_alpha], "The test log loss is: ",
      log_loss(ytest, y_pred, labels= sgdc.classes_, eps = 1e-15))
```

```
11%|▉         | 1/9 [00:02<00:17,  2.22s/it]
```

```
For values of alpha =  1e-06 The log loss is: 1.1228315127012858
```

22%|[redacted] | 2/9 [00:04<00:15, 2.22s/it]

For values of alpha = 1e-05 The log loss is: 1.114202189579543

33%|[redacted] | 3/9 [00:07<00:14, 2.35s/it]

For values of alpha = 0.0001 The log loss is: 1.0640641026833166

44%|[redacted] | 4/9 [00:10<00:13, 2.71s/it]

For values of alpha = 0.001 The log loss is: 1.0777160973207607

56%|[redacted] | 5/9 [00:13<00:10, 2.60s/it]

For values of alpha = 0.01 The log loss is: 1.3511606015130861

67%|[redacted] | 6/9 [00:15<00:07, 2.50s/it]

For values of alpha = 0.1 The log loss is: 1.887003669080445

78%|[redacted] | 7/9 [00:19<00:06, 3.04s/it]

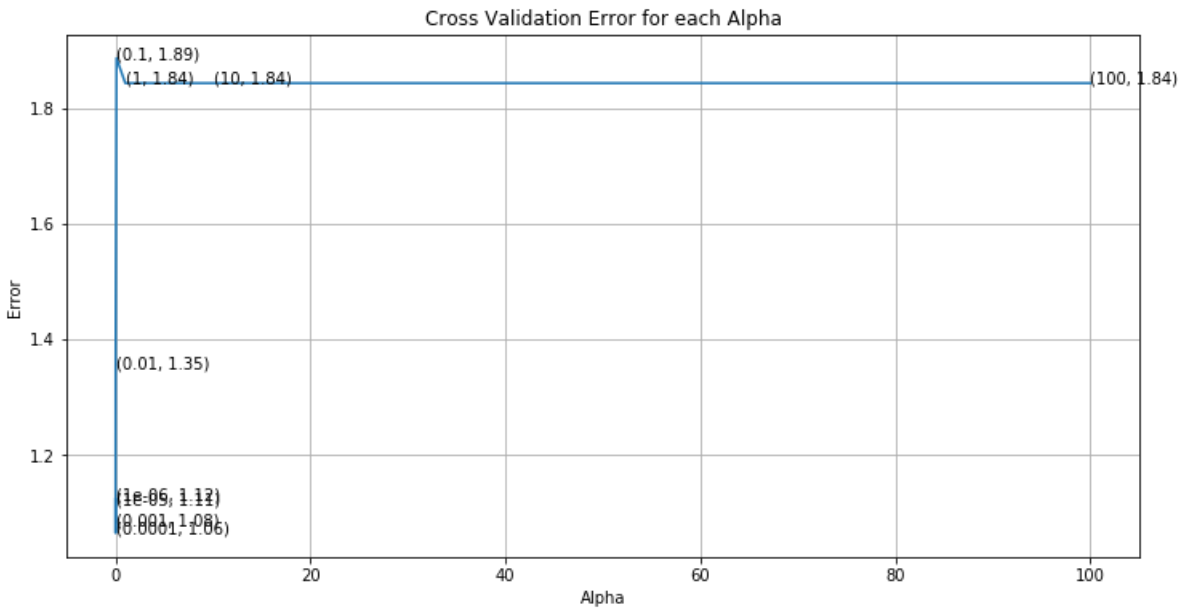
For values of alpha = 1 The log loss is: 1.843498894331801

89%|[redacted] | 8/9 [00:21<00:02, 2.77s/it]

For values of alpha = 10 The log loss is: 1.8434971865585768

100%|[redacted]| 9/9 [00:23<00:00, 2.64s/it]

For values of alpha = 100 The log loss is: 1.8434971632715997



For values of best alpha: 0.0001 The train log loss is: 0.4172309536000051
For values of best alpha: 0.0001 The cv log loss is: 1.0640641026833166
For values of best alpha: 0.0001 The test log loss is: 1.0034688950931545

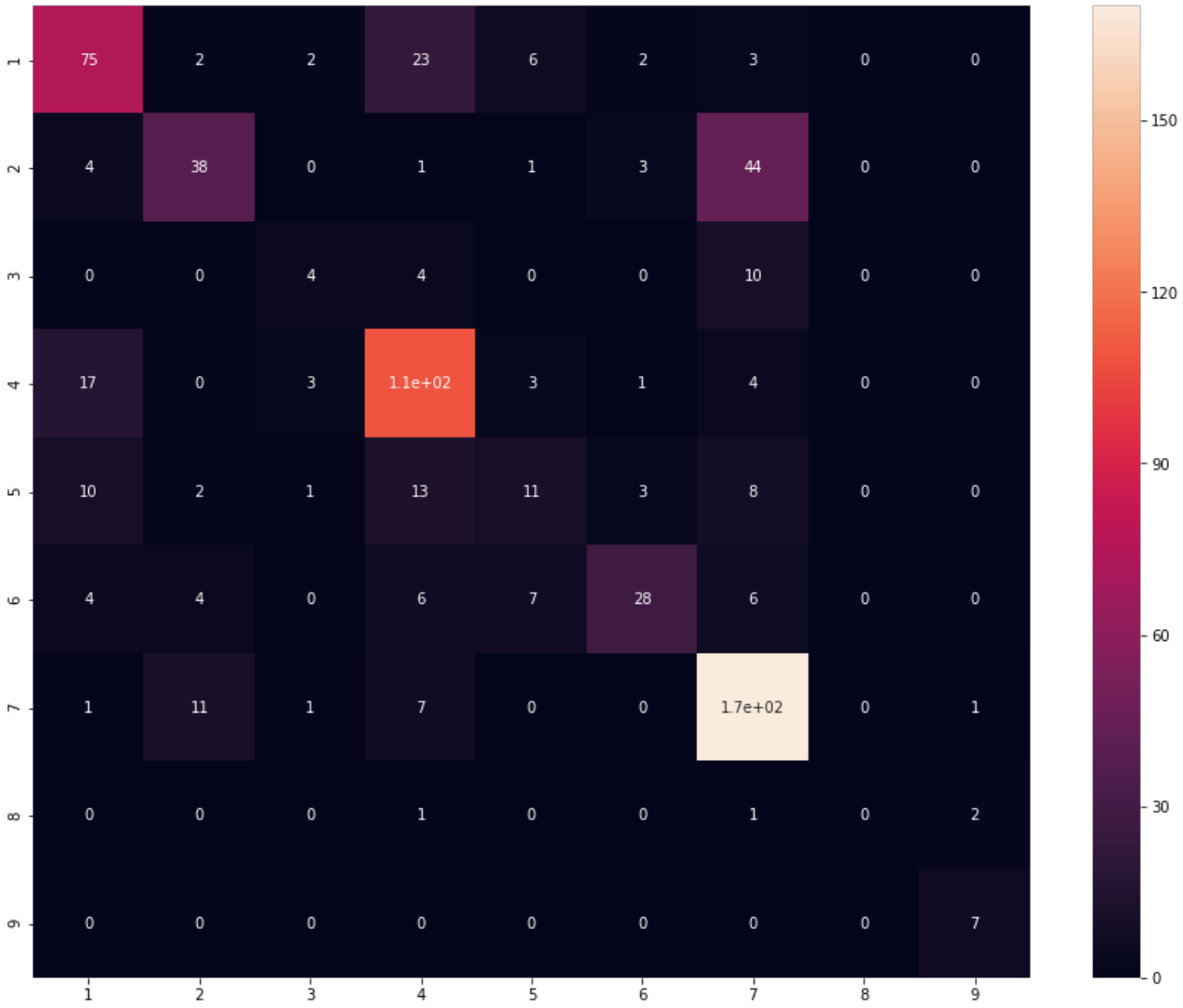
4.4.2. Testing model with best hyper parameters

In [122]:

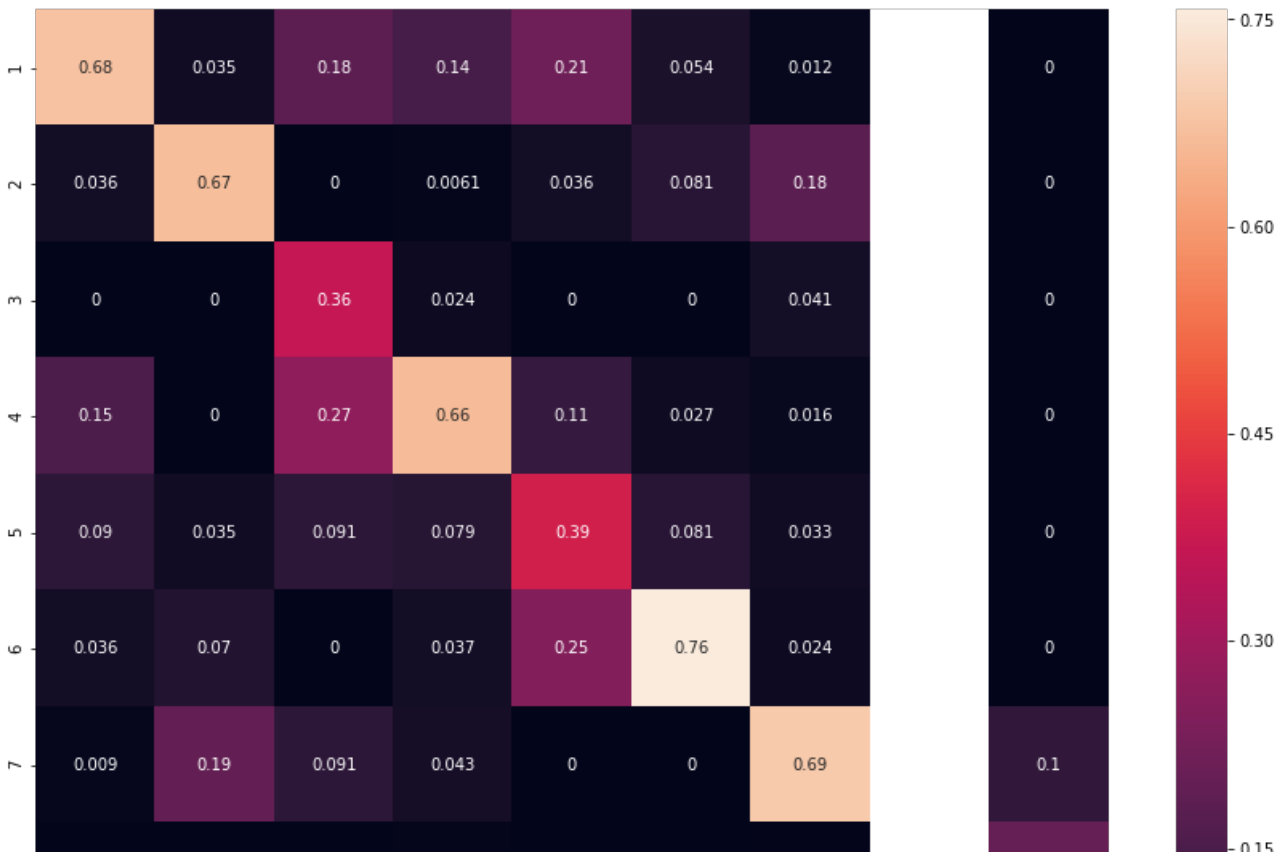
```
predict_and_plot_confusion_matrix(x_train_ohe, ytrain, x_test_ohe, ytest,
                                  SGDClassifier(loss= 'hinge', penalty= 'l2', alpha= alpha[best_alpha], random_state= 42,
                                                class_weight= 'balanced'))
```

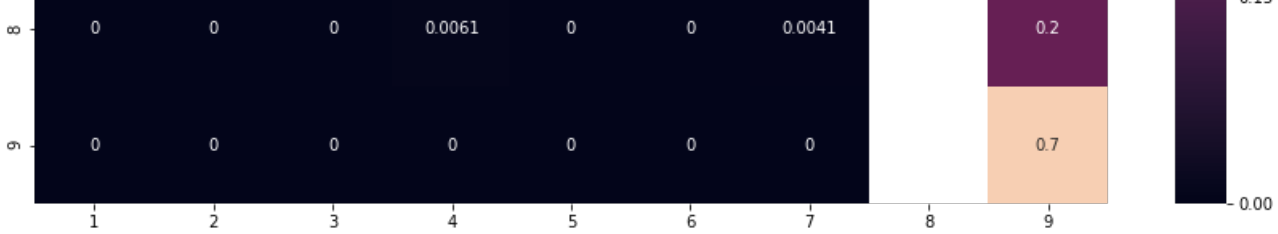
Log loss : 1.0034688950931545
Number of mis-classified points : 0.33433734939759036

********** Confusion Matrix **********

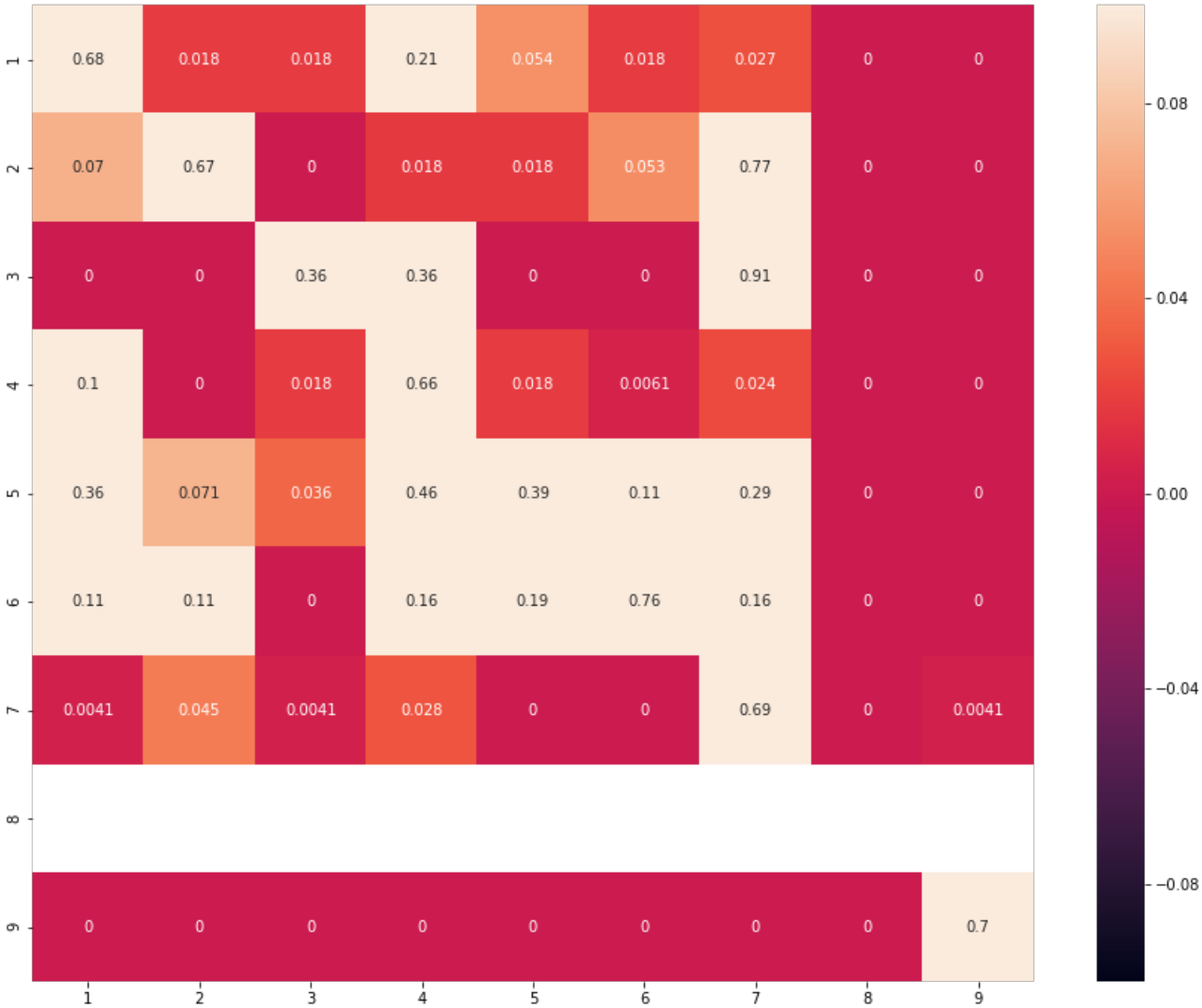


********** Precision Matrix **********





********** Recall Matrix **********



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [124]:

```
test_point_index = 2
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-sgdc.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[1.860e-02 2.550e-02 1.050e-02 1.210e-02 3.430e-02 8.862e-01 1.180e-02
  9.000e-04 2.000e-04]]
Actual Class : 6
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [123]:

```
test_point_index = 1
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-sgdc.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.4401 0.0078 0.0034 0.4652 0.0055 0.0043 0.0718 0.0014 0.0005]]
Actual Class : 1
--------------------------------------------------
263 Text feature [aacr] present in test data point [True]
308 Text feature [achieve] present in test data point [True]
345 Text feature [achieved] present in test data point [True]
473 Text feature [acquisition] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [178]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.
# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/
#sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
# ----------------------------
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification

estimator= [5, 10, 50, 100, 200, 500, 1000]
max_depth= [2,3,4,5,6,7,8,9,10]
cv_log_error= []
for i in tqdm(estimator):
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        rfc= RandomForestClassifier(n_estimators= i, max_depth= j, class_weight= 'balanced', n_jobs= -1)
        rfc.fit(x_train_ohe, ytrain)
        cccv= CalibratedClassifierCV(base_estimator= rfc, method= 'sigmoid')
        cccv.fit(x_train_ohe, ytrain)
        y_pred= cccv.predict_proba(x_cv_ohe)
        cv_log_error.append(log_loss(ycv, y_pred, labels= rfc.classes_, eps = 1e-15))
        print("The log loss is:",log_loss(ycv, y_pred, labels= rfc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (16, 8))
features = np.dot(np.array(estimator)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((estimator[int(i%2)],max_depth[int(i%2)],str(j)), (features[i], cv_log_error[i]))
```

```
plt.title('Cross Validation Error for each Estimator & Max-Depth')
plt.xlabel('Estimator & Max-Depth')
plt.ylabel('Error')
plt.grid()
plt.show()

best_est_md= np.argmin(cv_log_error)
# The estimator is 1000 and max_depth is 8 - are considered by best_est_md

rfc= RandomForestClassifier(n_estimators= 1000, max_depth= 8,
                            random_state=42, n_jobs=-1, class_weight= 'balanced')
rfc.fit(x_train_ohe, ytrain)
cccv= CalibratedClassifierCV(base_estimator= rfc, method= 'sigmoid')
cccv.fit(x_train_ohe, ytrain)

y_pred= cccv.predict_proba(x_train_ohe)
print('For values of best estimator: ', 1000, 'and max depth:', 8,
    "The train log loss is: ", log_loss(ytrain, y_pred, labels= rfc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_ohe)
print('For values of best estimator: ', 1000, 'and max depth:', 8,
    "The cv log loss is: ", log_loss(ycv, y_pred, labels= rfc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_ohe)
print('For values of best estimator: ', 1000, 'and max depth:', 8,
    "The test log loss is: ", log_loss(ytest, y_pred, labels= rfc.classes_, eps = 1e-15))
```

```
0%|          | 0/7 [00:00<?, ?it/s]

for n_estimators = 5 and max depth =  2
The log loss is: 1.6532590426959284
for n_estimators = 5 and max depth =  3
The log loss is: 1.7024324731485772
for n_estimators = 5 and max depth =  4
The log loss is: 1.6168201016675112
for n_estimators = 5 and max depth =  5
The log loss is: 1.5923146605419267
for n_estimators = 5 and max depth =  6
The log loss is: 1.5924207515324822
for n_estimators = 5 and max depth =  7
The log loss is: 1.5650965310215212
for n_estimators = 5 and max depth =  8
The log loss is: 1.57202193787134
for n_estimators = 5 and max depth =  9
The log loss is: 1.533064344544194
for n_estimators = 5 and max depth =  10

14%|▉         | 1/7 [00:15<01:33, 15.59s/it]

The log loss is: 1.518216158359641
for n_estimators = 10 and max depth =  2
The log loss is: 1.6746876783568574
for n_estimators = 10 and max depth =  3
The log loss is: 1.6355042118259684
for n_estimators = 10 and max depth =  4
The log loss is: 1.5299569520798
for n_estimators = 10 and max depth =  5
The log loss is: 1.4970077141719476
for n_estimators = 10 and max depth =  6
The log loss is: 1.5364469378279266
for n_estimators = 10 and max depth =  7
The log loss is: 1.4520270265832405
for n_estimators = 10 and max depth =  8
The log loss is: 1.495793371465223
for n_estimators = 10 and max depth =  9
The log loss is: 1.4559955059975098
for n_estimators = 10 and max depth =  10

29%|▉▉▉       | 2/7 [00:31<01:18, 15.78s/it]

The log loss is: 1.4485795725227237
for n_estimators = 50 and max depth =  2
The log loss is: 1.586335654163959
for n_estimators = 50 and max depth =  3
The log loss is: 1.4913445840497017
for n_estimators = 50 and max depth =  4
The log loss is: 1.3804929440597953
for n_estimators = 50 and max depth =  5
The log loss is: 1.3456577799484357
for n_estimators = 50 and max depth =  6
```

```
The log loss is: 1.3210834187767293
for n_estimators = 50 and max depth =  7
The log loss is: 1.2869074371796183
for n_estimators = 50 and max depth =  8
The log loss is: 1.304043087997293
for n_estimators = 50 and max depth =  9
The log loss is: 1.2977394021121171
for n_estimators = 50 and max depth =  10
```

```
43%|▌▌▌▌▌▌      | 3/7 [00:55<01:12, 18.04s/it]
```

```
The log loss is: 1.297372800108202
for n_estimators = 100 and max depth =  2
The log loss is: 1.569958404241197
for n_estimators = 100 and max depth =  3
The log loss is: 1.434827823966733
for n_estimators = 100 and max depth =  4
The log loss is: 1.3348526262552332
for n_estimators = 100 and max depth =  5
The log loss is: 1.2901142248046173
for n_estimators = 100 and max depth =  6
The log loss is: 1.281654537517901
for n_estimators = 100 and max depth =  7
The log loss is: 1.2687318808034733
for n_estimators = 100 and max depth =  8
The log loss is: 1.2685350133395734
for n_estimators = 100 and max depth =  9
The log loss is: 1.265500011690373
for n_estimators = 100 and max depth =  10
```

```
57%|▌▌▌▌▌▌▌     | 4/7 [01:27<01:07, 22.47s/it]
```

```
The log loss is: 1.2645856171067396
for n_estimators = 200 and max depth =  2
The log loss is: 1.5860971483834916
for n_estimators = 200 and max depth =  3
The log loss is: 1.415552128499012
for n_estimators = 200 and max depth =  4
The log loss is: 1.301802730153224
for n_estimators = 200 and max depth =  5
The log loss is: 1.2639179305430421
for n_estimators = 200 and max depth =  6
The log loss is: 1.2488662974867188
for n_estimators = 200 and max depth =  7
The log loss is: 1.2476253465071725
for n_estimators = 200 and max depth =  8
The log loss is: 1.2408039547531056
for n_estimators = 200 and max depth =  9
The log loss is: 1.2505840662459795
for n_estimators = 200 and max depth =  10
```

```
71%|▌▌▌▌▌▌▌▌▌    | 5/7 [02:23<01:04, 32.46s/it]
```

```
The log loss is: 1.249343568190606
for n_estimators = 500 and max depth =  2
The log loss is: 1.5592113628091278
for n_estimators = 500 and max depth =  3
The log loss is: 1.3813075754266608
for n_estimators = 500 and max depth =  4
The log loss is: 1.2906400317335875
for n_estimators = 500 and max depth =  5
The log loss is: 1.2432512868019865
for n_estimators = 500 and max depth =  6
The log loss is: 1.219657740672712
for n_estimators = 500 and max depth =  7
The log loss is: 1.220836852447544
for n_estimators = 500 and max depth =  8
The log loss is: 1.2282179493815653
for n_estimators = 500 and max depth =  9
The log loss is: 1.236736239927468
for n_estimators = 500 and max depth =  10
```

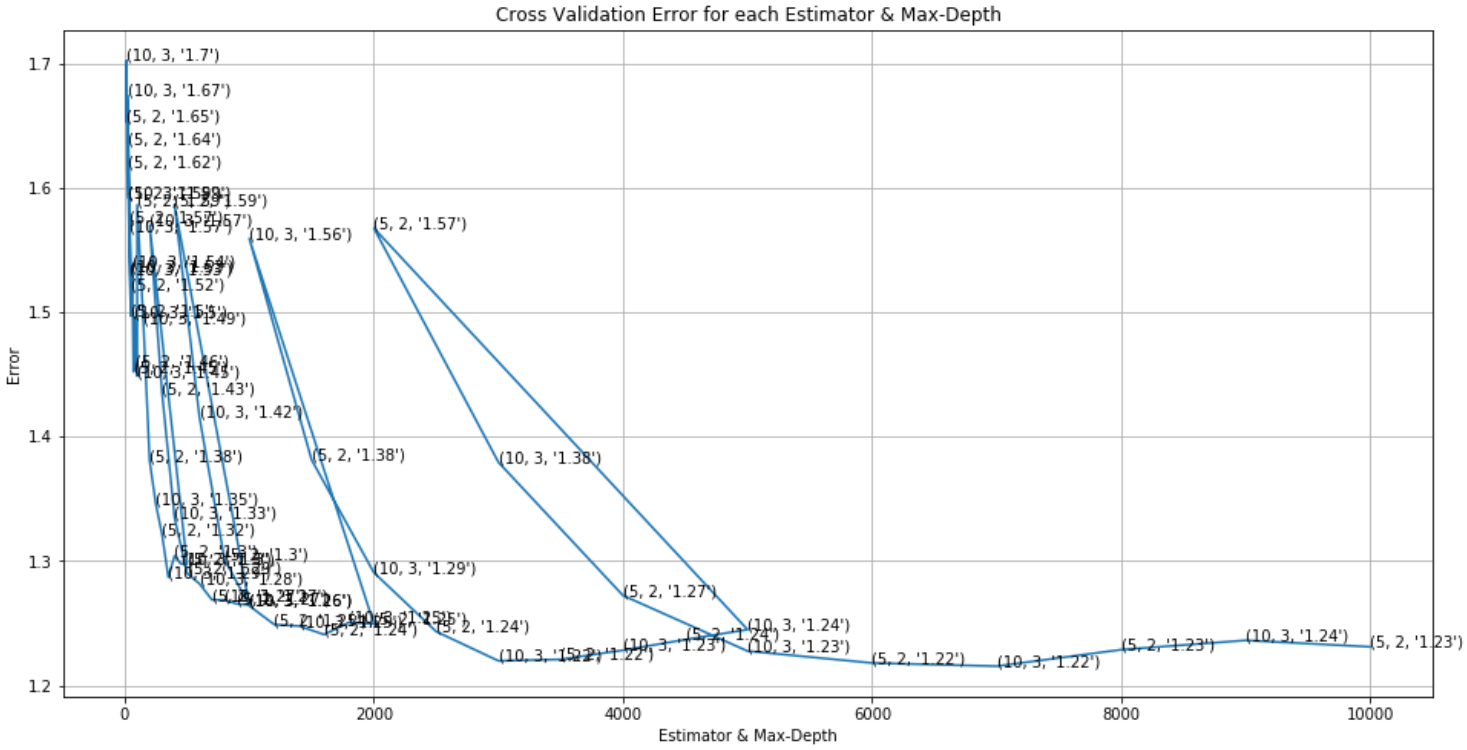
```
86%|▌▌▌▌▌▌▌▌▌▌   | 6/7 [04:19<00:57, 57.54s/it]
```

```
The log loss is: 1.2447742967599096
for n_estimators = 1000 and max depth =  2
The log loss is: 1.5675951655101716
for n_estimators = 1000 and max depth =  3
```

The log loss is: 1.379335551981137
for n_estimators = 1000 and max depth = 4
The log loss is: 1.272203158681552
for n_estimators = 1000 and max depth = 5
The log loss is: 1.2275240745292146
for n_estimators = 1000 and max depth = 6
The log loss is: 1.2179432977743707
for n_estimators = 1000 and max depth = 7
The log loss is: 1.215371695870668
for n_estimators = 1000 and max depth = 8
The log loss is: 1.2286266765279812
for n_estimators = 1000 and max depth = 9
The log loss is: 1.2361370379701275
for n_estimators = 1000 and max depth = 10

100%|[redacted]| 7/7 [08:01<00:00, 68.84s/it]

The log loss is: 1.2309337296097695



For values of best estimator: 1000 and max depth: 8 The train log loss is: 0.6814210321783863
For values of best estimator: 1000 and max depth: 8 The cv log loss is: 1.2272932933700114
For values of best estimator: 1000 and max depth: 8 The test log loss is: 1.1587232861668606

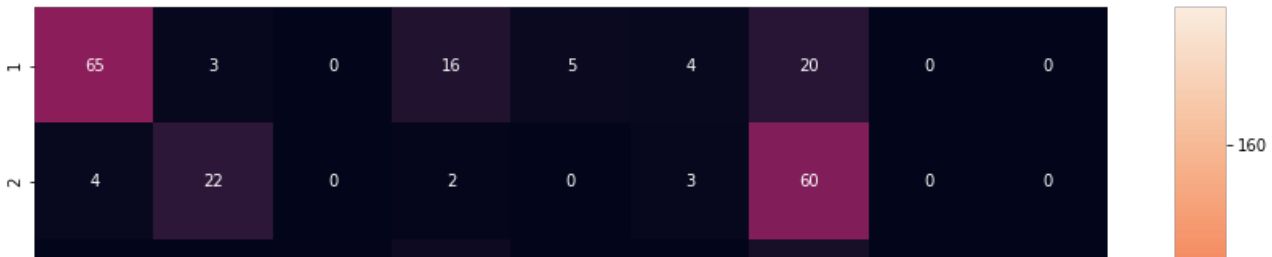
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

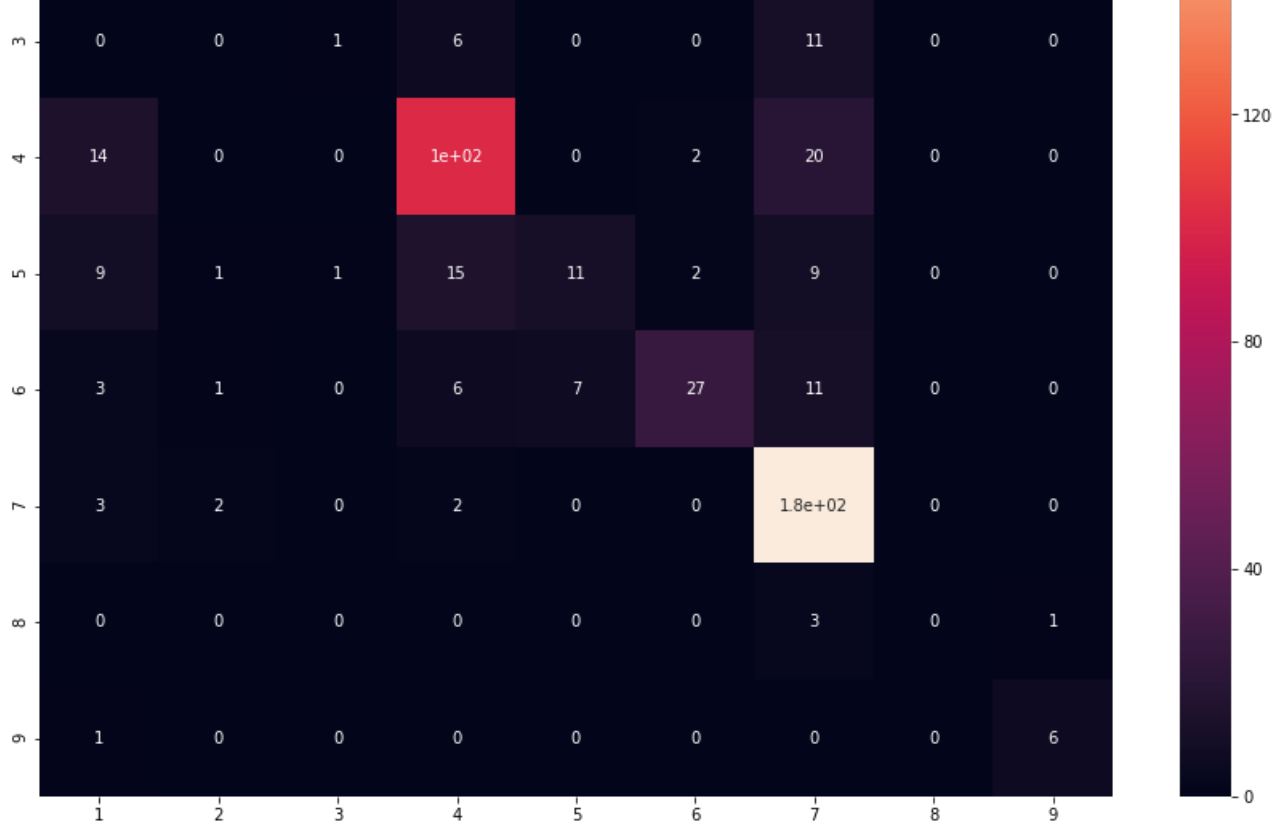
In [179]:

```
predict_and_plot_confusion_matrix(x_train_ohe, ytrain, x_test_ohe, ytest,
                                  RandomForestClassifier(n_estimators= 1000, max_depth= 8, random_state=42, n_jobs=-1,
                                                         class_weight= 'balanced'))
```

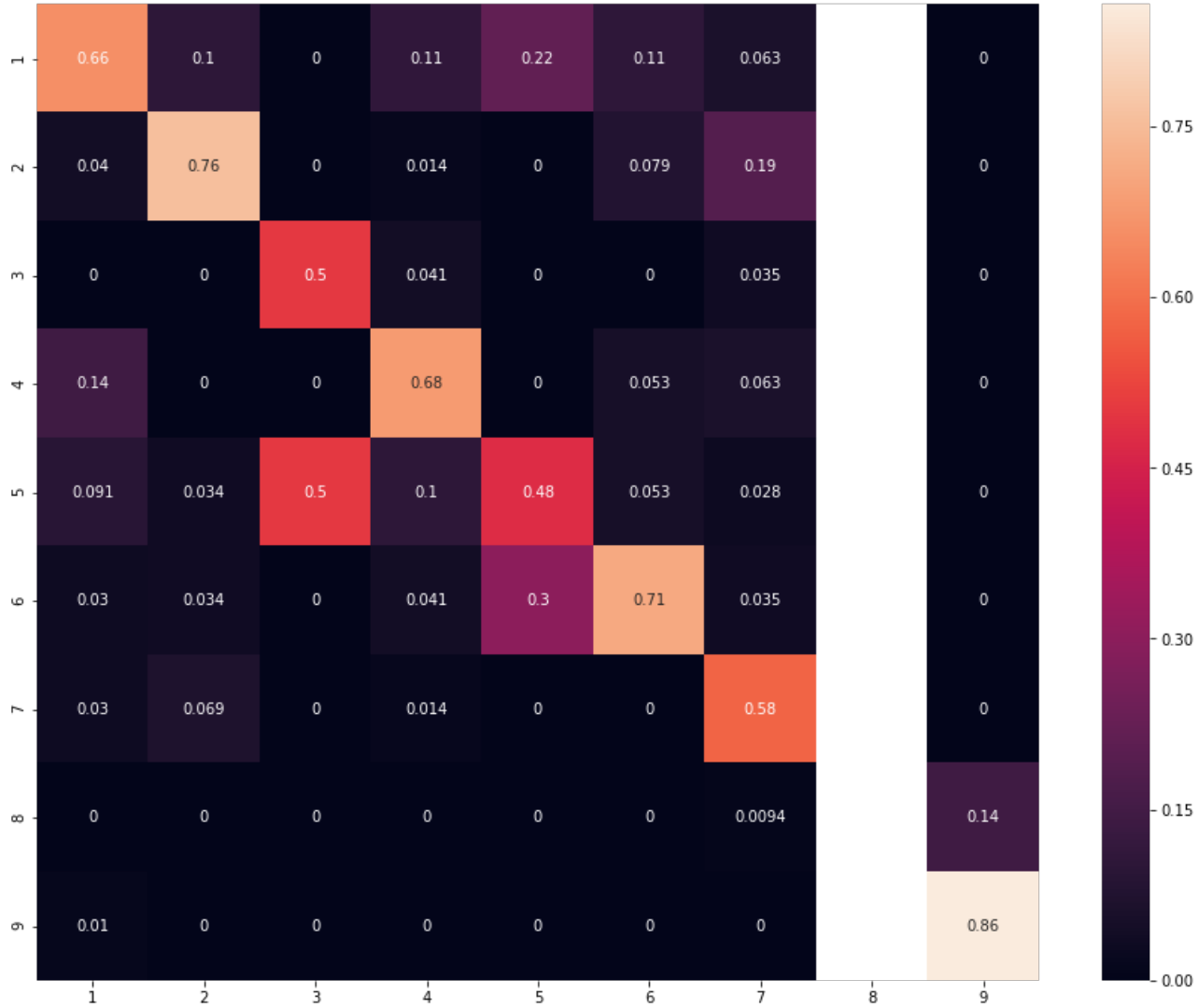
Log loss : 1.1587232861668606
Number of mis-classified points : 0.37198795180722893

********** Confusion Matrix **********

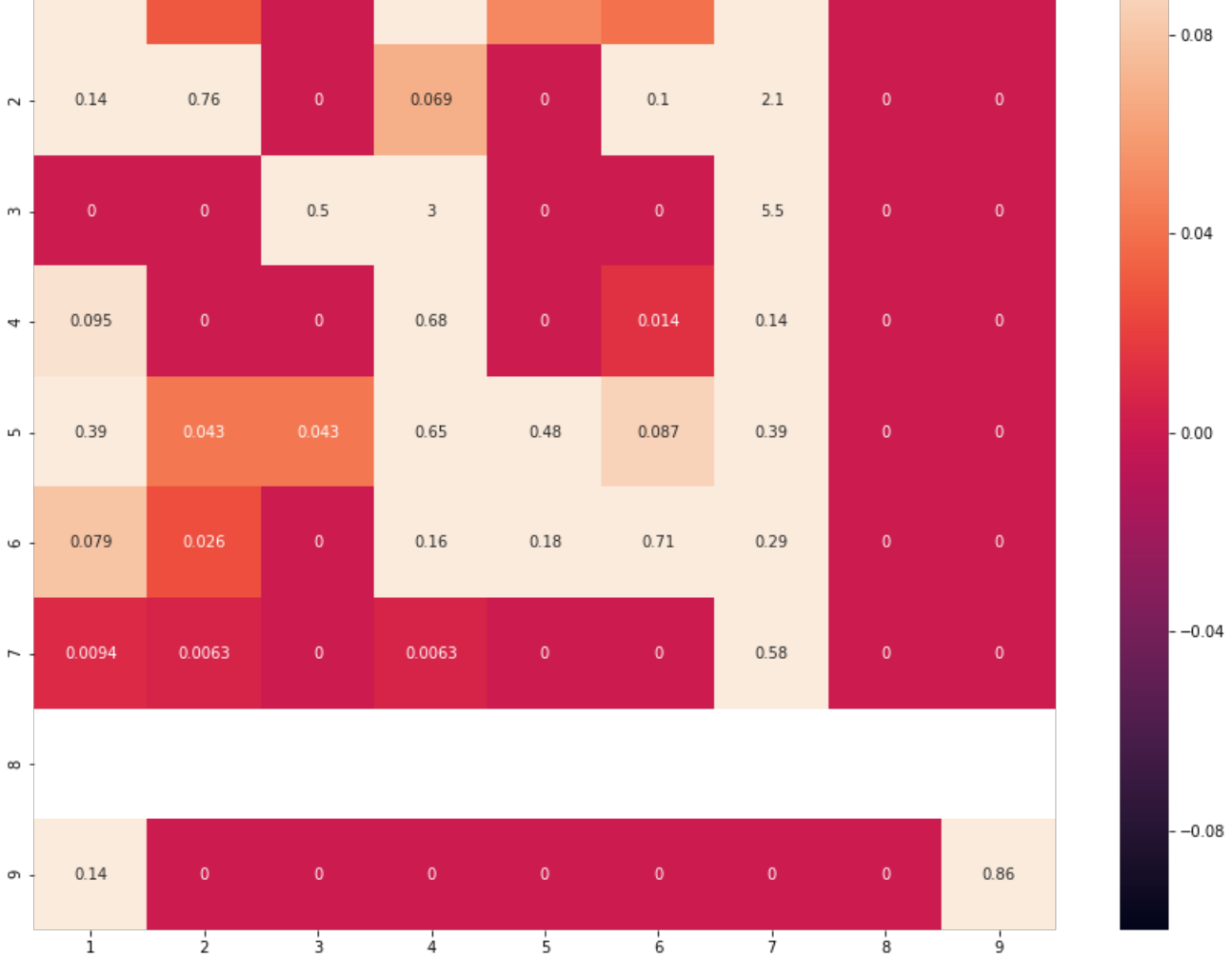




********** Precision Matrix **********



********** Recall Matrix **********



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [181]:

```
test_point_index = 1
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-rfc.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3244 0.1035 0.0185 0.1287 0.0464 0.0376 0.3195 0.0074 0.014 ]]
Actual Class : 1
--------------------------------------------------
10 Text feature [absolute] present in test data point [True]
82 Text feature [acids] present in test data point [True]
108 Text feature [acetylation] present in test data point [True]
128 Text feature [aag] present in test data point [True]
274 Text feature [achieved] present in test data point [True]
292 Text feature [abundantly] present in test data point [True]
296 Text feature [able] present in test data point [True]
322 Text feature [absent] present in test data point [True]
333 Text feature [accomplish] present in test data point [True]
359 Text feature [ablating] present in test data point [True]
392 Text feature [accession] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [182]:

```
test_point_index = 5
```

```
test_point_index = 3
no_feature = 500
predicted_cls = cccv.predict(x_test_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_ohe[test_point_index]),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-rfc.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], x_test['TEXT'].iloc[test_point_index], x_test['Gene'].iloc[test_point_index],
                     x_test['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.165  0.096  0.025  0.1524 0.0768 0.1005 0.3592 0.0101 0.0148]]
Actual Class : 6
--------------------------------------------------
82 Text feature [acids] present in test data point [True]
296 Text feature [able] present in test data point [True]
392 Text feature [accession] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [183]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.
# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/
#sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
# ----------------------------
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification

estimator= [5, 10, 50, 100, 200, 500, 1000]
max_depth= [2,3,4,5,6,7,8,9,10]
cv_log_error= []
for i in tqdm(estimator):
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        rfc= RandomForestClassifier(n_estimators= i, max_depth= j, class_weight= 'balanced', n_jobs= -1)
        rfc.fit(x_train_rc, ytrain)
        cccv= CalibratedClassifierCV(base_estimator= rfc, method= 'sigmoid')
        cccv.fit(x_train_rc, ytrain)
        y_pred= cccv.predict_proba(x_cv_rc)
        cv_log_error.append(log_loss(ycv, y_pred, labels= rfc.classes_, eps = 1e-15))
        print("The log loss is:",log_loss(ycv, y_pred, labels= rfc.classes_, eps=1e-15))

fig, ax= plt.subplots(figsize= (16, 8))
features = np.dot(np.array(estimator)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error)
for i, j in enumerate(np.round(cv_log_error, 2)):
    ax.annotate((estimator[int(i%2)],max_depth[int(i%2)],str(j)), (features[i], cv_log_error[i]))
plt.title('Cross Validation Error for each Estimator & Max-Depth')
plt.xlabel('Estimator & Max-Depth')
plt.ylabel('Error')
plt.grid()
plt.show()

best_est_md= np.argmin(cv_log_error)
# The estimator is 200 and max_depth is 8 --> are considered by best_est_md

rfc= RandomForestClassifier(n_estimators=200 , max_depth= 8,
```

```
                        random_state=42, n_jobs=-1, class_weight= 'balanced')
rfc.fit(x_train_rc, ytrain)
cccv= CalibratedClassifierCV(base_estimator= rfc, method= 'sigmoid')
cccv.fit(x_train_rc, ytrain)

y_pred= cccv.predict_proba(x_train_rc)
print('For values of best estimator: ', 200, 'and max depth:', 8,
    "The train log loss is: ", log_loss(ytrain, y_pred, labels= rfc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_cv_rc)
print('For values of best estimator: ', 200, 'and max depth:', 8,
    "The cv log loss is: ", log_loss(ycv, y_pred, labels= rfc.classes_, eps = 1e-15))
y_pred= cccv.predict_proba(x_test_rc)
print('For values of best estimator: ', 200, 'and max depth:', 8,
    "The test log loss is: ", log_loss(ytest, y_pred, labels= rfc.classes_, eps = 1e-15))
```

```
0%|          | 0/7 [00:00<?, ?it/s]

for n_estimators = 5 and max depth =  2
The log loss is: 2.570237751231094
for n_estimators = 5 and max depth =  3
The log loss is: 1.5764953316877688
for n_estimators = 5 and max depth =  4
The log loss is: 1.7390474524163846
for n_estimators = 5 and max depth =  5
The log loss is: 1.590246210712659
for n_estimators = 5 and max depth =  6
The log loss is: 1.28489513290464
for n_estimators = 5 and max depth =  7
The log loss is: 1.3929272281399132
for n_estimators = 5 and max depth =  8
The log loss is: 1.4700801583885674
for n_estimators = 5 and max depth =  9
The log loss is: 1.5734689469723584
for n_estimators = 5 and max depth =  10

14%|▉         | 1/7 [00:10<01:03, 10.58s/it]

The log loss is: 1.6389494770611337
for n_estimators = 10 and max depth =  2
The log loss is: 2.1589292437782825
for n_estimators = 10 and max depth =  3
The log loss is: 1.456136978141318
for n_estimators = 10 and max depth =  4
The log loss is: 1.5325524311132541
for n_estimators = 10 and max depth =  5
The log loss is: 1.6411553943450694
for n_estimators = 10 and max depth =  6
The log loss is: 1.480708829964889
for n_estimators = 10 and max depth =  7
The log loss is: 1.5576904983950488
for n_estimators = 10 and max depth =  8
The log loss is: 1.532481844391208
for n_estimators = 10 and max depth =  9
The log loss is: 1.623420686122857
for n_estimators = 10 and max depth =  10

29%|▉▉▉       | 2/7 [00:21<00:53, 10.61s/it]

The log loss is: 1.4532540219034045
for n_estimators = 50 and max depth =  2
The log loss is: 1.669152570592556
for n_estimators = 50 and max depth =  3
The log loss is: 1.4639639572160925
for n_estimators = 50 and max depth =  4
The log loss is: 1.483849135000074
for n_estimators = 50 and max depth =  5
The log loss is: 1.3771960015087057
for n_estimators = 50 and max depth =  6
The log loss is: 1.3553983419597315
for n_estimators = 50 and max depth =  7
The log loss is: 1.27556031948962
for n_estimators = 50 and max depth =  8
The log loss is: 1.2669286688700578
for n_estimators = 50 and max depth =  9
The log loss is: 1.3098179481396537
for n_estimators = 50 and max depth =  10

43%|▉▉▉▉▉     | 3/7 [00:36<00:47, 11.92s/it]
```

```
The log loss is: 1.348374515744617
for n_estimators = 100 and max depth =  2
The log loss is: 1.6295100662814845
for n_estimators = 100 and max depth =  3
The log loss is: 1.5479972415363537
for n_estimators = 100 and max depth =  4
The log loss is: 1.5414484454896602
for n_estimators = 100 and max depth =  5
The log loss is: 1.3365767154044572
for n_estimators = 100 and max depth =  6
The log loss is: 1.3266748329356315
for n_estimators = 100 and max depth =  7
The log loss is: 1.2871397032655487
for n_estimators = 100 and max depth =  8
The log loss is: 1.2515704802423175
for n_estimators = 100 and max depth =  9
The log loss is: 1.368829840033529
for n_estimators = 100 and max depth =  10
```

```
57%|[redacted]| 4/7 [00:55<00:42, 14.14s/it]
```

```
The log loss is: 1.4059366359139431
for n_estimators = 200 and max depth =  2
The log loss is: 1.6218228482382984
for n_estimators = 200 and max depth =  3
The log loss is: 1.5800050013302345
for n_estimators = 200 and max depth =  4
The log loss is: 1.4890439072142823
for n_estimators = 200 and max depth =  5
The log loss is: 1.390781406563427
for n_estimators = 200 and max depth =  6
The log loss is: 1.3335642368486251
for n_estimators = 200 and max depth =  7
The log loss is: 1.3021071931394235
for n_estimators = 200 and max depth =  8
The log loss is: 1.3055869950994905
for n_estimators = 200 and max depth =  9
The log loss is: 1.3191707943149227
for n_estimators = 200 and max depth =  10
```

```
71%|[redacted]| 5/7 [01:23<00:36, 18.33s/it]
```

```
The log loss is: 1.373615194045296
for n_estimators = 500 and max depth =  2
The log loss is: 1.663289084688107
for n_estimators = 500 and max depth =  3
The log loss is: 1.5644972765662923
for n_estimators = 500 and max depth =  4
The log loss is: 1.4312743205854528
for n_estimators = 500 and max depth =  5
The log loss is: 1.3672502151869412
for n_estimators = 500 and max depth =  6
The log loss is: 1.2941747694939811
for n_estimators = 500 and max depth =  7
The log loss is: 1.2776740059811562
for n_estimators = 500 and max depth =  8
The log loss is: 1.2869538785268353
for n_estimators = 500 and max depth =  9
The log loss is: 1.2966472020975057
for n_estimators = 500 and max depth =  10
```

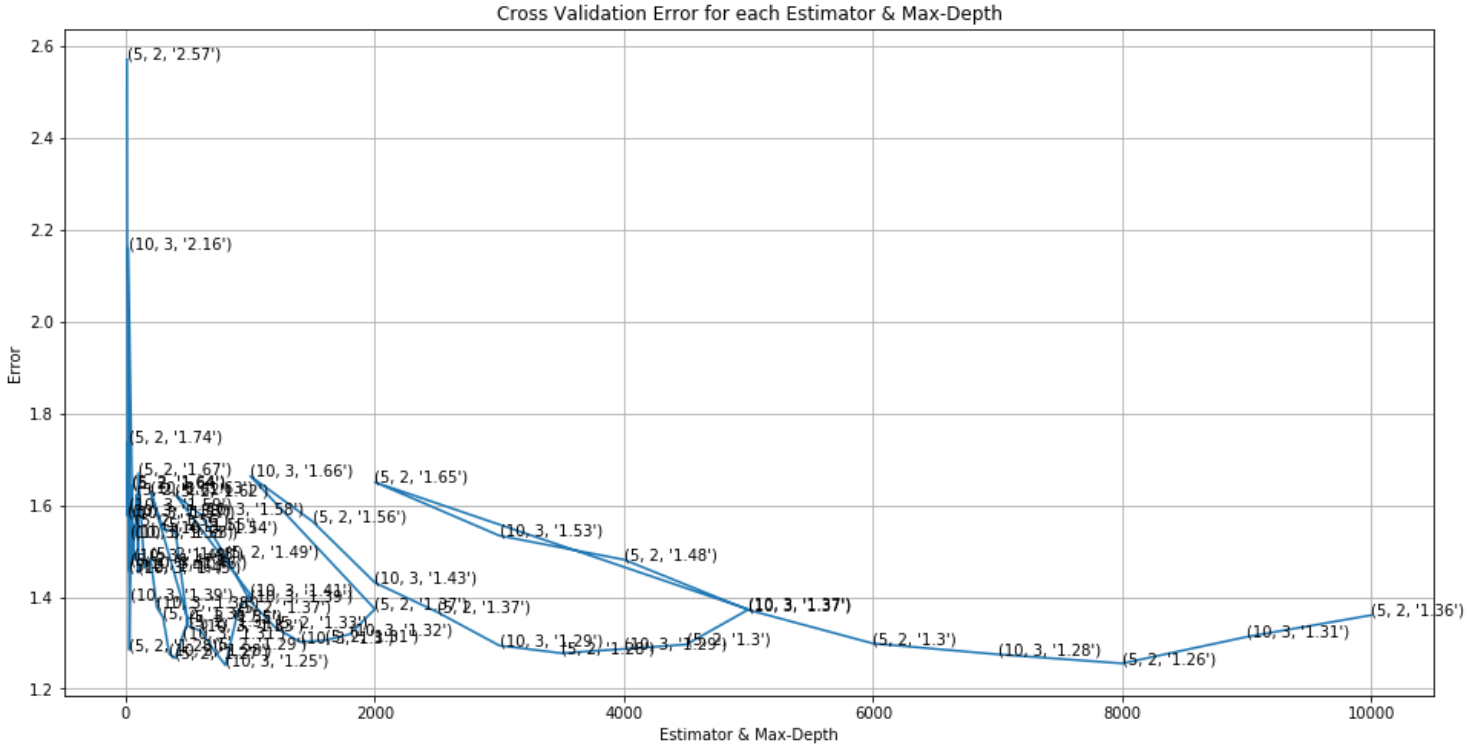
```
86%|[redacted]| 6/7 [02:32<00:33, 33.55s/it]
```

```
The log loss is: 1.3729183259687439
for n_estimators = 1000 and max depth =  2
The log loss is: 1.6499914697257743
for n_estimators = 1000 and max depth =  3
The log loss is: 1.5334950791183457
for n_estimators = 1000 and max depth =  4
The log loss is: 1.4812768486316281
for n_estimators = 1000 and max depth =  5
The log loss is: 1.3717187093231626
for n_estimators = 1000 and max depth =  6
The log loss is: 1.298781206552123
for n_estimators = 1000 and max depth =  7
The log loss is: 1.2751339205375567
for n_estimators = 1000 and max depth =  8
```

The log loss is: 1.2554778238823352
for n_estimators = 1000 and max depth = 9
The log loss is: 1.313291360964601
for n_estimators = 1000 and max depth = 10

100%|[redacted]| 7/7 [04:48<00:00, 41.24s/it]

The log loss is: 1.360477697387301



For values of best estimator: 200 and max depth: 8 The train log loss is: 0.033930165949889755
For values of best estimator: 200 and max depth: 8 The cv log loss is: 1.3017712074997825
For values of best estimator: 200 and max depth: 8 The test log loss is: 1.284718032055703

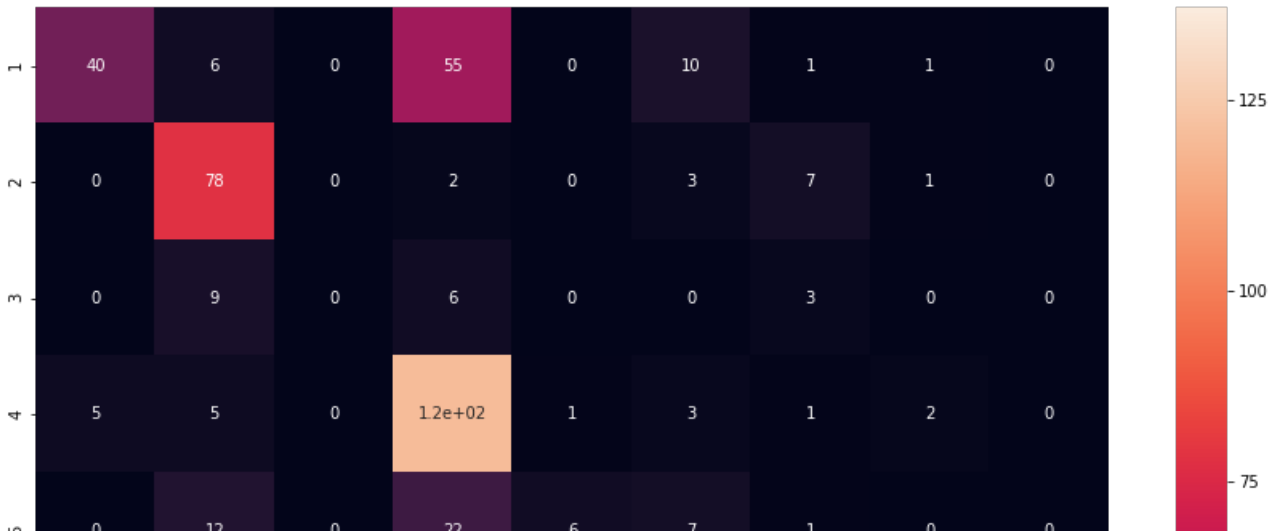
4.5.4. Testing model with best hyper parameters (Response Coding)

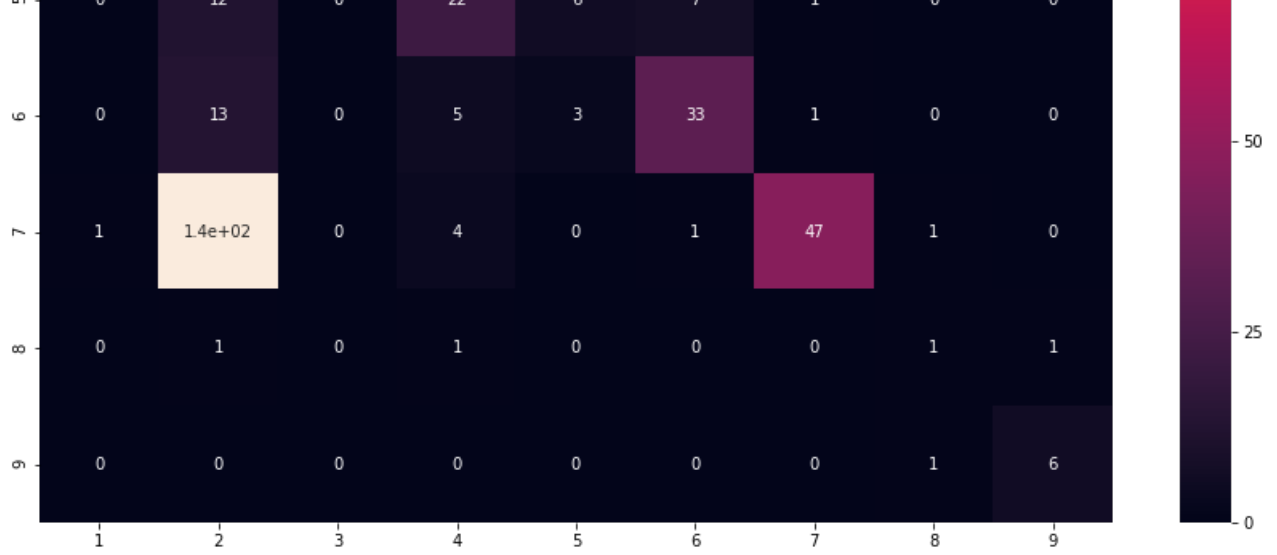
In [184]:

```
predict_and_plot_confusion_matrix(x_train_rc, ytrain, x_test_rc, ytest,
                                  RandomForestClassifier(n_estimators= 200, max_depth= 8, random_state=42, n_jobs=-1,
                                                         class_weight= 'balanced'))
```

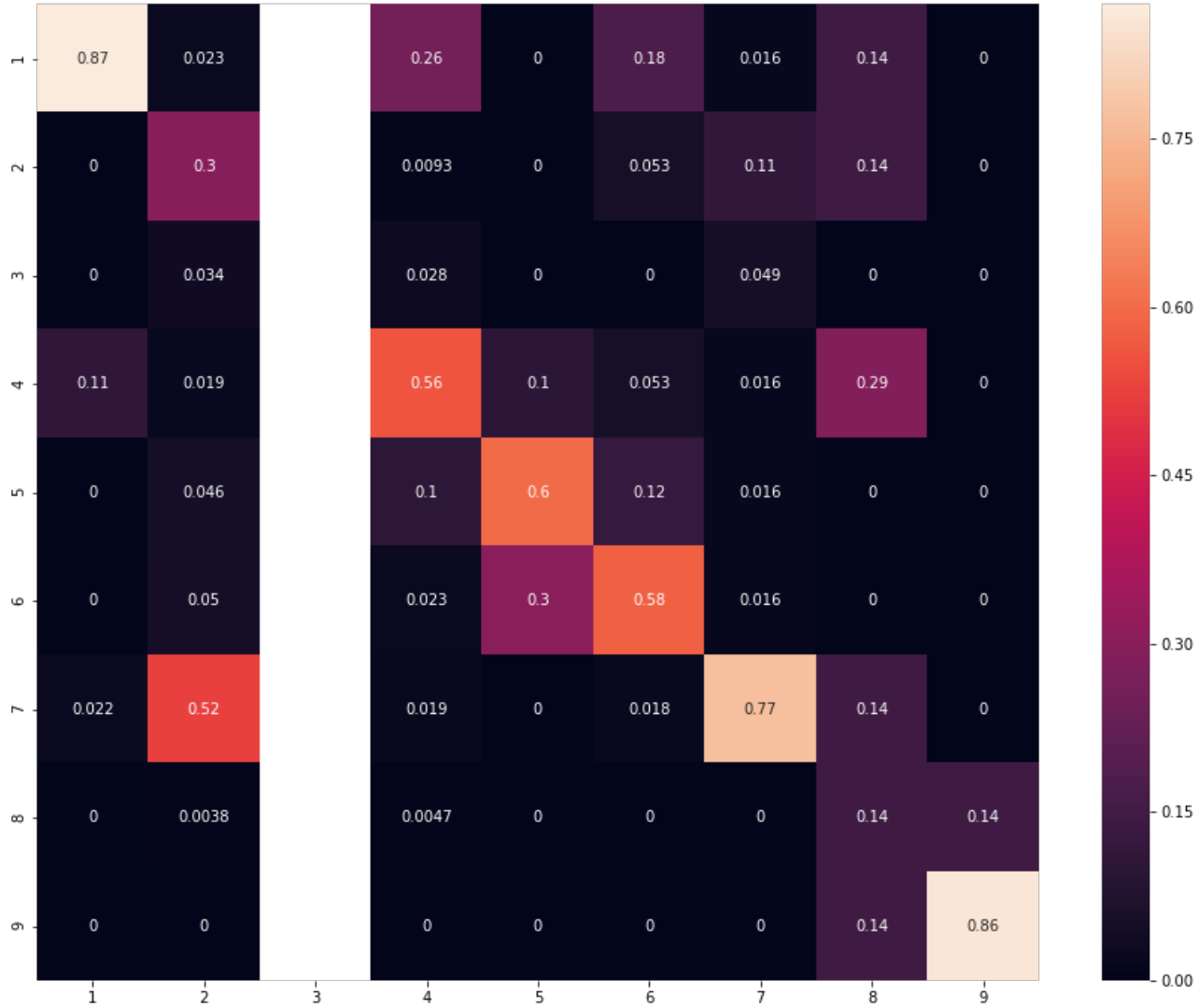
Log loss : 1.284718032055703
Number of mis-classified points : 0.5015060240963856

********** Confusion Matrix **********

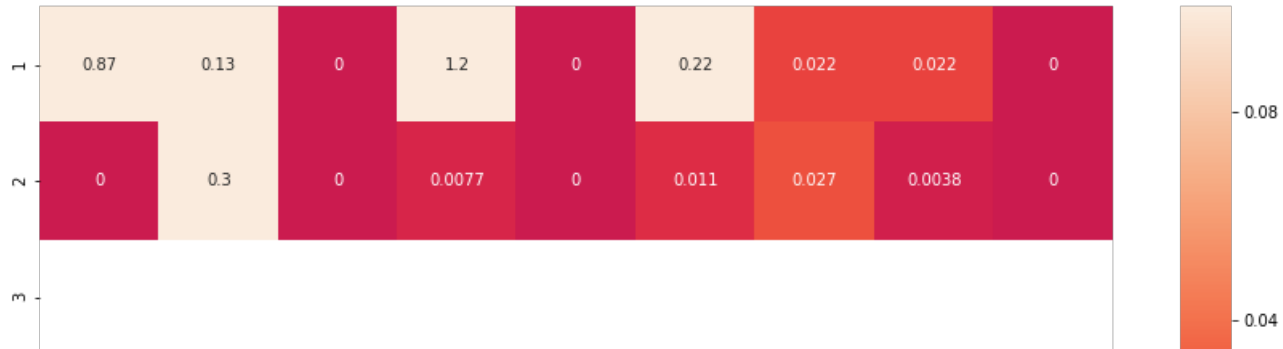


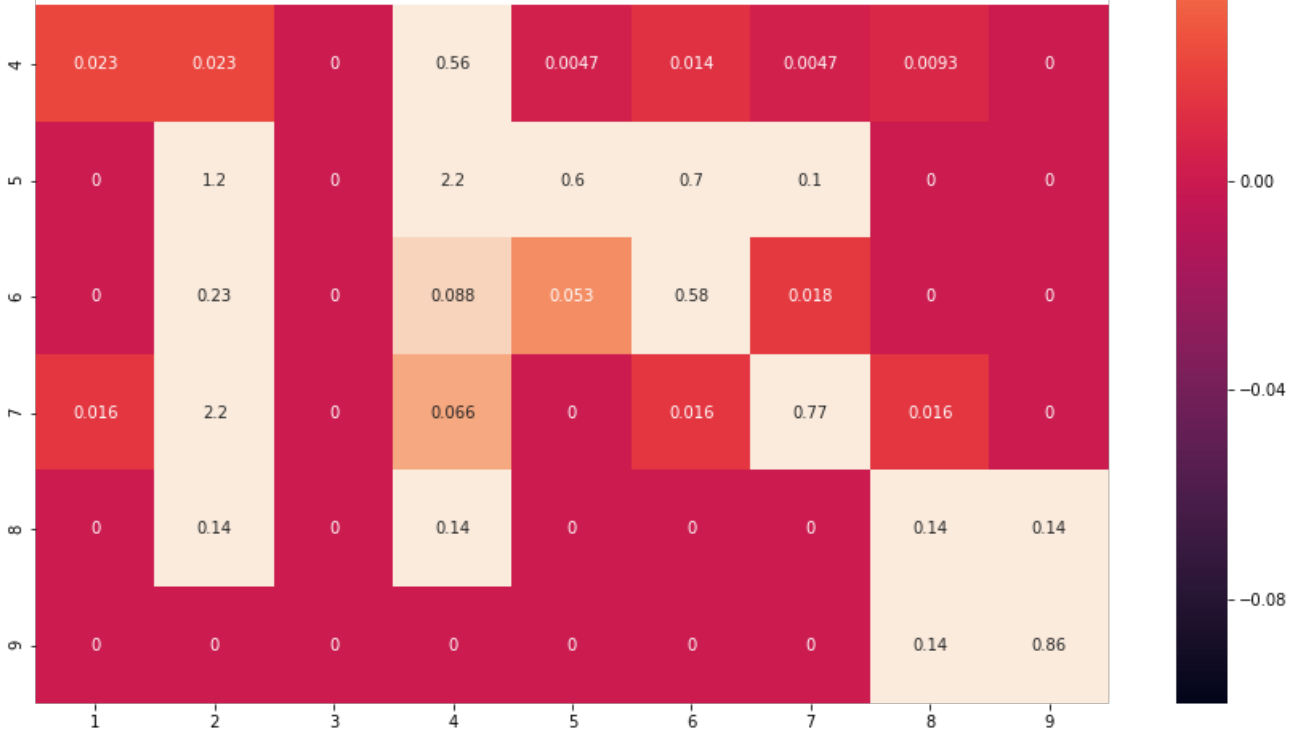


********** Precision Matrix **********



********** Recall Matrix **********





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [188]:

```
test_point_index = 1
no_feature = 500
predicted_cls = cccv.predict(x_test_rc[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_rc[test_point_index].reshape(1,-1)),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-rfc.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.9536 0.0022 0.0013 0.0325 0.0013 0.0035 0.002  0.0018 0.0019]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

```
Gene is important feature
Gene is important feature
```

4.5.3.2. Inorrectly Classified point

In [191]:

```
test_point_index = 4
no_feature = 500
predicted_cls = cccv.predict(x_test_rc[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(cccv.predict_proba(x_test_rc[test_point_index].reshape(1,-1)),4))
print("Actual Class :", ytest[test_point_index])
indices = np.argsort(-rfc.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0449 0.3946 0.0231 0.0912 0.041  0.0786 0.1985 0.1062 0.0218]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [194]:

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

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#------------------------------
```

```
# read more about support vector machines with linear kernals here
# http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# -------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.


# read more about support vector machines with linear kernals here
# http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).


# Logistic Regression
clf1= SGDClassifier(alpha= 0.0001, penalty= 'l2', loss= 'log', class_weight= 'balanced', random_state= 0)
clf1.fit(x_train_ohe, ytrain)
sig_clf1= CalibratedClassifierCV(clf1, method= "sigmoid")

# Linear Support Vector Machine
clf2= SGDClassifier(alpha= 0.0001, penalty= 'l2', loss= 'hinge', class_weight= 'balanced', random_state= 0)
clf2.fit(x_train_ohe, ytrain)
sig_clf2= CalibratedClassifierCV(clf2, method= "sigmoid")

# Multinomial Naive Bayes
clf3= MultinomialNB(alpha= 0.00001)
clf3.fit(x_train_ohe, ytrain)
sig_clf3= CalibratedClassifierCV(clf3, method= "sigmoid")

sig_clf1.fit(x_train_ohe, ytrain)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(ycv, sig_clf1.predict_proba(x_cv_ohe))))
sig_clf2.fit(x_train_ohe, ytrain)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(ycv, sig_clf2.predict_proba(x_cv_ohe))))
sig_clf3.fit(x_train_ohe, ytrain)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(ycv, sig_clf3.predict_proba(x_cv_ohe))))
print("-"*50)

alpha= [0.0001,0.001,0.01,0.1,1,10]
best_alpha= 999
for i in alpha:
    # Logistic Regression
    lr= LogisticRegression(C=i)
    sclf= StackingClassifier(classifiers= [sig_clf1, sig_clf2, sig_clf3], meta_classifier= lr, use_probas= True)
    sclf.fit(x_train_ohe, ytrain)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(ycv, sclf.predict_proba(x_cv_ohe))))
    log_error =log_loss(ycv, sclf.predict_proba(x_cv_ohe))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.06
Naive Bayes : Log Loss: 1.30
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.172
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.979
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.368
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.108
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.370
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.930
```

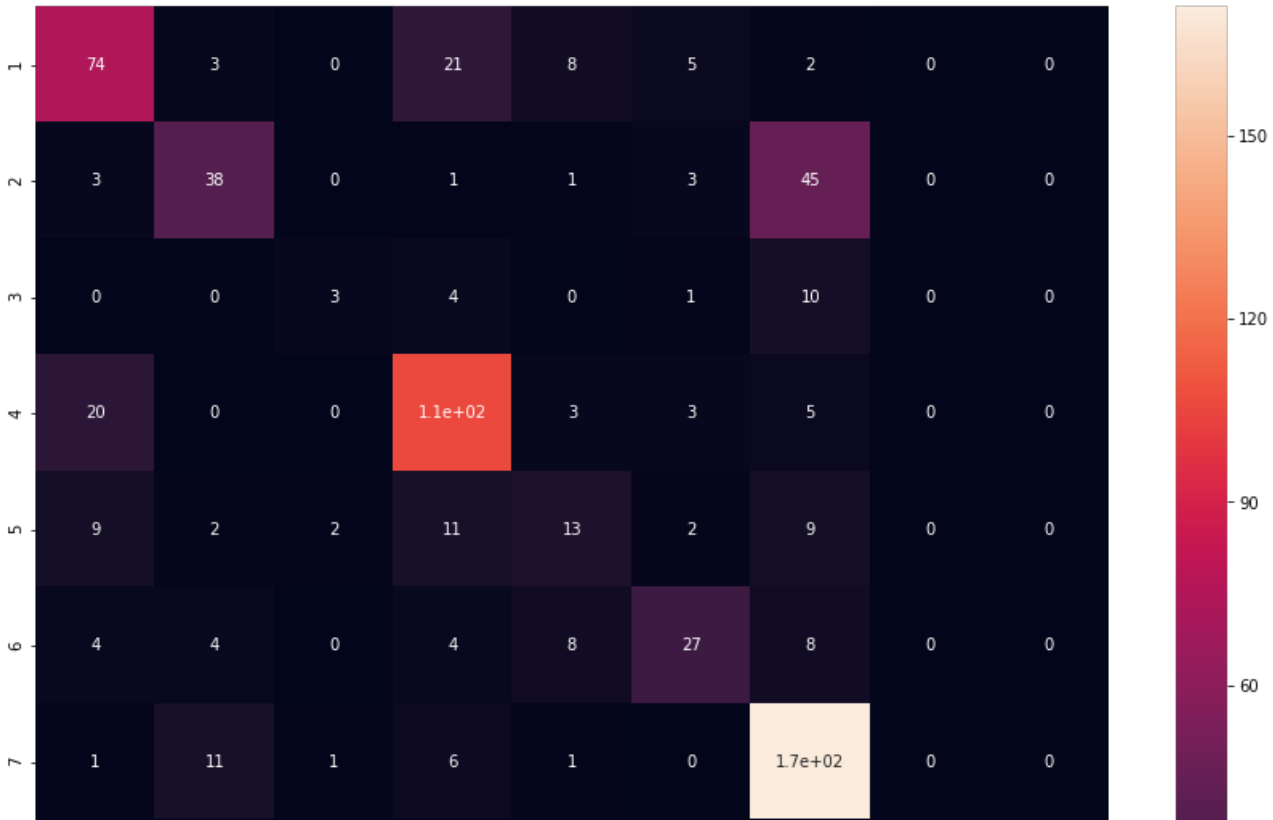
In [202]:

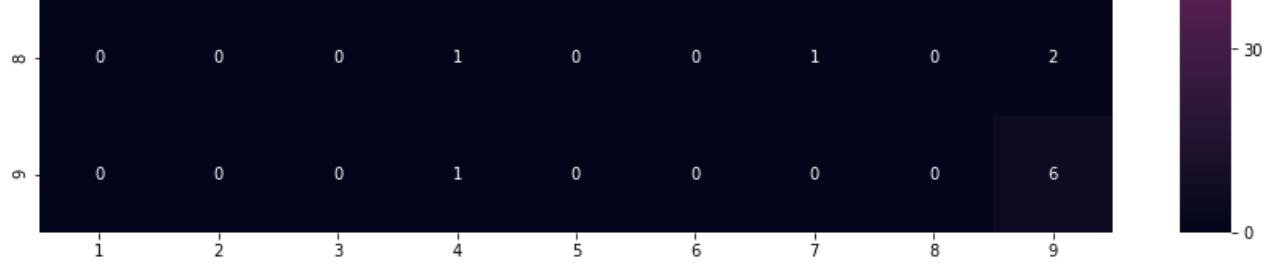
```
# Confusion Matrix for Stacking Classifier

def predict_and_plot_confusion_matrix_sc(x_train_ohe, ytrain, x_test_ohe, ytest, clf):
    clf.fit(x_train_ohe, ytrain)
    y_pred = clf.predict(x_test_ohe)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(ytest, clf.predict_proba(x_test_ohe)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((y_pred - ytest))/ytest.shape[0])
    print()
    print('*'*35 +' Confusion Matrix ' + '*'*35)
    c_m = confusion_matrix(ytest, y_pred)
    heatmap(c_m)
    # Precision
    print('*'*35 +' Precision Matrix ' + '*'*35)
    precision = c_m / c_m.sum(axis=0)
    heatmap(precision)
    # Recall
    print('*'*35 +' Recall Matrix ' + '*'*35)
    recall = (c_m.T / c_m.sum(axis=0)).T
    heatmap(recall)
```

In [203]:

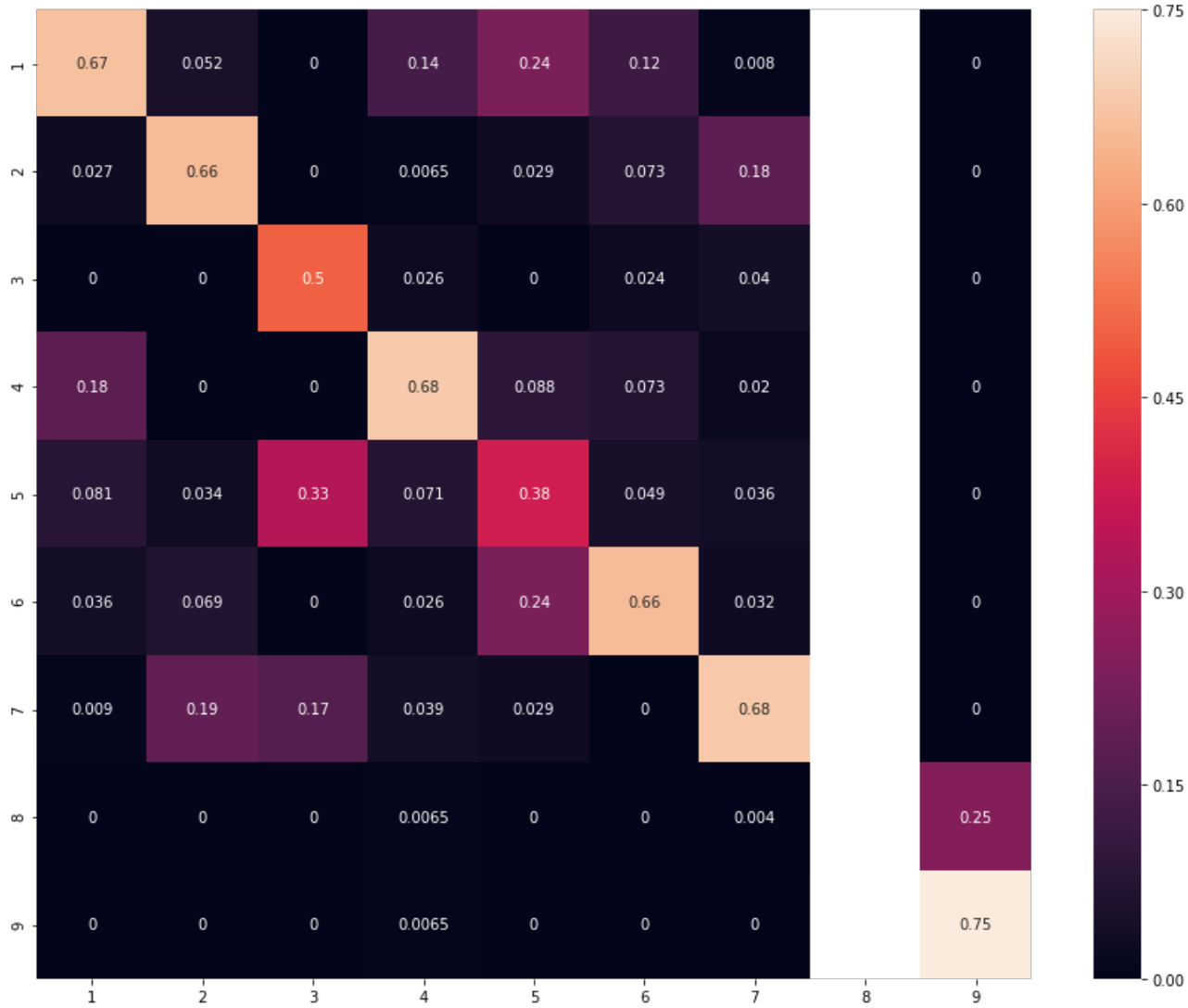
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(x_train_ohe, ytrain)

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

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

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

predict_and_plot_confusion_matrix_sc(x_train_ohe, ytrain, x_test_ohe, ytest, StackingClassifier(classifiers= [sig_clf1, sig_clf2, sig_clf3],
                                                                                                meta_classifier=lr, use_probas=True))
```

```
Log loss (train) on the stacking classifier : 0.3453982162708228
Log loss (CV) on the stacking classifier : 1.1084209013583475
Log loss (test) on the stacking classifier : 1.072860528855775
Log loss : 1.072860528855775
Number of mis-classified points : 0.34036144578313254

********************************** Confusion Matrix **********************************
```

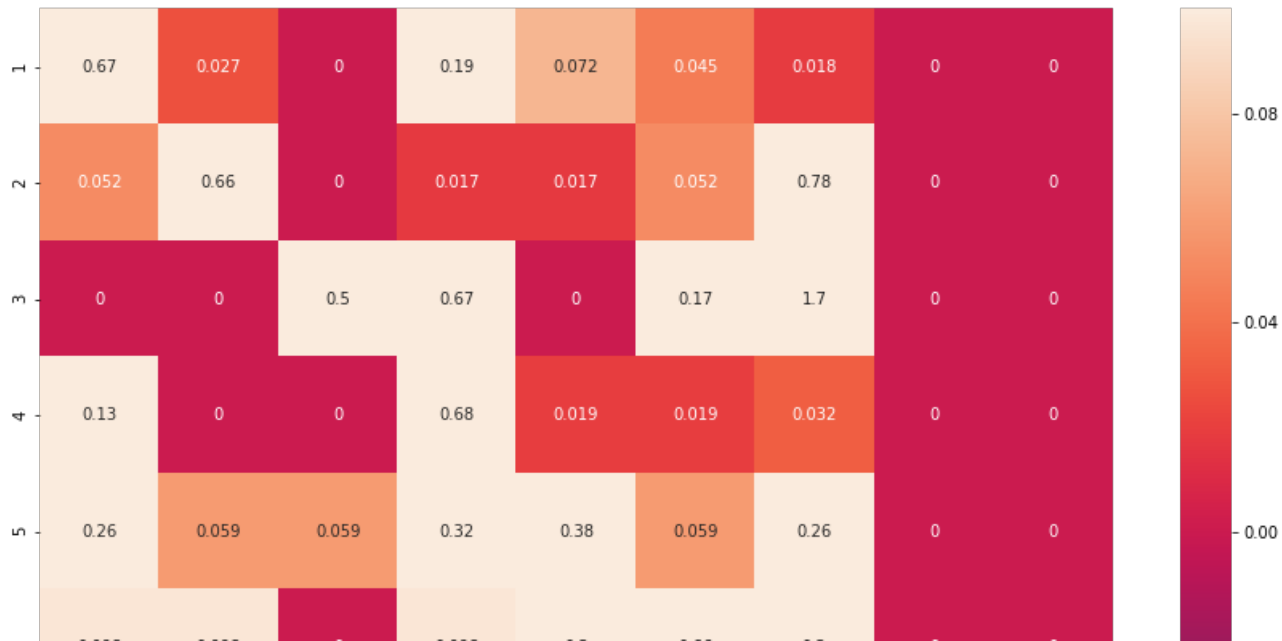


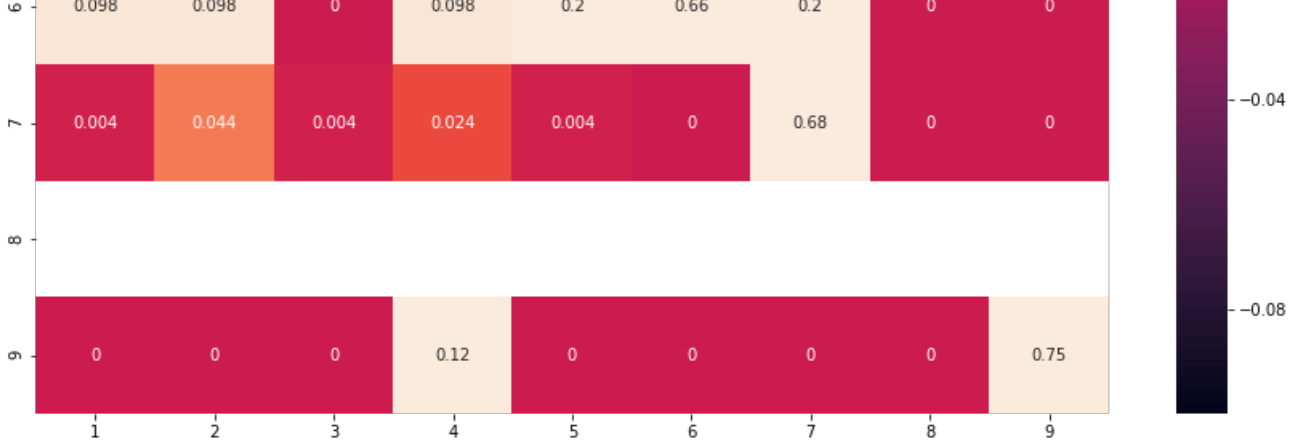


********** Precision Matrix **********



********** Recall Matrix **********





4.7.3 Maximum Voting classifier

In [204]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier

clf1= SGDClassifier(loss= 'log', penalty= 'l2', alpha= 0.0001, random_state= 42, class_weight= 'balanced')
clf1.fit(x_train_ohe, ytrain)
sig_clf1 = CalibratedClassifierCV(clf1, method= "sigmoid")

clf2= SGDClassifier(loss= 'hinge', penalty= 'l2', alpha= 0.0001, random_state= 42, class_weight= 'balanced')
clf2.fit(x_train_ohe, ytrain)
sig_clf2 = CalibratedClassifierCV(clf2, method= "sigmoid")

clf3= RandomForestClassifier(n_estimators= 1000, max_depth= 8, random_state=42, n_jobs=-1,
                             class_weight= 'balanced')
clf3.fit(x_train_ohe, ytrain)
sig_clf3 = CalibratedClassifierCV(clf3, method= "sigmoid")

vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(x_train_ohe, ytrain)
print("Log loss (train) on the VotingClassifier :", log_loss(ytrain, vclf.predict_proba(x_train_ohe)))
print("Log loss (CV) on the VotingClassifier :", log_loss(ycv, vclf.predict_proba(x_cv_ohe)))
print("Log loss (test) on the VotingClassifier :", log_loss(ytest, vclf.predict_proba(x_test_ohe)))
```

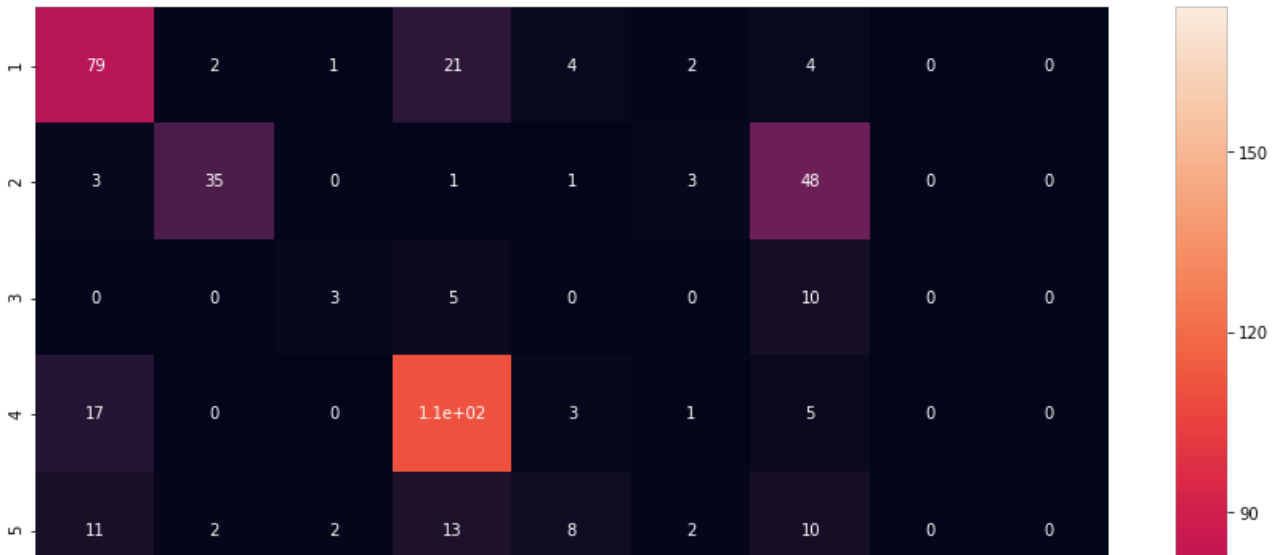
Log loss (train) on the VotingClassifier : 0.48588942816525466
Log loss (CV) on the VotingClassifier : 1.0105732613596352
Log loss (test) on the VotingClassifier : 0.971637810694532

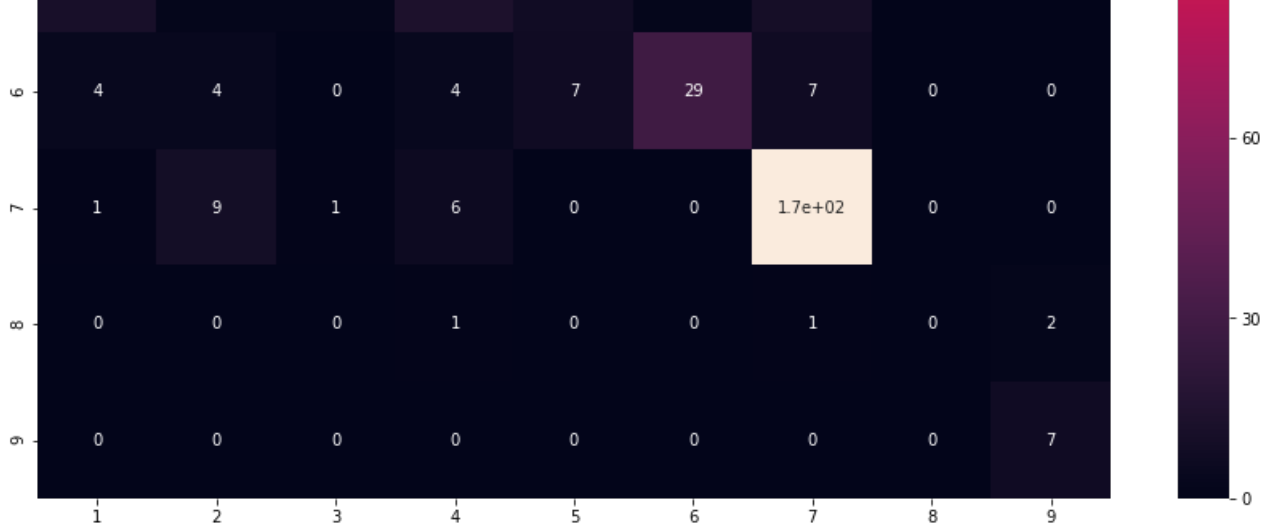
In [205]:

```
predict_and_plot_confusion_matrix_sc(x_train_ohe, ytrain, x_test_ohe, ytest, vclf)
```

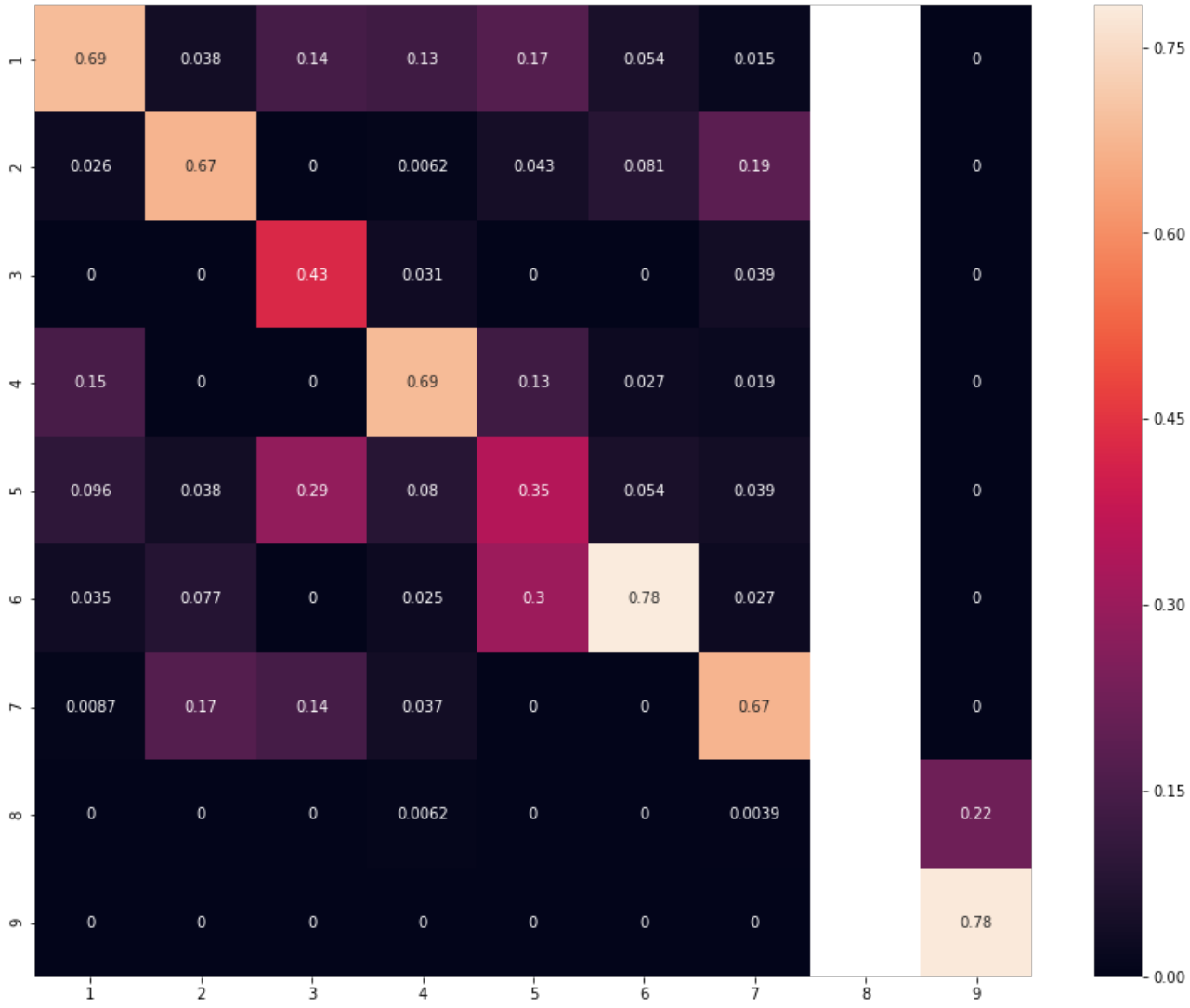
Log loss : 0.971637810694532
Number of mis-classified points : 0.32831325301204817

********** Confusion Matrix **********

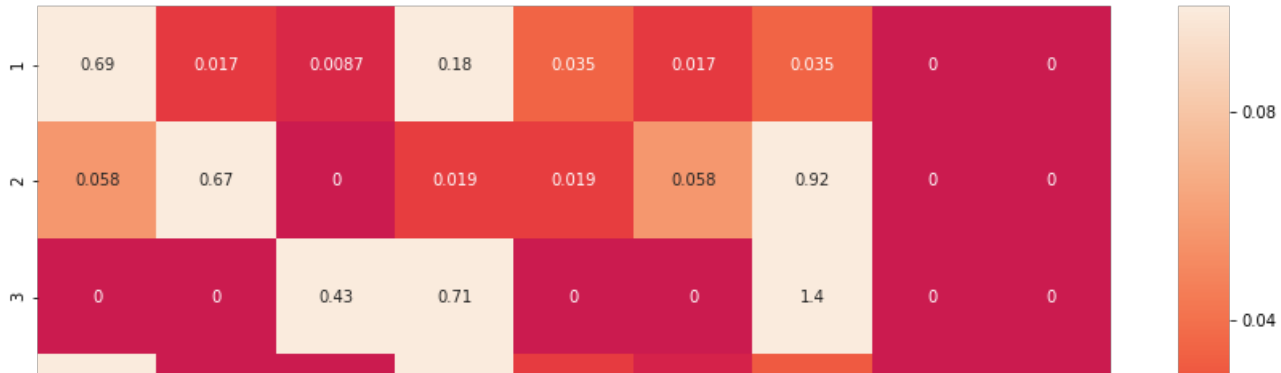


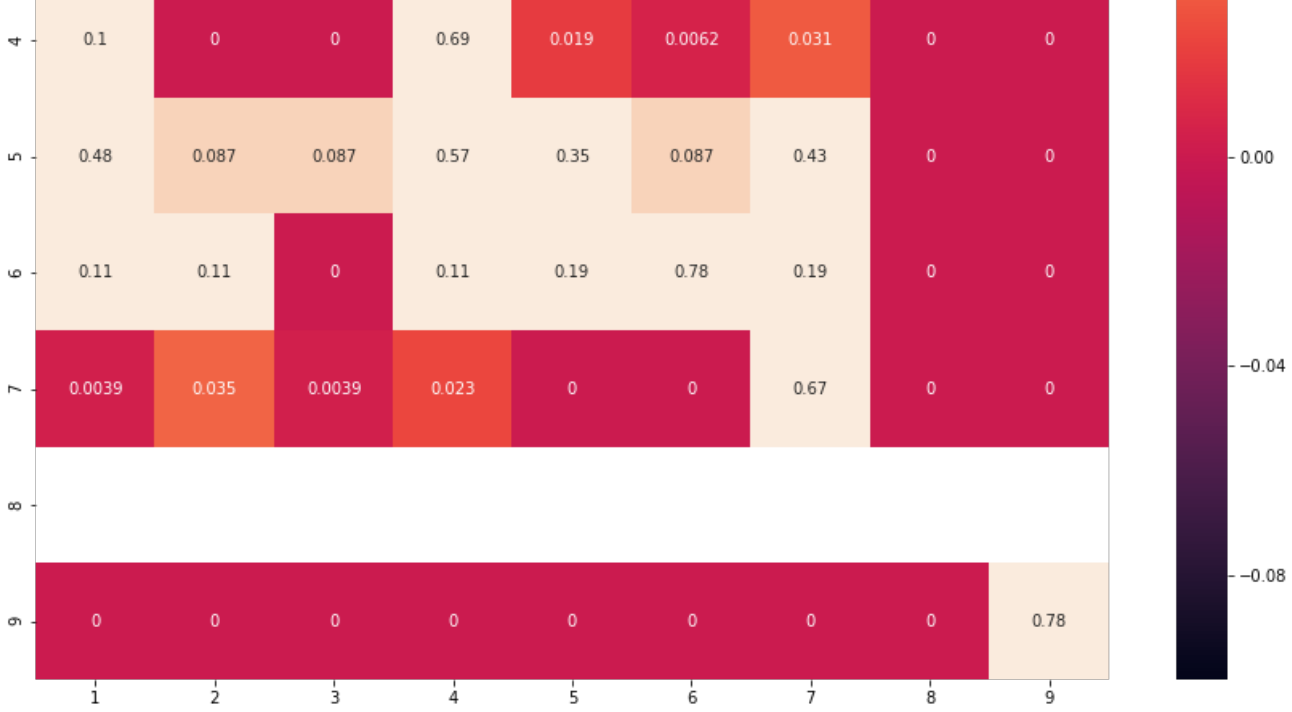


********** Precision Matrix **********



********** Recall Matrix **********





Conclusions

In [213]:

```
print("Below are the detailed conclusions about various Classifiers\n")
x = PrettyTable()

x.field_names = ["Rank", "Classifier", "Train", "CV", "Test", "MisClassified %", "Hyperparameters", "Comments"]
x.add_row([1, "Logistic Regression", 0.42, 1.05, 0.96, 0.32, "Alpha= 0.0001", "B, I, OHE"])
x.add_row([2, "Logistic Regression", 0.42, 1.09, 0.99, 0.33, "Alpha= 0.0001", "IB, I, OHE"])
x.add_row([3, "Voting Classifier", 0.49, 1.01, 0.97 , 0.33, "LR, Linear SVM, RF", "B, NI, OHE"])
x.add_row([4, "Linear SVM", 0.42, 1.06, 1.00, 0.33, "Alpha= 0.0001", "B, I, OHE"])
x.add_row([5, "Stacking Classifier", 0.34, 1.10, 1.07, 0.34, "LR, Linear SVM, MNB", "IB, NI, OHE"])
x.add_row([6, "K Nearest Neighbors", 0.84, 1.13, 1.08, 0.38, "K= 41", "IB, NI, RC"])
x.add_row([7, "Random Forest", 0.68, 1.23, 1.16, 0.38, "E= 1000, Max-Depth= 8", "B, I, OHE"])
x.add_row([8, "Multinomial Naive Bayes", 0.66, 1.30, 1.21, 0.38, "Alpha= 0.00001", "IB, I, OHE"])
x.add_row([9, "Random Forest", 0.03, 1.30, 1.28, 0.50, "E= 200, Max-Depth= 8", "B, NI, RC"])

print(x)
print()
print('B - Balanced')
print('E - Estimator')
print('IB - Imbalanced')
print('OHE - OneHotEncoding')
print('RC - Response Coding')
print('NI - Not Interpretable')
print('LR - Logistic Regression')
print('RF - Random Forest')
print('MNB - Multinomial Naive Bayes')
```

Below are the detailed conclusions about various Classifiers

```
+------+-----------------------+-------+------+------+----------------+---------------------+-------------+
| Rank |       Classifier      | Train |  CV  | Test | MisClassified % |   Hyperparameters   |   Comments  |
+------+-----------------------+-------+------+------+----------------+---------------------+-------------+
|  1   |  Logistic Regression  | 0.42 | 1.05 | 0.96 |       0.32      |    Alpha= 0.0001    |  B, I, OHE  |
|  2   |  Logistic Regression  | 0.42 | 1.09 | 0.99 |       0.33      |    Alpha= 0.0001    |  IB, I, OHE |
|  3   |   Voting Classifier   | 0.49 | 1.01 | 0.97 |       0.33      |  LR, Linear SVM, RF |  B, NI, OHE |
|  4   |       Linear SVM      | 0.42 | 1.06 | 1.0  |       0.33      |    Alpha= 0.0001    |  B, I, OHE  |
|  5   |  Stacking Classifier  | 0.34 | 1.1  | 1.07 |       0.34      | LR, Linear SVM, MNB | IB, NI, OHE |
|  6   |  K Nearest Neighbors  | 0.84 | 1.13 | 1.08 |       0.38      |        K= 41        |  IB, NI, RC |
|  7   |     Random Forest     | 0.68 | 1.23 | 1.16 |       0.38      | E= 1000, Max-Depth= 8 |  B, I, OHE  |
|  8   | Multinomial Naive Bayes | 0.66 | 1.3  | 1.21 |       0.38      |    Alpha= 0.00001   |  IB, I, OHE |
|  9   |     Random Forest     | 0.03 | 1.3  | 1.28 |       0.5       | E= 200, Max-Depth= 8 |  B, NI, RC  |
+------+-----------------------+-------+------+------+----------------+---------------------+-------------+

B - Balanced
E - Estimator
IB - Imbalanced
OHE - OneHotEncoding
RC - Response Coding
NI - Not Interpretable
```

NF - Not Interpretable
LR - Logistic Regression
RF - Random Forest
MNB - Multinomial Naive Bayes

In [ ]: