3.6 Featurizing text data with tfidf weighted word-vectors

In [0]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy
```

In [0]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC

from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier



from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
```

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

In [3]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:

```
# avoid decoding problems
df = pd.read_csv("gdrive/My Drive/Quora/train.csv")
```

In [0]:

```
# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
# ----------------- python 3 ---------------------
```

```
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
```

In [6]:

```
df.head()
```

Out[6]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df['question1']) + list(df['question2'])
```

In [0]:

```
df_3 = pd.DataFrame({'question1':list(df['question1']), 'question2':list(df['question2'])})
```

In [9]:

```
df_3.head()
```

Out[9]:

	question1	question2
0	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...
1	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...
2	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...
3	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...
4	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?

In [0]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('gdrive/My Drive/Quora/nlp_features_train.csv'):
    dfnlp = pd.read_csv("gdrive/My Drive/Quora/nlp_features_train.csv",encoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous notebook")

if os.path.isfile('gdrive/My Drive/Quora/df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("gdrive/My Drive/Quora/df_fe_without_preprocessing_train.csv",encoding='la
tin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [0]:

```
y_true = dfppro['is_duplicate']
```

In [0]:

```
df1 = dfnlp.drop(['id','qid1','qid2','question1','question2','is_duplicate'],axis=1)
df2 = dfppro.drop(['id','qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [13]:

```
df1.head()
```

Out[13]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	token_
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	100
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	86
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	66
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	36
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	67

In [14]:

```
df2.head()
```

Out[14]:

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	freq
0	1	1	66	57	14	12	10.0	23.0	0.434783	2	0
1	4	1	51	88	8	13	4.0	20.0	0.200000	5	3
2	1	1	73	59	14	10	4.0	24.0	0.166667	2	0
3	1	1	50	65	11	9	0.0	19.0	0.000000	2	0
4	3	1	76	39	13	7	2.0	20.0	0.100000	4	2

df_1_2 consists of all numerical features before TfidfVectorizations

In [0]:

```
df_1_2 = hstack((df1,df2))
```

In [0]:

```
df_1_2_df = pd.DataFrame(df_1_2.toarray())
```

df_3 consists of rawtext of question_1 and question_2

In [17]:

```
df_3.columns
```

Out[17]:

```
Index(['question1', 'question2'], dtype='object')
```

In [18]:

```
df_1_2_df.head()
```

Out[18]:

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	...	1.0	66.0	57.0	14.0	12.0	10.0	23.0	0.43
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	...	1.0	51.0	88.0	8.0	13.0	4.0	20.0	0.20
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	...	1.0	73.0	59.0	14.0	10.0	4.0	24.0	0.16
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	...	1.0	50.0	65.0	11.0	9.0	0.0	19.0	0.00
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	...	1.0	76.0	39.0	13.0	7.0	2.0	20.0	0.10

5 rows × 26 columns

dataset consists of all the features with raw text and numerical features

In [0]:

```
dataset = pd.concat([df_1_2_df,df_3],axis=1)
```

In [20]:

```
dataset.head()
```

Out[20]:

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	...	57.0	14.0	12.0	10.0	23.0	0.434783	2.0
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	...	88.0	8.0	13.0	4.0	20.0	0.200000	5.0
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	...	59.0	14.0	10.0	4.0	24.0	0.166667	2.0
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	...	65.0	11.0	9.0	0.0	19.0	0.000000	2.0
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	...	39.0	13.0	7.0	2.0	20.0	0.100000	4.0

5 rows × 28 columns

Train_test_split of dataset

In [0]:

```
X_train,X_test, y_train, y_test = train_test_split(dataset, y_true, stratify=y_true, test_size=0.3,
random_state=0)
```

In [0]:

```
X_train_1 =  X_train[:]['question1']
X_test_1 = X_test[:]['question1']
```

Tfidf Vectorizations

In [23]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=5)
train_tfidf_1 = vectorizer.fit_transform(X_train_1)
test_tfidf_1 = vectorizer.transform(X_test_1)

print(train_tfidf_1.shape,test_tfidf_1.shape)
```

```
(283003, 18471) (121287, 18471)
```

In [0]:

```
X_train_2 =  X_train[:]['question2']
X_test_2 = X_test[:]['question2']
```

In [25]:

```
vectorizer3 = TfidfVectorizer(min_df=5)
train_tfidf_2 = vectorizer3.fit_transform(X_train_2)
test_tfidf_2 = vectorizer3.transform(X_test_2)

print(train_tfidf_2.shape,test_tfidf_2.shape)
```

```
(283003, 17586) (121287, 17586)
```

In [26]:

```
import scipy
X_train_cn = X_train.drop(['question1','question2'],axis=1)
X_train_cn = scipy.sparse.csr_matrix(X_train_cn)
print(X_train_cn.shape)

X_test_cn =  X_test.drop(['question1','question2'],axis=1)
X_test_cn = scipy.sparse.csr_matrix(X_test_cn)
print(X_test_cn.shape)
```

```
(283003, 26)
(121287, 26)
```

concatenating all features TfidfVectorizations and Numerical Features

In [0]:

```
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler

X_train_tfidf = hstack((X_train_cn,train_tfidf_1,train_tfidf_2))
X_train_tfidf = X_train_tfidf.tocsr()
```

```
train_scalar = StandardScaler(with_mean = False)
X_train_tfidf = train_scalar.fit_transform(X_train_tfidf)
```

In [0]:

```
X_test_tfidf =hstack((X_test_cn,test_tfidf_1,test_tfidf_2))
X_test_tfidf = X_test_tfidf.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_tfidf = test_scalar.fit_transform(X_test_tfidf)
```

In [29]:

```
print("Number of data points in train data :",X_train_tfidf.shape)
print("Number of data points in test data :",X_test_tfidf.shape)
```

```
Number of data points in train data : (283003, 36083)
Number of data points in test data : (121287, 36083)
```

In [30]:

```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len)
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in train data ----------
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639
```

In [0]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))

    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
```

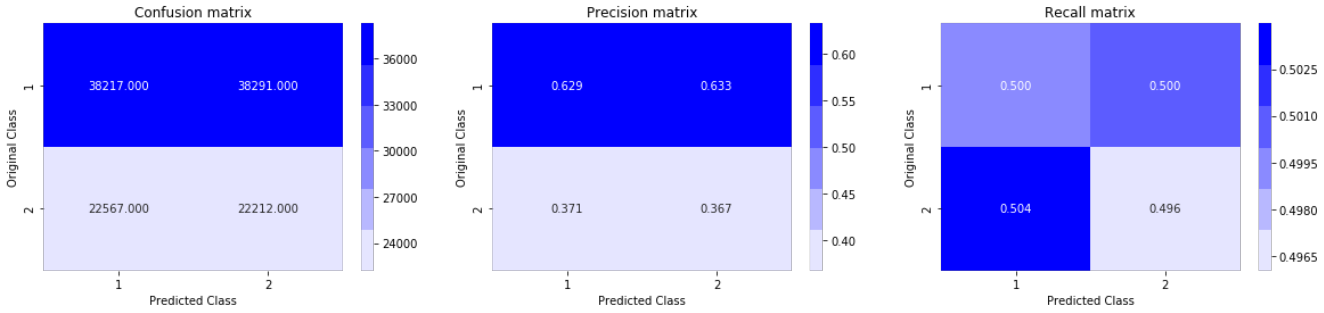
```
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

Random-Model

In [32]:

```
# 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
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-15))

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

```
Log loss on Test Data using Random Model 0.8859064507964042
```



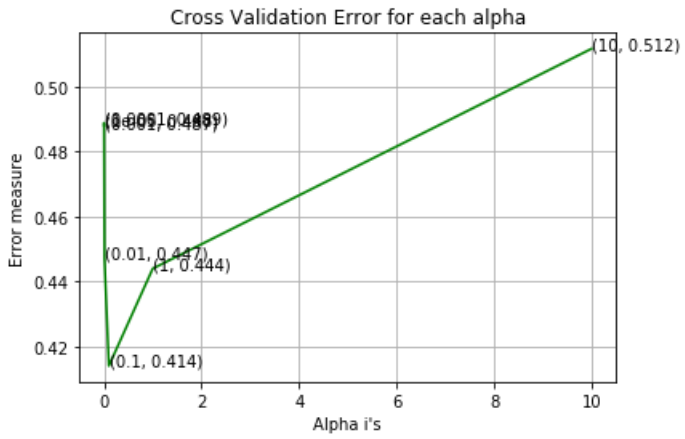
Logistic Regression Model

In [33]:

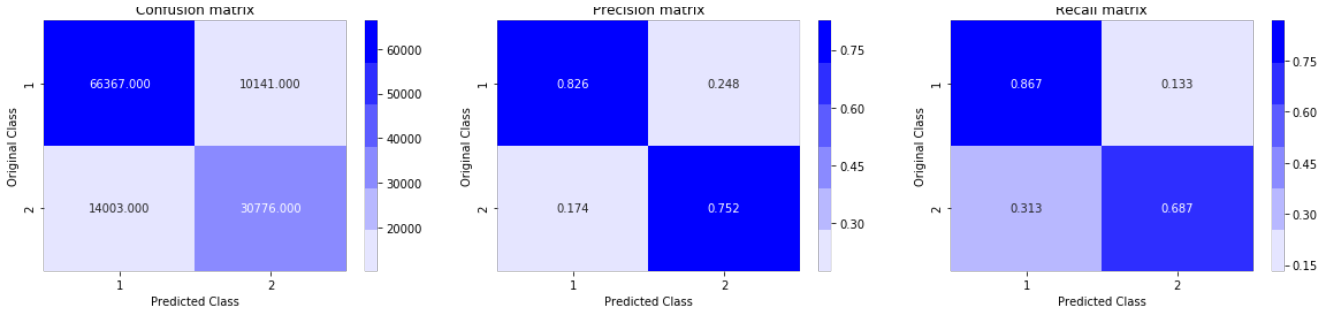
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
```

```
# predict(X) Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(X_train_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.48801186939433655
For values of alpha =  0.0001 The log loss is: 0.4888219072295493
For values of alpha =  0.001 The log loss is: 0.48736640019457933
For values of alpha =  0.01 The log loss is: 0.4470870372129665
For values of alpha =  0.1 The log loss is: 0.4140991710844135
For values of alpha =  1 The log loss is: 0.44399270596606905
For values of alpha =  10 The log loss is: 0.5115952552564148
```




```
For values of best alpha =  0.1 The train log loss is: 0.35785403695333023
For values of best alpha =  0.1 The test log loss is: 0.4140991710844135
Total number of data points : 121287
```



Linear-SVM Model

In [34]:

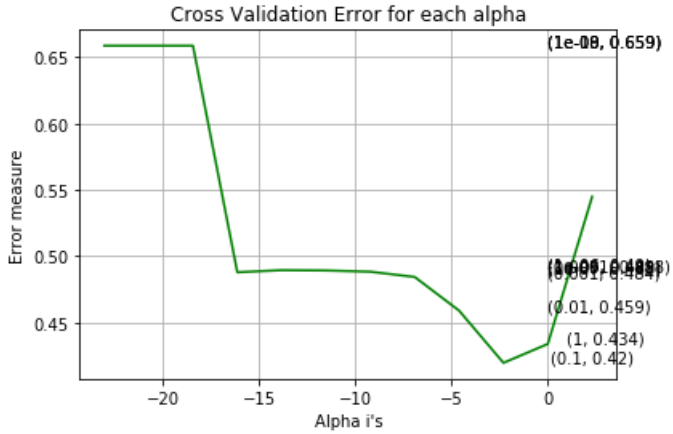
```
alpha = [10 ** x for x in range(-10, 2)] # hyperparam for SGD classifier.

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i,loss='hinge', random_state=42)
    clf.fit(X_train_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_tfidf, y_train)
    predict_y = sig_clf.predict_proba(X_test_tfidf)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))
```

```
For values of alpha =  1e-10 The log loss is: 0.6585278256347589
For values of alpha =  1e-09 The log loss is: 0.6585278256347589
For values of alpha =  1e-08 The log loss is: 0.6585278256347589
For values of alpha =  1e-07 The log loss is: 0.4879849790066122
For values of alpha =  1e-06 The log loss is: 0.4895706862499595
For values of alpha =  1e-05 The log loss is: 0.48932982809633896
For values of alpha =  0.0001 The log loss is: 0.488400091567775
For values of alpha =  0.001 The log loss is: 0.48441931281538647
For values of alpha =  0.01 The log loss is: 0.45908834891535266
For values of alpha =  0.1 The log loss is: 0.4197791277898448
For values of alpha =  1 The log loss is: 0.43402211424542847
For values of alpha =  10 The log loss is: 0.5449229094237861
```

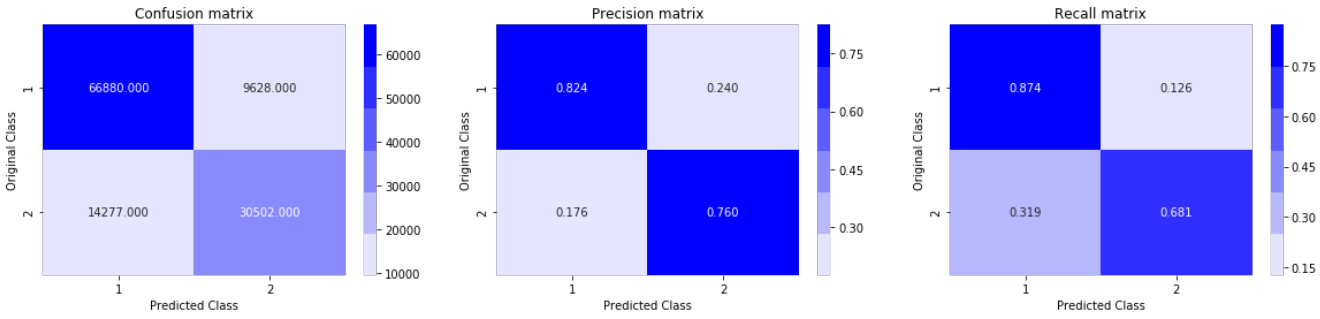
In [35]:

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

```
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha],loss='hinge', random_state=42)
clf.fit(X_train_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_tfidf, y_train)

predict_y = sig_clf.predict_proba(X_train_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```




```
For values of best alpha =  0.1 The train log loss is: 0.3589597205933806
For values of best alpha =  0.1 The test log loss is: 0.4197791277898448
Total number of data points : 121287
```



gradient_boosting with Random-Forest Model

In [0]:

```
d_train = xgb.DMatrix(X_train_tfidf, label=y_train)
d_test = xgb.DMatrix(X_test_tfidf, label=y_test)
```

In [0]:

```
from xgboost import XGBClassifier
import xgboost as xgb
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score


clf = XGBClassifier(objective='binary:logistic',silent=True,eval_metric='logloss')
```

```
parameters = {

    'num_boost_round': [100, 250,500,750],
    'max_depth': [3,5,7,9]

}
```

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
random_search = RandomizedSearchCV(clf,param_distributions=parameters)
```

In [0]:

```
random_search.fit(X_train_tfidf, y_train)
```

Out[0]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, eval_metric='logloss', gamma=0,
       learning_rate=0.1, max_delta_step=0, max_depth=3,
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'num_boost_round': [100, 250, 500, 750], 'max_depth': [3, 5, 7, 9]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=0)
```

In [0]:

```
print('\n All results:')
print(random_search.cv_results_)
print('\n Best estimator:')
print(random_search.best_estimator_)

print(random_search.best_score_ * 2 - 1)
print('\n Best hyperparameters:')
print(random_search.best_params_)
results = pd.DataFrame(random_search.cv_results_)
results
```

```
 All results:
{'mean_fit_time': array([291.18769368, 226.77261408, 224.56700325, 225.41055846,
       101.26991455, 102.59936468, 162.75541162, 161.93595274,
       297.2657733 , 160.35895634]), 'std_fit_time': array([10.40516179,  8.83474391,  9.87862124,
7.86569036,  1.94013593,
        2.20344525,  3.40814909,  3.82151651,  4.96820864,  2.54527585]), 'mean_score_time': array(
[2.2961247 , 2.04757961, 2.08886695, 2.06537493, 1.58836285,
       1.50862956, 1.77330573, 1.78854362, 2.27617534, 1.81396802]), 'std_score_time':
array([0.01579114, 0.01076521, 0.0445974 , 0.01889902, 0.0332575 ,
       0.00774818, 0.0103041 , 0.02239724, 0.02121014, 0.02709846]), 'param_num_boost_round':
masked_array(data=[250, 250, 500, 100, 250, 100, 500, 750, 750, 250],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_max_depth': masked_array(data=[9, 7, 7, 7, 3, 3, 5, 5, 9, 5],
             mask=[False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'num_boost_round': 250, 'max_depth': 9}, {'num_boost_round':
250, 'max_depth': 7}, {'num_boost_round': 500, 'max_depth': 7}, {'num_boost_round': 100,
'max_depth': 7}, {'num_boost_round': 250, 'max_depth': 3}, {'num_boost_round': 100, 'max_depth': 3
}, {'num_boost_round': 500, 'max_depth': 5}, {'num_boost_round': 750, 'max_depth': 5},
{'num_boost_round': 750, 'max_depth': 9}, {'num_boost_round': 250, 'max_depth': 5}],
'split0_test_score': array([0.84348333, 0.83911592, 0.83911592, 0.83911592, 0.82026819,
       0.82026819, 0.83180156, 0.83180156, 0.84348333, 0.83180156]), 'split1_test_score':
array([0.84128734, 0.8372485 , 0.8372485 , 0.8372485 , 0.82107194,
       0.82107194, 0.8322768 , 0.8322768 , 0.84128734, 0.8322768 ]), 'split2_test_score':
array([0.84120254, 0.8371955 , 0.8371955 , 0.8371955 , 0.82059491,
       0.82059491, 0.82979626, 0.82979626, 0.84120254, 0.82979626]), 'mean_test_score':
```

```
array([0.84199107, 0.83785331, 0.83785331, 0.83785331, 0.82064501,
       0.82064501, 0.83129154, 0.83129154, 0.84199107, 0.83129154]), 'std_test_score':
array([0.00105576, 0.00089307, 0.00089307, 0.00089307, 0.00033003,
       0.00033003, 0.00107498, 0.00107498, 0.00105576, 0.00107498]), 'rank_test_score': array([1,
3, 3, 3, 9, 9, 6, 6, 1, 6], dtype=int32), 'split0_train_score': array([0.86484194, 0.84948163,
0.84948163, 0.84948163, 0.82144296,
       0.82144296, 0.83489516, 0.83489516, 0.86484194, 0.83489516]), 'split1_train_score':
array([0.86605113, 0.84984285, 0.84984285, 0.84984285, 0.82124249,
       0.82124249, 0.83632181, 0.83632181, 0.86605113, 0.83632181]), 'split2_train_score':
array([0.86531969, 0.85052658, 0.85052658, 0.85052658, 0.82212234,
       0.82212234, 0.8361787 , 0.8361787 , 0.86531969, 0.8361787 ]), 'mean_train_score':
array([0.86540425, 0.84995035, 0.84995035, 0.84995035, 0.8216026 ,
       0.8216026 , 0.83579856, 0.83579856, 0.86540425, 0.83579856]), 'std_train_score':
array([0.00049726, 0.00043332, 0.00043332, 0.00043332, 0.00037652,
       0.00037652, 0.00064147, 0.00064147, 0.00049726, 0.00064147])}

 Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, eval_metric='logloss', gamma=0,
       learning_rate=0.1, max_delta_step=0, max_depth=9,
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, num_boost_round=250, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
0.6839821485991315

 Best hyperparameters:
{'num_boost_round': 250, 'max_depth': 9}
```

Out[0]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_num_boost_round	param_max_depth	
0	291.187694	10.405162	2.296125	0.015791	250	9	{'num_boos 250, 'max_
1	226.772614	8.834744	2.047580	0.010765	250	7	{'num_boos 250, 'max_
2	224.567003	9.878621	2.088867	0.044597	500	7	{'num_boos 500, 'max_
3	225.410558	7.865690	2.065375	0.018899	100	7	{'num_boos 100, 'max_
4	101.269915	1.940136	1.588363	0.033257	250	3	{'num_boos 250, 'max_
5	102.599365	2.203445	1.508630	0.007748	100	3	{'num_boos 100, 'max_
6	162.755412	3.408149	1.773306	0.010304	500	5	{'num_boos 500, 'max_
7	161.935953	3.821517	1.788544	0.022397	750	5	{'num_boos 750, 'max_
8	297.265773	4.968209	2.276175	0.021210	750	9	{'num_boos 750, 'max_
9	160.358956	2.545276	1.813968	0.027098	250	5	{'num_boos 250, 'max_

In [0]:

```
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['num_boost_round'] = 250
params['max_depth'] = 9

d_train = xgb.DMatrix(X_train_tfidf, label=y_train)
d_test = xgb.DMatrix(X_test_tfidf, label=y_test)

watchlist = [(d_train, 'train'), (d_test, 'valid')]
```

```
bst = xgb.train(params, d_train, 250, watchlist,  verbose_eval=10)

xgdmat = xgb.DMatrix(X_train_tfidf,y_train)
predict_y = bst.predict(d_test)
```

```
[0] train-logloss:0.559537 valid-logloss:0.565763
[10] train-logloss:0.32826 valid-logloss:0.352158
[20] train-logloss:0.310161 valid-logloss:0.338871
[30] train-logloss:0.298128 valid-logloss:0.332603
[40] train-logloss:0.290474 valid-logloss:0.328241
[50] train-logloss:0.2859 valid-logloss:0.3264
[60] train-logloss:0.279997 valid-logloss:0.324306
[70] train-logloss:0.274894 valid-logloss:0.322435
[80] train-logloss:0.270575 valid-logloss:0.321105
[90] train-logloss:0.267292 valid-logloss:0.320249
[100] train-logloss:0.264138 valid-logloss:0.318958
[110] train-logloss:0.260593 valid-logloss:0.317887
[120] train-logloss:0.258003 valid-logloss:0.317378
[130] train-logloss:0.254846 valid-logloss:0.316312
[140] train-logloss:0.251377 valid-logloss:0.31529
[150] train-logloss:0.249303 valid-logloss:0.3149
[160] train-logloss:0.247566 valid-logloss:0.31426
[170] train-logloss:0.244946 valid-logloss:0.313855
[180] train-logloss:0.242103 valid-logloss:0.312675
[190] train-logloss:0.240644 valid-logloss:0.312412
[200] train-logloss:0.238984 valid-logloss:0.312128
[210] train-logloss:0.237398 valid-logloss:0.311909
[220] train-logloss:0.235032 valid-logloss:0.311345
[230] train-logloss:0.233683 valid-logloss:0.311079
[240] train-logloss:0.232073 valid-logloss:0.310845
[249] train-logloss:0.23038 valid-logloss:0.310489
```

In [0]:

```
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
```

```
The test log loss is: 0.3105131150851162
```

PrettyTable

In [6]:

```
import numpy as np
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Alpha", "Train-AUC","Test-AUC"]


x.add_row(["Logistic-Regression",0.1,0.3578,0.4140])
x.add_row(["Linear-Regression",0.1,0.3589,0.41977])
print(x)
y = PrettyTable()
y.field_names = ["Model", "num_boost_round","max_depth","Train-AUC","Test-AUC"]
y.add_row(["Gradient-Boosting",250,9,0.23,0.31051])
print(y)
```

```
+---------------------+-------+-----------+----------+
|        Model        | Alpha | Train-AUC | Test-AUC |
+---------------------+-------+-----------+----------+
| Logistic-Regression |  0.1  |   0.3578  |  0.414   |
|  Linear-Regression  |  0.1  |   0.3589  | 0.41977  |
+---------------------+-------+-----------+----------+
+-------------------+-----------------+-----------+-----------+----------+
|       Model       | num_boost_round | max_depth | Train-AUC | Test-AUC |
+-------------------+-----------------+-----------+-----------+----------+
| Gradient-Boosting |       250       |     9     |    0.23   | 0.31051  |
+-------------------+-----------------+-----------+-----------+----------+
```