Pima Indians Diabetes Database

1.Introduction

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

```
Pregnancies: Number of times pregnant
Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
BloodPressure: Diastolic blood pressure (mm Hg)
SkinThickness: Triceps skin fold thickness (mm)
Insulin: 2-Hour serum insulin (mu U/ml)
BMI: Body mass index (weight in kg/(height in m)^2)
DiabetesPedigreeFunction: Diabetes pedigree function
Age: Age (years)
Outcome: Class variable (0 or 1)
```

1.1. Description

Source: https://www.kaggle.com/uciml/pima-indians-diabetes-database/downloads/pima-indians-diabetes-database.zip/1 (https://www.kaggle.com/uciml/pima-indians-diabetes-database/downloads/pima-indians-diabetes-database.zip/1)

Data: Download diabetes.zip from Kaggle.

Problem statement :
The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

1.2. Source/Useful Links

Some blogs about problem statement

https://www.researchgate.net/publication/301335647_Cascaded_Modeling_for_PIMA_Indian_Diabetes_Data

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 (https://www.kaggle.com/c/msk-redefining-cancer-treatment/data)](https://www.kaggle.com/c/msk-redefining-cancer-treatment/data)

```
We have one data file: one conatins the information about the diabetes.

Data file information:
diabetes (Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigree
Function,Age,Outcome)
```

2.2. Machine Learing Objectives and Constraints

Objective:

Predict the probability of each data-point belonging to two 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]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
import sqlite3
#from sqlalchemy import create_engine
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
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.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from nltk.stem.porter import PorterStemmer


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

3.1. Reading Data

In [7]:
```
diabetes = pd.read_csv('diabetes.csv')
print(diabetes.columns)
print('\nNumber of data points : ', diabetes.shape[0])
print(' Number of features : ', diabetes.shape[1])
print('\nFeatures : ', diabetes.columns.values)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

Number of data points :  768
 Number of features :  9

Features :  ['Pregnancies' 'Glucose' 'BloodPressure' 'SkinThickness' 'Insulin' '
BMI'
 'DiabetesPedigreeFunction' 'Age' 'Outcome']
```

In [34]:
```
diabetes.head()
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

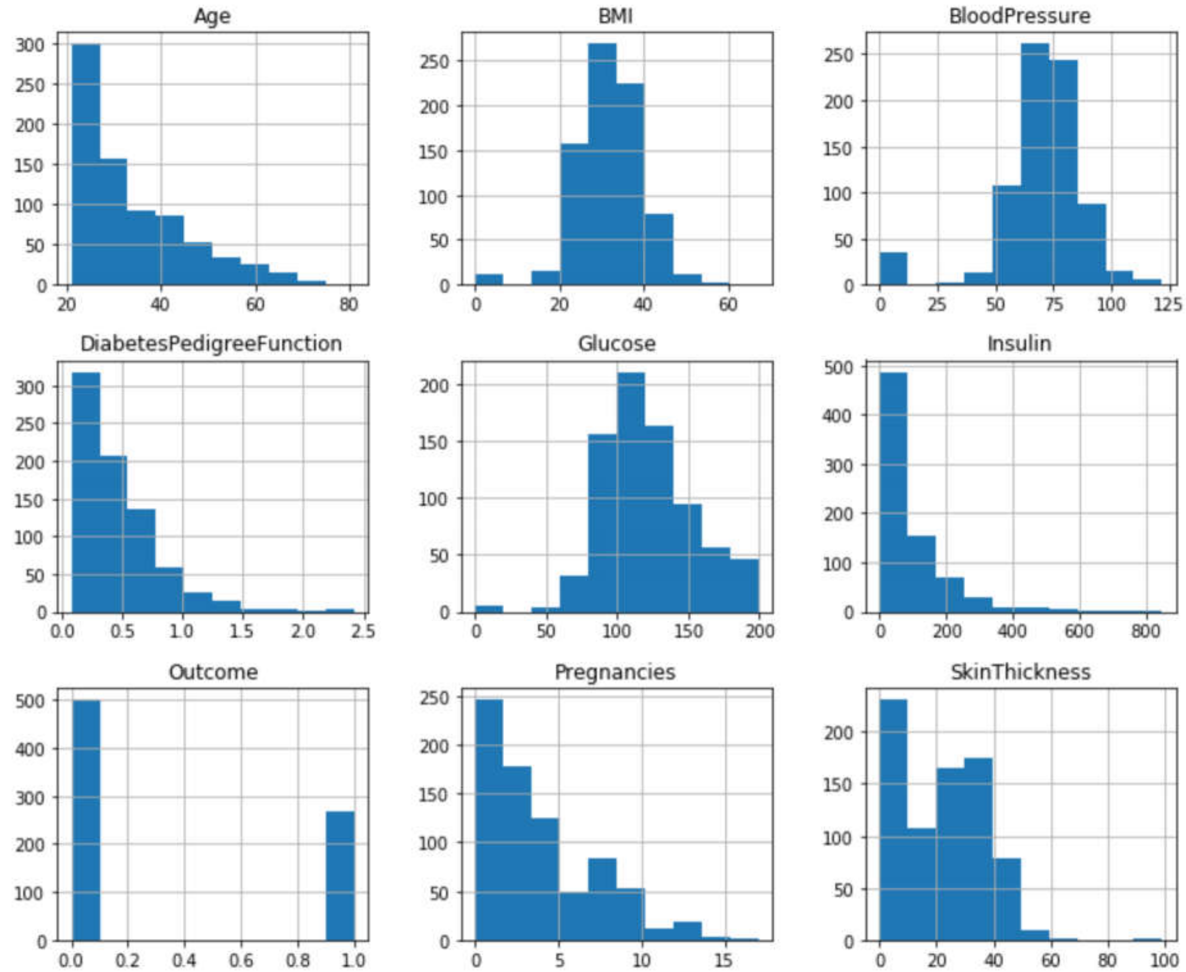
Preprocessing of data

In [11]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

        data_text[column][index] = string
```

In [12]:
```
#text processing stage.
start_time = time.clock()
for index, row in diabetes.iterrows():
    if type(row['Outcome']) is str:
        nlp_preprocessing(row['Outcome'], index, 'Outcome')
    else:
        print("There is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "second
s")
```

There is no text description for id: 0
There is no text description for id: 1
There is no text description for id: 2
There is no text description for id: 3
There is no text description for id: 4
There is no text description for id: 5
There is no text description for id: 6
There is no text description for id: 7
There is no text description for id: 8
There is no text description for id: 9
There is no text description for id: 10
There is no text description for id: 11
There is no text description for id: 12
There is no text description for id: 13
There is no text description for id: 14
There is no text description for id: 15
There is no text description for id: 16
There is no text description for id: 17
There is no text description for id: 18
There is no text description for id: 19
There is no text description for id: 20
There is no text description for id: 21
There is no text description for id: 22
There is no text description for id: 23
There is no text description for id: 24
There is no text description for id: 25
There is no text description for id: 26
There is no text description for id: 27
There is no text description for id: 28
There is no text description for id: 29
There is no text description for id: 30
There is no text description for id: 31
There is no text description for id: 32
There is no text description for id: 33
There is no text description for id: 34
There is no text description for id: 35
There is no text description for id: 36
There is no text description for id: 37
There is no text description for id: 38
There is no text description for id: 39
There is no text description for id: 40
There is no text description for id: 41
There is no text description for id: 42
There is no text description for id: 43
There is no text description for id: 44
There is no text description for id: 45
There is no text description for id: 46
There is no text description for id: 47
There is no text description for id: 48
There is no text description for id: 49
There is no text description for id: 50
There is no text description for id: 51
There is no text description for id: 52
There is no text description for id: 53
There is no text description for id: 54
There is no text description for id: 55
There is no text description for id: 56
There is no text description for id: 57
There is no text description for id: 58
There is no text description for id: 59
There is no text description for id: 60
There is no text description for id: 61
There is no text description for id: 62
There is no text description for id: 63

In [27]:
```
df = diabetes.hist(figsize=(12,10))
```



In [28]:
```
sns.pairplot(diabetes,hue='Outcome')
```

Out[28]: <seaborn.axisgrid.PairGrid at 0x2c4a00c0a90>

In [23]:
```
X = diabetes.drop(diabetes.columns[8:],axis=1)
y = diabetes['Outcome']
```

In [25]:
```
x.shape
```

Out[25]: (768,)

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

In [68]:

```
# split the data into test and train by maintaining same distribution of output var
aible 'y_true' [stratify=y_true]
#X_train, X_test, y_train, y_test = train_test_split(x, y, stratify=y, test_siz
e=0.2)
# split the train data into train and cross validation by maintaining same distribu
tion of output varaible 'y_train' [stratify=y_train]
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_trai
n, test_size=0.2)



X_train, X_test, y_train, y_test = train_test_split(diabetes.loc[:, diabetes.column
s != 'Outcome'], diabetes['Outcome'], stratify=diabetes['Outcome'], test_size=0.2)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train,
test_size=0.2)
```

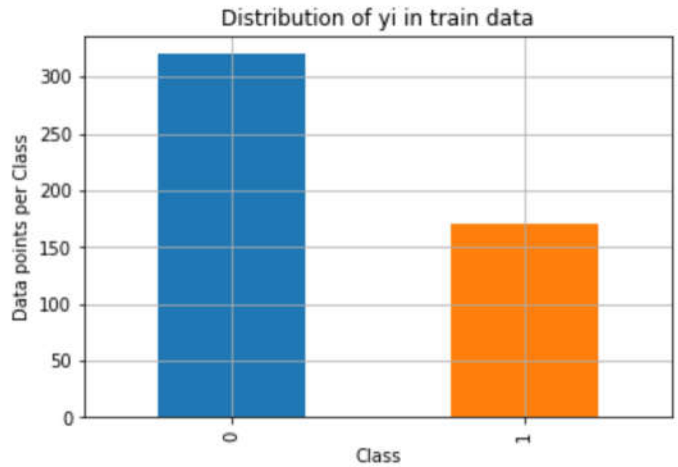
In [59]:

```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

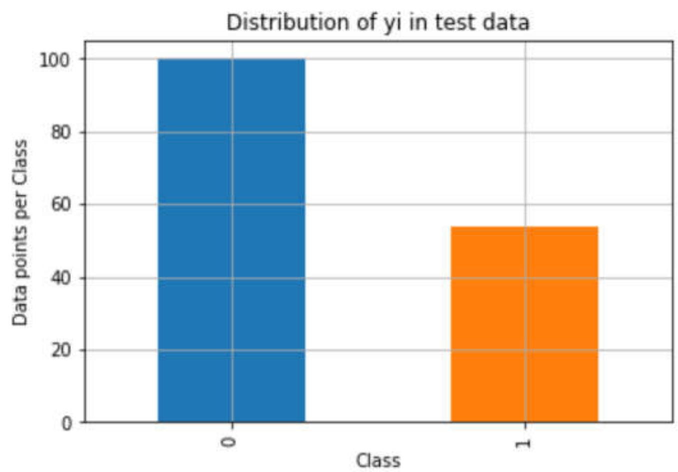
```
Number of data points in train data: 491
Number of data points in test data: 154
Number of data points in cross validation data: 123
```

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

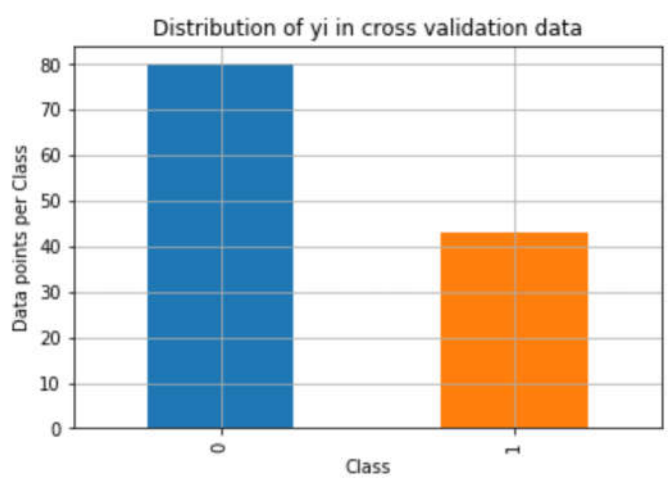
In [60]:
```
# it returns a dict, keys as class labels and values as the number of data points i
n that class
train_class_distribution = X_train.value_counts().sortlevel()
test_class_distribution = X_test.value_counts().sortlevel()
cv_class_distribution = X_cv.value_counts().sortlevel()

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

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


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

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

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

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




```
Number of data points in class 1 : 320 ( 65.173 %)
Number of data points in class 2 : 171 ( 34.827 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 1 : 100 ( 64.935 %)
Number of data points in class 2 : 54 ( 35.065 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 1 : 80 ( 65.041 %)
Number of data points in class 2 : 43 ( 34.959 %)
```

ML Models

Random Model

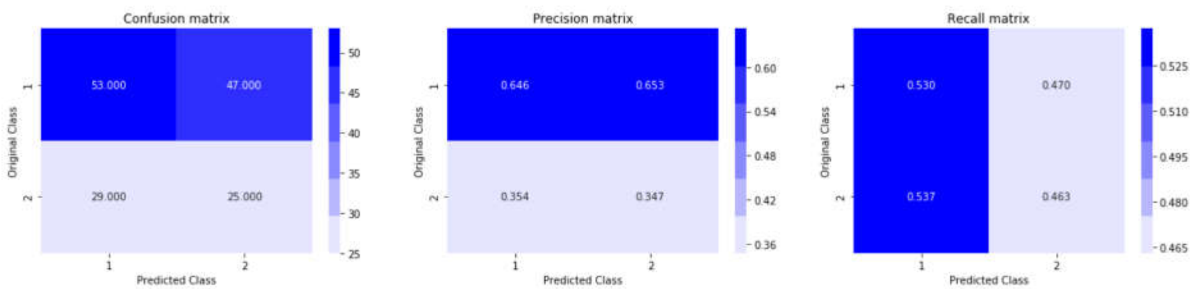
In [61]:
```
from collections import Counter

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])/tr
ain_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:  0.6517311608961304 Class 1:  0.34826883910386963
---------- Distribution of output variable in train data ----------
Class 0:  0.35064935064935066 Class 1:  0.35064935064935066
```

```
In [62]: # 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 pr
edicted class j

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

    # 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 ro
w
    # 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")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabel
s=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, yticklabel
s=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, yticklabel
s=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [63]:
```python
# 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
test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,2))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,2)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predi
cted_y, eps=1e-15))
print()

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, ep
s=1e-15))

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

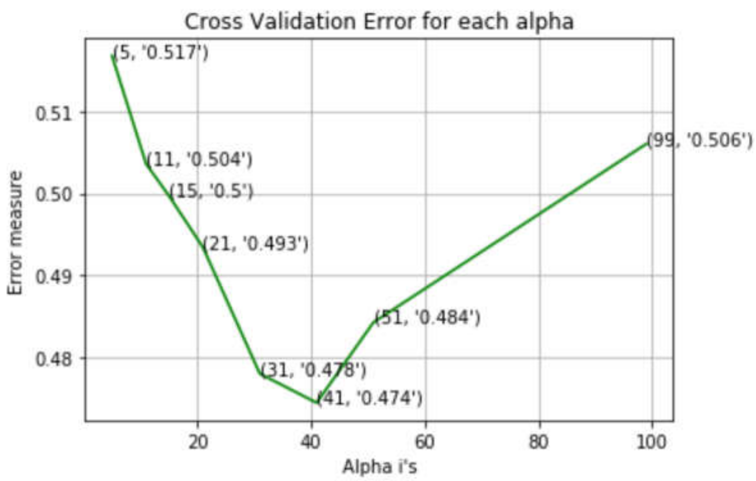
```
Log loss on Cross Validation Data using Random Model 0.8441214457325574

Log loss on Test Data using Random Model 0.8721471837343794
```



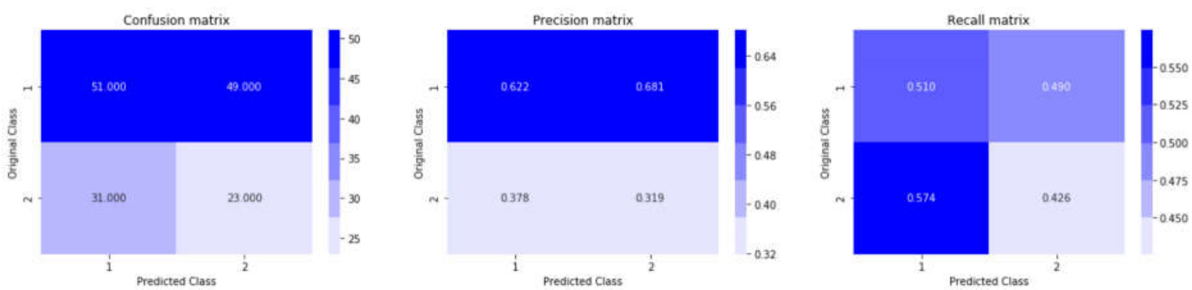
KNN

In [70]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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
#-------------------------------------
# video link:
#-------------------------------------


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, ep
s=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(X_train, y_train)
```

```
for alpha = 5
Log Loss : 0.5168060931689324
for alpha = 11
Log Loss : 0.5035917615910691
for alpha = 15
Log Loss : 0.4997111429382002
for alpha = 21
Log Loss : 0.49338986384326183
for alpha = 31
Log Loss : 0.4779613889542596
for alpha = 41
Log Loss : 0.47439788081078116
for alpha = 51
Log Loss : 0.4842252018950429
for alpha = 99
Log Loss : 0.5060386078722139
```



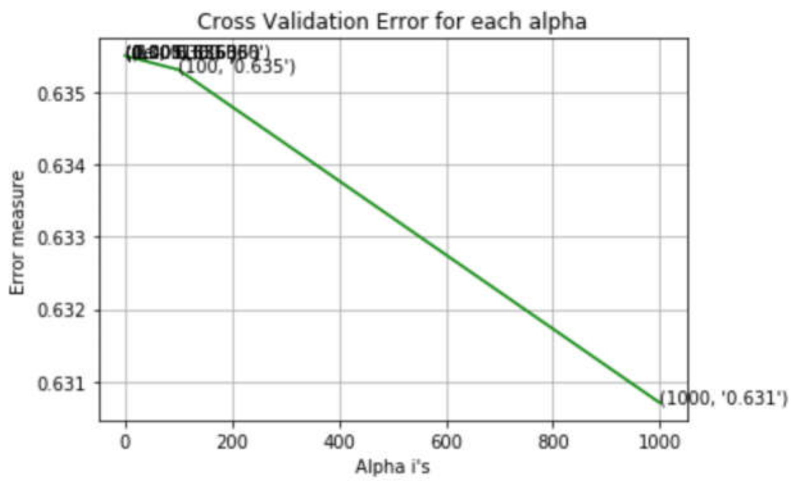

```
For values of best alpha =  41 The train log loss is: 0.5158195221936793
For values of best alpha =  41 The cross validation log loss is:
0.47439788081078116
For values of best alpha =  41 The test log loss is: 0.5304084636794256
```



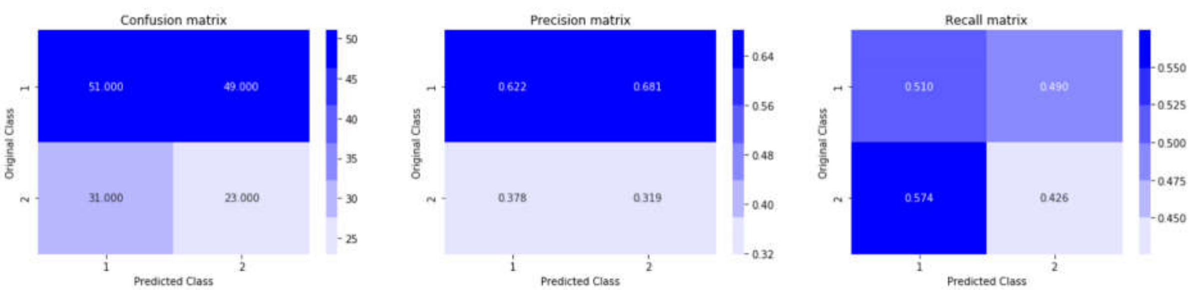
Naive Bayes

In [72]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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
#-------------------------------------
# video link:
#-------------------------------------


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

alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, ep
s=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
```

```
for alpha = 1e-05
Log Loss : 0.6355109916137066
for alpha = 0.0001
Log Loss : 0.6355109914261106
for alpha = 0.001
Log Loss : 0.6355109895501544
for alpha = 0.1
Log Loss : 0.6355107832743276
for alpha = 1
Log Loss : 0.6355089151257886
for alpha = 10
Log Loss : 0.6354908249959852
for alpha = 100
Log Loss : 0.6353178859475573
for alpha = 1000
Log Loss : 0.6307028245388422
```




```
For values of best alpha =  1000 The train log loss is: 0.6351331380973919
For values of best alpha =  1000 The cross validation log loss is:
0.6307028245388422
For values of best alpha =  1000 The test log loss is: 0.6292748733492219
```

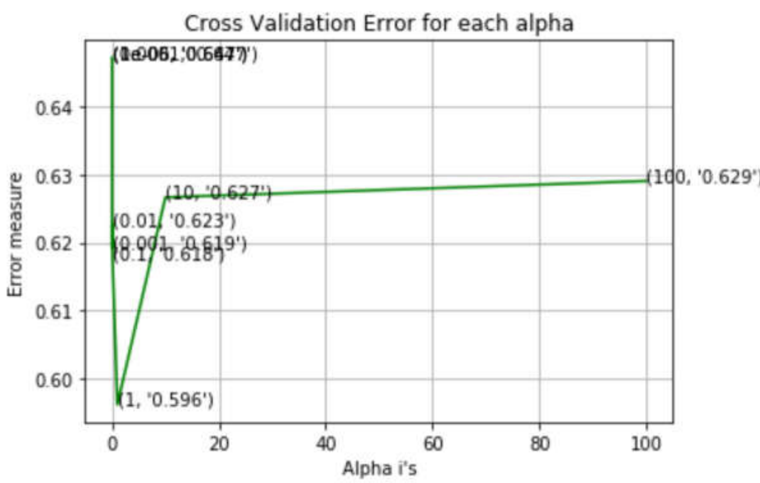


Logistic Regression

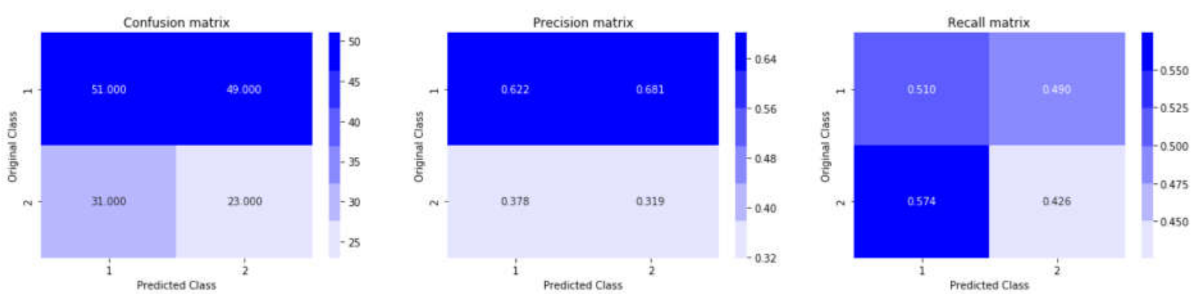
In [73]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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
#--------------------------------------
# video link:
#--------------------------------------


#alpha = [5, 11, 15, 21, 31, 41, 51, 99]
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, ep
s=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_stat
```

```
for alpha = 1e-06
Log Loss : 0.6471952054203212
for alpha = 1e-05
Log Loss : 0.6471952054203212
for alpha = 0.0001
Log Loss : 0.6471952054203212
for alpha = 0.001
Log Loss : 0.6192179915827012
for alpha = 0.01
Log Loss : 0.6226757684881098
for alpha = 0.1
Log Loss : 0.6176607480382904
for alpha = 1
Log Loss : 0.5961438120893248
for alpha = 10
Log Loss : 0.626639513879148
for alpha = 100
Log Loss : 0.6290463180792908
```



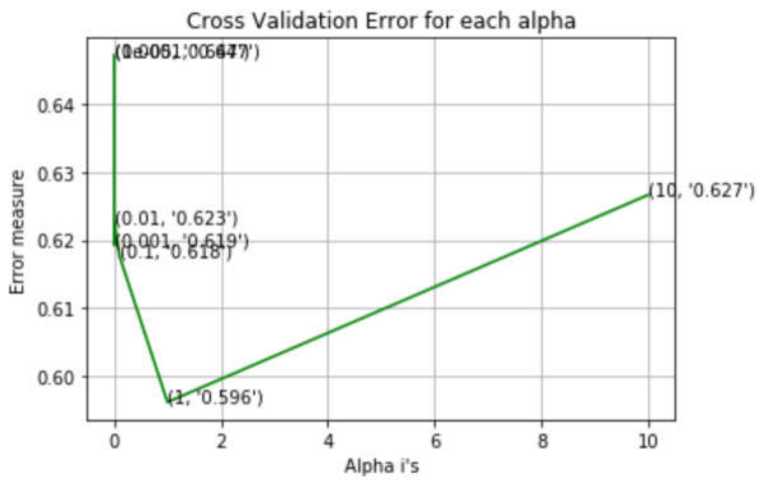

```
For values of best alpha =  1 The train log loss is: 0.5990292770619033
For values of best alpha =  1 The cross validation log loss is:
0.5961438120893248
For values of best alpha =  1 The test log loss is: 0.5953866725726077
```



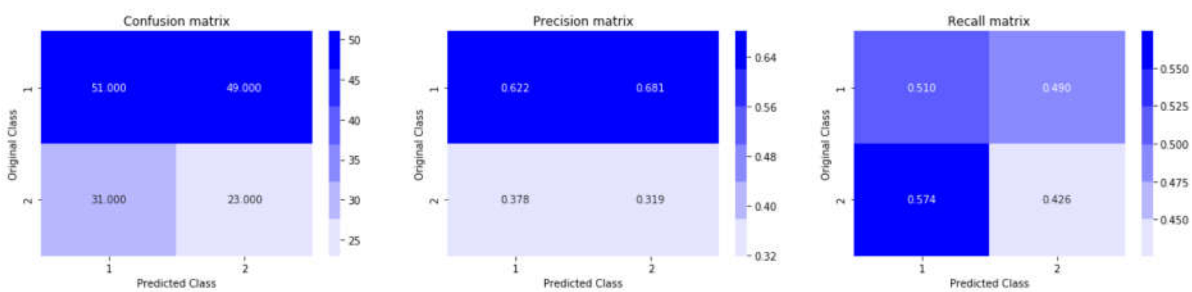
Linear SVM

In [74]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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.
#------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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
#------------------------------------
# video link:
#------------------------------------

alpha = [10 ** x for x in range(-5, 2)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, ep
s=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
estimates
    print("Log Loss :",log_loss(y_cv, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_stat
e=42)
clf.fit(X_train, y_train)
```

```
for alpha = 1e-05
Log Loss : 0.6471952054203212
for alpha = 0.0001
Log Loss : 0.6471952054203212
for alpha = 0.001
Log Loss : 0.6192179915827012
for alpha = 0.01
Log Loss : 0.6226757684881098
for alpha = 0.1
Log Loss : 0.6176607480382904
for alpha = 1
Log Loss : 0.5961438120893248
for alpha = 10
Log Loss : 0.626639513879148
```



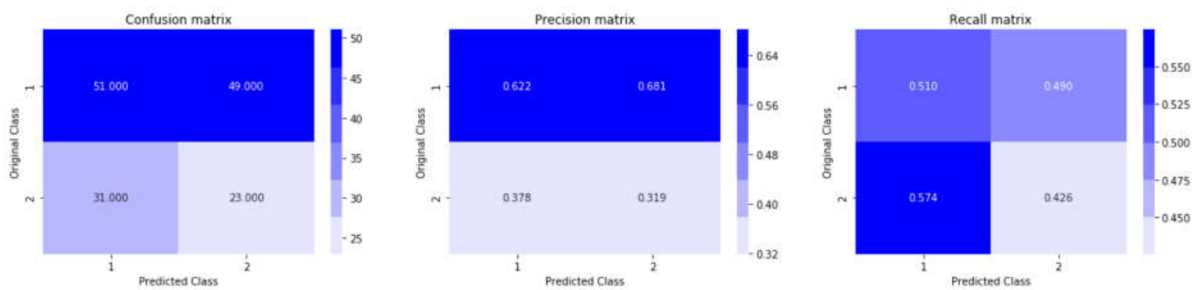

```
For values of best alpha =  1 The train log loss is: 0.5990292770619033
For values of best alpha =  1 The cross validation log loss is:
0.5961438120893248
For values of best alpha =  1 The test log loss is: 0.5953866725726077
```



Random Forest Classifier

In [78]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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.
#------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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
#------------------------------------
# video link:
#------------------------------------

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for alpha =", i)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j,
random_state=42, n_jobs=-1)
        clf.fit(X_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_train, y_train)
        sig_clf_probs = sig_clf.predict_proba(X_cv)
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
        # to avoid rounding error while multiplying probabilites we use log-probabi
lity estimates
        print("Log Loss :",log_loss(y_cv, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
```

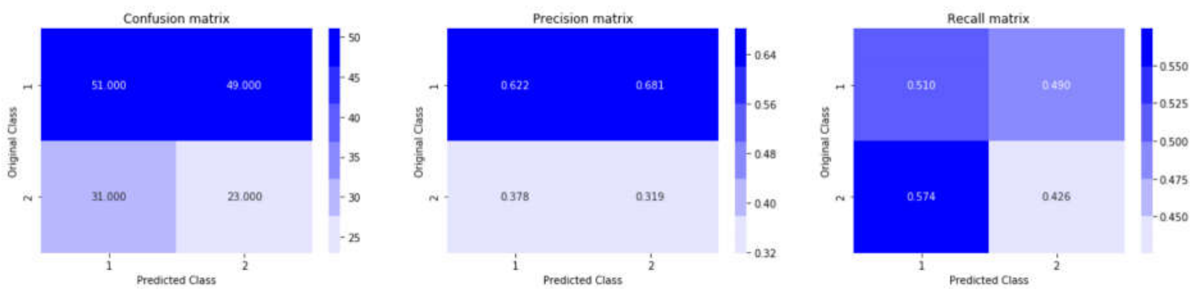
```
for alpha = 100
Log Loss : 0.44212826172224595
for alpha = 100
Log Loss : 0.45163382737202235
for alpha = 200
Log Loss : 0.44009834208492116
for alpha = 200
Log Loss : 0.44724275084205495
for alpha = 500
Log Loss : 0.4406821458038341
for alpha = 500
Log Loss : 0.44902273377077806
for alpha = 1000
Log Loss : 0.43885195935768606
for alpha = 1000
Log Loss : 0.44575086668260694
for alpha = 2000
Log Loss : 0.4400612356016051
for alpha = 2000
Log Loss : 0.4467824480801597
The train log loss is: 0.3384895930782588
The cross validation log loss is: 0.43885195935768606
The test log loss is: 0.5182113221974364
```



XgBoost

In [80]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/module
s/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_siz
e=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.
#------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesso
ns/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/mod
ules/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
#------------------------------------
# video link:
#------------------------------------
from xgboost.sklearn import XGBClassifier
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for alpha =", i)
        clf = XGBClassifier(n_estimators=i,  max_depth=j, random_state=42, n_job
s=-1)


        clf.fit(X_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(X_train, y_train)
        sig_clf_probs = sig_clf.predict_proba(X_cv)
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
        # to avoid rounding error while multiplying probabilites we use log-probabi
lity estimates
        print("Log Loss :",log_loss(y_cv, sig_clf_probs))

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

```
for alpha = 100
Log Loss : 0.4804556235542688
for alpha = 100
Log Loss : 0.4913428765529893
for alpha = 200
Log Loss : 0.4935896353742086
for alpha = 200
Log Loss : 0.5084724983414165
for alpha = 500
Log Loss : 0.5090778262749956
for alpha = 500
Log Loss : 0.5123155540670282
for alpha = 1000
Log Loss : 0.515958089327309
for alpha = 1000
Log Loss : 0.5186448183703771
for alpha = 2000
Log Loss : 0.5206624008038975
for alpha = 2000
Log Loss : 0.5249457770322743
For values of best alpha =  100 The train log loss is: 0.3024696664015824
For values of best alpha =  100 The cross validation log loss is:
0.4804556235542688
For values of best alpha =  100 The test log loss is: 0.5141637726418922
```



Conclusion

In [1]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Train log loss","CV log loss", "Test log loss"]
x.add_row(["Random medel",'..', 0.84,0.87])
x.add_row(["KNN",0.51, 0.47,0.53])
x.add_row(["Naive Bayes",0.63, 0.63,0.62])
x.add_row(["Logistic Regression",0.59, 0.59,0.59])
x.add_row(["Linear SVM",0.59, 0.59,0.59])
x.add_row(["Random Forest",0.33, 0.43,0.51])
x.add_row(["XgBoost",0.30, 0.48,0.51])
print(x)
```

```
+---------------------+----------------+-------------+---------------+
|        Model        | Train log loss | CV log loss | Test log loss |
+---------------------+----------------+-------------+---------------+
|     Random medel    |       ..       |     0.84    |      0.87     |
|         KNN         |      0.51      |     0.47    |      0.53     |
|     Naive Bayes     |      0.63      |     0.63    |      0.62     |
| Logistic Regression |      0.59      |     0.59    |      0.59     |
|      Linear SVM     |      0.59      |     0.59    |      0.59     |
|    Random Forest    |      0.33      |     0.43    |      0.51     |
|       XgBoost       |      0.3       |     0.48    |      0.51     |
+---------------------+----------------+-------------+---------------+
```