Santiago_Gabriela

April 8, 2022

1 1.) Load Data and perform basic EDA

2 This jupyter notebook is prepared by "Gabriela Santiago".

import libraries: pandas, numpy, matplotlib (set %matplotlib inline), matplotlib's pyplot, seaborn, missingno, scipy's stats, sklearn (1 pt)

[134]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import missingno as msno
import scipy.stats as st
import sklearn
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import imblearn
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
import math
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import (precision_recall_curve, PrecisionRecallDisplay)
from sklearn.model_selection import cross_val_predict
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import make_hastie_10_2
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import CategoricalNB
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets, metrics, model_selection, svm
from sklearn.metrics import auc
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import f1_score
from sklearn.neighbors import KNeighborsClassifier
```

import the data to a dataframe and show the count of rows and columns (1 pt)

[135]:
```
df = pd.read_csv('hrdata2.csv')
print("Rows: ", len(df))
print("Columns: ", len(df.columns))
```

```
Rows:  8955
Columns:  15
```

Show the top 5 and last 5 rows (1 pt)

[136]:
```
df.head()
```

[136]:

	Unnamed: 0	enrollee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	company_type	last_new_job
0	1	29725	city_40	0.776	Male	No relevent experience	no_enrollment	Graduate	STEM	15.0	50-99	Pvt Ltd	>4
1	4	666	city_162	0.767	Male	Has relevent experience	no_enrollment	Masters	STEM	21.0	50-99	Funded Startup	4
2	7	402	city_46	0.762	Male	Has relevent experience	no_enrollment	Graduate					
3	8	27107	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate					
4	11	23853	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate					

```
2             STEM        13.0          <10         Pvt Ltd           >4
3             STEM         7.0        50-99         Pvt Ltd            1
4             STEM         5.0    5000-9999         Pvt Ltd            1

   training_hours  target
0              47     0.0
1               8     0.0
2              18     1.0
3              46     1.0
4             108     0.0
```

[137]:
```
df.tail()
```

[137]:
```
      Unnamed: 0  enrollee_id      city  city_development_index  gender  \
8950       19147        21319   city_21                   0.624    Male
8951       19149          251  city_103                   0.920    Male
8952       19150        32313  city_160                   0.920  Female
8953       19152        29754  city_103                   0.920  Female
8954       19155        24576  city_103                   0.920    Male

          relevent_experience enrolled_university education_level  \
8950   No relevent experience    Full time course        Graduate
8951  Has relevent experience       no_enrollment         Masters
8952  Has relevent experience       no_enrollment        Graduate
8953  Has relevent experience       no_enrollment        Graduate
8954  Has relevent experience       no_enrollment        Graduate

     major_discipline  experience company_size    company_type last_new_job  \
8950             STEM         1.0      100-500         Pvt Ltd            1
8951             STEM         9.0        50-99         Pvt Ltd            1
8952             STEM        10.0      100-500   Public Sector            3
8953       Humanities         7.0        10/49  Funded Startup            1
8954             STEM        21.0        50-99         Pvt Ltd            4

      training_hours  target
8950              52     1.0
8951              36     1.0
8952              23     0.0
8953              25     0.0
8954              44     0.0
```

Show how many columns have null values

[138]:
```
nulls = df.isnull().sum().to_frame('nulls')
nulls.sort_values("nulls", inplace = True, ascending = False)
for index, column in nulls.iteritems():
    print("Number of columns containing null values: ", column[0])
```

```
Number of columns containing null values:  0
```

Plot the count of target and discuss its imbalances and probably issues and solutions

[139]:
```
print("Total count of target values: ", df['target'].count())
print("Total count of target = 1: ", df['target'].value_counts()[1])
print("Total count of target = 0: ", df['target'].value_counts()[0])
```

```
Total count of target values:  8955
Total count of target = 1:  1483
Total count of target = 0:  7472
```

3 Answer:

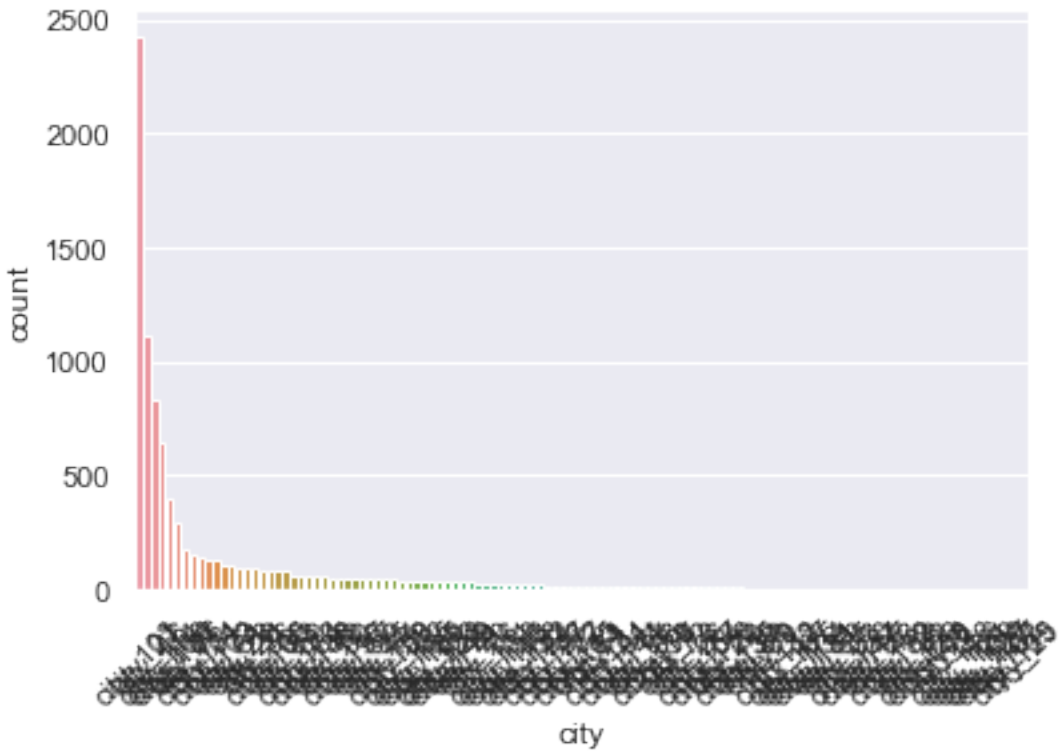
There's definitely an inbalance of target 'yes' values to 'no' values, with 'no' being far more plentiful than the 'yes' value. This might be fixed by undersampling, although it would feel like a waste of a lot of potentially valuable data. Over sampling might help too, but making too much synthetic data might mess with the reliability of the result.

4 2.) Feature Selection and Pre-processing

Preprocessing City:

Plot #of records per city so that the highest city counts are shown in descending order

[140]:
```
city = df['city']
city = np.sort(city)
city = city[::-1]
chart = sns.countplot(x = 'city', data = df, order = df['city'].value_counts().
 ↪index)
chart.set_xticklabels(chart.get_xticklabels(), rotation=45)
sns.set(rc={'figure.figsize':(200, 600)},)
plt.show()
```



How many rows belong to the top 4 cities in total and how many for the remaining?

[141]:
```
df['city'].value_counts()
```

[141]:
```
city_103    2426
city_21     1111
city_16      836
city_114     648
city_160     401
            …
city_127       1
city_107       1
city_62        1
city_109       1
city_25        1
Name: city, Length: 116, dtype: int64
```

[142]:
```
print("Total cities in the top 4: ", (df['city'].value_counts()['city_103'] +␣
↪df['city'].value_counts()['city_21'] + df['city'].value_counts()['city_16']␣
↪+ df['city'].value_counts()['city_114']))
```

```
print("Total cities not in the top 4: ", 8955 - (df['city'].
↪value_counts()['city_103'] + df['city'].value_counts()['city_21'] +␣
↪df['city'].value_counts()['city_16'] + df['city'].
↪value_counts()['city_114']))
```

```
Total cities in the top 4:  5021
Total cities not in the top 4:  3934
```

Top four cities in order from most to least are: city_103, city_21, city_16, and city_114. Since there's 8955 total rows, and the total count of rows in the top 4 are 5021, 8955 - 5021 = 3934.

Replace the city name with city_others if the city name is not within the top 4 city names

[143]:
```
for rows in df['city']:
    if (rows != 'city_103') and (rows != 'city_21') and (rows != 'city_16') and␣
↪(rows != 'city_114'):
        df['city'] = df['city'].replace(rows,'city_others')
```

Show some sample data that the records have changed appropriately

[144]:
```
df.head()
```

[144]:

	Unnamed: 0	enrollee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	company_type	last_new_job	training_hours	target
0	1	29725	city_others	0.776	Male	No relevent experience	no_enrollment	Graduate	STEM	15.0	50-99	Pvt Ltd	>4	47	0.0
1	4	666	city_others	0.767	Male	Has relevent experience	no_enrollment	Masters	STEM	21.0	50-99	Funded Startup	4	8	0.0
2	7	402	city_others	0.762	Male	Has relevent experience	no_enrollment	Graduate	STEM	13.0	<10	Pvt Ltd	>4	18	1.0
3	8	27107	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate	STEM	7.0	50-99	Pvt Ltd	1	46	1.0
4	11	23853	city_103	0.920	Male	Has relevent experience	no_enrollment	Graduate	STEM	5.0	5000-9999	Pvt Ltd	1	108	0.0

Education Level:

Show the unique values of education level.

[145]:
```
df['education_level'].unique()
```

[145]:
```
array(['Graduate', 'Masters', 'Phd'], dtype=object)
```

Replace the value of Education level column like ordinal values, “Graduate” -> 0, Masters->1, and Phd -> 2

[146]:
```
df['education_level'] = df['education_level'].replace({'Graduate':0, 'Masters':
↪1, 'Phd':2})
```

Show some sample data that the records have changed appropriately

[147]:
```
df.tail()
```

[147]:

	Unnamed: 0	enrollee_id	city	city_development_index	gender	\
8950	19147	21319	city_21	0.624	Male	
8951	19149	251	city_103	0.920	Male	
8952	19150	32313	city_others	0.920	Female	
8953	19152	29754	city_103	0.920	Female	
8954	19155	24576	city_103	0.920	Male	

	relevent_experience	enrolled_university	education_level	\
8950	No relevent experience	Full time course	0	
8951	Has relevent experience	no_enrollment	1	
8952	Has relevent experience	no_enrollment	0	
8953	Has relevent experience	no_enrollment	0	
8954	Has relevent experience	no_enrollment	0	

	major_discipline	experience	company_size	company_type	last_new_job	\
8950	STEM	1.0	100-500	Pvt Ltd	1	
8951	STEM	9.0	50-99	Pvt Ltd	1	
8952	STEM	10.0	100-500	Public Sector	3	
8953	Humanities	7.0	10/49	Funded Startup	1	
8954	STEM	21.0	50-99	Pvt Ltd	4	

	training_hours	target
8950	52	1.0
8951	36	1.0
8952	23	0.0
8953	25	0.0
8954	44	0.0

Company_size column:

Show the unique values of the company_size column

[148]:
```
df['company_size'].unique()
```

[148]:
```
array(['50-99', '<10', '5000-9999', '1000-4999', '10/49', '100-500',
       '10000+', '500-999'], dtype=object)
```

Change the values of the company_size column from 0 to 7 where e0 is <10 and 7 is 10000+. The order of the numbers should be based on the values of the column-like an ordinary variable.

[149]:
```
df['company_size'] = df['company_size'].replace({'<10':0, '10/49':1, '50-99':2,␣
↪'100-500':3, '500-999':4, '1000-4999':5, '5000-9999':6, '10000+':7})
```

Show the updated unique values

[150]:
```
df['company_size'].unique()
```

[150]:
```
array([2, 0, 6, 5, 1, 3, 7, 4])
```

Last_new_job:

Show the unique values of the last_new_job column

[151]:
```
df['last_new_job'].unique()
```

[151]:
```
array(['>4', '4', '1', '3', '2', 'never'], dtype=object)
```

Convert the values of this column to never->0, 1->1,….>4 –>5

[152]:
```
df['last_new_job'] = df['last_new_job'].replace({'never':0, '1':1, '2':2, '3':
↪3, '4':4, '>4':5})
```

Show the updated values

[153]:
```
df['last_new_job'].unique()
```

[153]:
```
array([5, 4, 1, 3, 2, 0])
```

Other columns:

Show the unique values of company_type, major_descipline, enrolled_university, relevant_experience, gender, and updated city column

[154]:
```
df['company_type'].unique()
```

[154]:
```
array(['Pvt Ltd', 'Funded Startup', 'Early Stage Startup',
       'Public Sector', 'NGO', 'Other'], dtype=object)
```

[155]:
```
df['major_discipline'].unique()
```

[155]:
```
array(['STEM', 'Humanities', 'Business Degree', 'Other', 'No Major',
       'Arts'], dtype=object)
```

[156]:
```
df['enrolled_university'].unique()
```

[156]:
```
array(['no_enrollment', 'Part time course', 'Full time course'],
      dtype=object)
```

[157]:
```
df['relevent_experience'].unique()
```

[157]:
```
array(['No relevent experience', 'Has relevent experience'], dtype=object)
```

[158]:
```
df['gender'].unique()
```

[158]:
```
array(['Male', 'Female', 'Other'], dtype=object)
```

[159]:
```
df['city'].unique()
```

[159]:
```
array(['city_others', 'city_103', 'city_114', 'city_21', 'city_16'],
      dtype=object)
```

As one-hot encoding is a bit strict, use panda's get_dummies function to create binary columns for the values of the following columns:

company_tye

major_descipline

enrolled_university

relevant_eperience

gender

updated city column

[160]:
```
df = pd.get_dummies(df, prefix='company_type', prefix_sep='.',
                            columns=['company_type'])
```

[161]:
```
df = pd.get_dummies(df, prefix='major_discipline', prefix_sep='.',
                            columns=['major_discipline'])
```

[162]:
```
df = pd.get_dummies(df, prefix='enrolled_university', prefix_sep='.',
                            columns=['enrolled_university'])
```

[163]:
```
df = pd.get_dummies(df, prefix='relevent_experience', prefix_sep='.',
                            columns=['relevent_experience'])
```

[164]:
```
df = pd.get_dummies(df, prefix='gender', prefix_sep='.',
                            columns=['gender'])
```

[165]:
```
df = pd.get_dummies(df, prefix='city', prefix_sep='.',
                            columns=['city'])
```

SHow the top 5 and last 5 rows to show that the table has changed

[166]:
```
pd.set_option('display.max_columns', None)
df.head()
```

[166]:

	Unnamed: 0	enrollee_id	city_development_index	education_level	experience	company_size	last_new_job	training_hours	target	company_type.Early Stage Startup	company_type.Funded Startup	company_type.NGO	company_type.Other	company_type.Public Sector	company_type.Pvt Ltd	major_discipline.Arts	major_discipline.Business Degree	major_discipline.Humanities	major_discipline.No Major	major_discipline.Other	major_discipline.STEM
0	1	29725	0.776	0	15.0	2	5	47	0.0	0	0	0	0	0	1	0	0	0	0	0	1
1	4	666	0.767	1	21.0	2	4	8	0.0	0	1	0	0	0	0	0	0	0	0	0	1
2	7	402	0.762	0	13.0	0	5	18	1.0	0	0	0	0	0	1	0	0	0	0	0	1
3	8	27107	0.920	0	7.0	2	1	46	1.0	0	0	0	0	0	1	0	0	0	0	0	1
4	11	23853	0.920	0	5.0	6	1	108	0.0	0	0	0	0	0	1	0	0	0	0	0	1

```
1                          0                      0                      1
2                          0                      0                      1
3                          0                      0                      1
4                          0                      0                      1

   enrolled_university.Full time course  enrolled_university.Part time course  \
0                                     0                                     0
1                                     0                                     0
2                                     0                                     0
3                                     0                                     0
4                                     0                                     0

   enrolled_university.no_enrollment  \
0                                  1
1                                  1
2                                  1
3                                  1
4                                  1

   relevent_experience.Has relevent experience  \
0                                            0
1                                            1
2                                            1
3                                            1
4                                            1

   relevent_experience.No relevent experience  gender.Female  gender.Male  \
0                                           1              0            1
1                                           0              0            1
2                                           0              0            1
3                                           0              0            1
4                                           0              0            1

   gender.Other  city.city_103  city.city_114  city.city_16  city.city_21  \
0             0              0              0             0             0
1             0              0              0             0             0
2             0              0              0             0             0
3             0              1              0             0             0
4             0              1              0             0             0

   city.city_others
0                 1
1                 1
2                 1
3                 0
4                 0
```

[167]:
```
df.tail()
```

[167]:

	Unnamed: 0	enrollee_id	city_development_index	education_level	experience	company_size	last_new_job	training_hours	target	company_type.Early Stage Startup	company_type.Funded Startup	company_type.NGO	company_type.Other	company_type.Public Sector	company_type.Pvt Ltd	major_discipline.Arts	major_discipline.Business Degree	major_discipline.Humanities	major_discipline.No Major	major_discipline.Other
8950	19147	21319	0.624	0	1.0	3	1	52	1.0	0	0	0	0	0	1	0	0	0	0	0
8951	19149	251	0.920	1	9.0	2	1	36	1.0	0	0	0	0	0	1	0	0	0	0	0
8952	19150	32313	0.920	0	10.0	3	3	23	0.0	0	0	0	0	1	0	0	0	0	0	0
8953	19152	29754	0.920	0	7.0	1	1	25	0.0	0	1	0	0	0	0	0	0	1	0	0
8954	19155	24576	0.920	0	21.0	2	4	44	0.0	0	0	0	0	0	1	0	0	0	0	0

```
8952                          0                         0
8953                          0                         0
8954                          0                         0

      major_discipline.STEM  enrolled_university.Full time course  \
8950                      1                                     1
8951                      1                                     0
8952                      1                                     0
8953                      0                                     0
8954                      1                                     0

      enrolled_university.Part time course  enrolled_university.no_enrollment  \
8950                                     0                                   0
8951                                     0                                   1
8952                                     0                                   1
8953                                     0                                   1
8954                                     0                                   1

      relevent_experience.Has relevent experience  \
8950                                            0
8951                                            1
8952                                            1
8953                                            1
8954                                            1

      relevent_experience.No relevent experience  gender.Female  gender.Male  \
8950                                           1              0            1
8951                                           0              0            1
8952                                           0              1            0
8953                                           0              1            0
8954                                           0              0            1

      gender.Other  city.city_103  city.city_114  city.city_16  city.city_21  \
8950             0              0              0             0             1
8951             0              1              0             0             0
8952             0              0              0             0             0
8953             0              1              0             0             0
8954             0              1              0             0             0

      city.city_others
8950                 0
8951                 0
8952                 1
8953                 0
8954                 0
```

Also, show the shape of the table

[168]:
```
print("Rows: ", len(df))
print("Columns: ", len(df.columns))
```

```
Rows:  8955
Columns:  34
```

Drop the enrollee_id and any duplicate columns (if you have multiple city column one with actual and one with updated, then remove the actual one)

[169]:
```
df = df.drop(columns = ['enrollee_id'])
```

Use sklearn.preprocessing's MinMaxScaler to perform min max scaling to all the columns (see documentation on how to use it)

[170]:
```
scaler = MinMaxScaler()
normData = pd.DataFrame(scaler.fit_transform(df), index=df.index, columns=df.
↪columns)
```

Show sample records that show some the scaled records

[171]:
```
normData
```

[171]:

	Unnamed: 0	city_development_index	education_level	experience	\
0	0.000000	0.654691	0.0	0.714286	
1	0.000157	0.636727	0.5	1.000000	
2	0.000313	0.626747	0.0	0.619048	
3	0.000365	0.942116	0.0	0.333333	
4	0.000522	0.942116	0.0	0.238095	
…	…	…	…	…	
8950	0.999582	0.351297	0.0	0.047619	
8951	0.999687	0.942116	0.5	0.428571	
8952	0.999739	0.942116	0.0	0.476190	
8953	0.999843	0.942116	0.0	0.333333	
8954	1.000000	0.942116	0.0	1.000000	

	company_size	last_new_job	training_hours	target	\
0	0.285714	1.0	0.137313	0.0	
1	0.285714	0.8	0.020896	0.0	
2	0.000000	1.0	0.050746	1.0	
3	0.285714	0.2	0.134328	1.0	
4	0.857143	0.2	0.319403	0.0	
…	…	…	…	…	
8950	0.428571	0.2	0.152239	1.0	
8951	0.285714	0.2	0.104478	1.0	
8952	0.428571	0.6	0.065672	0.0	
8953	0.142857	0.2	0.071642	0.0	
8954	0.285714	0.8	0.128358	0.0	

	company_type.Early Stage Startup	company_type.Funded Startup	\

```
0                                0.0                                0.0
1                                0.0                                1.0
2                                0.0                                0.0
3                                0.0                                0.0
4                                0.0                                0.0
…                                …                                  …
8950                             0.0                                0.0
8951                             0.0                                0.0
8952                             0.0                                0.0
8953                             0.0                                1.0
8954                             0.0                                0.0

      company_type.NGO  company_type.Other  company_type.Public Sector  \
0                  0.0                 0.0                         0.0
1                  0.0                 0.0                         0.0
2                  0.0                 0.0                         0.0
3                  0.0                 0.0                         0.0
4                  0.0                 0.0                         0.0
…                  …                   …                           …
8950               0.0                 0.0                         0.0
8951               0.0                 0.0                         0.0
8952               0.0                 0.0                         1.0
8953               0.0                 0.0                         0.0
8954               0.0                 0.0                         0.0

      company_type.Pvt Ltd  major_discipline.Arts  \
0                      1.0                    0.0
1                      0.0                    0.0
2                      1.0                    0.0
3                      1.0                    0.0
4                      1.0                    0.0
…                      …                      …
8950                   1.0                    0.0
8951                   1.0                    0.0
8952                   0.0                    0.0
8953                   0.0                    0.0
8954                   1.0                    0.0

      major_discipline.Business Degree  major_discipline.Humanities  \
0                                  0.0                          0.0
1                                  0.0                          0.0
2                                  0.0                          0.0
3                                  0.0                          0.0
4                                  0.0                          0.0
…                                  …                            …
8950                               0.0                          0.0
8951                               0.0                          0.0
```

```
8952                                0.0                                0.0
8953                                0.0                                1.0
8954                                0.0                                0.0

      major_discipline.No Major  major_discipline.Other  \
0                           0.0                     0.0
1                           0.0                     0.0
2                           0.0                     0.0
3                           0.0                     0.0
4                           0.0                     0.0
…                             …                       …
8950                        0.0                     0.0
8951                        0.0                     0.0
8952                        0.0                     0.0
8953                        0.0                     0.0
8954                        0.0                     0.0

      major_discipline.STEM  enrolled_university.Full time course  \
0                       1.0                                   0.0
1                       1.0                                   0.0
2                       1.0                                   0.0
3                       1.0                                   0.0
4                       1.0                                   0.0
…                         …                                   …
8950                    1.0                                   1.0
8951                    1.0                                   0.0
8952                    1.0                                   0.0
8953                    0.0                                   0.0
8954                    1.0                                   0.0

      enrolled_university.Part time course  enrolled_university.no_enrollment  \
0                                      0.0                                1.0
1                                      0.0                                1.0
2                                      0.0                                1.0
3                                      0.0                                1.0
4                                      0.0                                1.0
…                                        …                                  …
8950                                   0.0                                0.0
8951                                   0.0                                1.0
8952                                   0.0                                1.0
8953                                   0.0                                1.0
8954                                   0.0                                1.0

      relevent_experience.Has relevent experience  \
0                                             0.0
1                                             1.0
2                                             1.0
```

```
3                                                1.0
4                                                1.0
…                                                …
8950                                             0.0
8951                                             1.0
8952                                             1.0
8953                                             1.0
8954                                             1.0

      relevent_experience.No relevent experience  gender.Female  gender.Male  \
0                                                1.0            0.0          1.0
1                                                0.0            0.0          1.0
2                                                0.0            0.0          1.0
3                                                0.0            0.0          1.0
4                                                0.0            0.0          1.0
…                                                …              …            …
8950                                             1.0            0.0          1.0
8951                                             0.0            0.0          1.0
8952                                             0.0            1.0          0.0
8953                                             0.0            1.0          0.0
8954                                             0.0            0.0          1.0

      gender.Other  city.city_103  city.city_114  city.city_16  city.city_21  \
0              0.0            0.0            0.0           0.0           0.0
1              0.0            0.0            0.0           0.0           0.0
2              0.0            0.0            0.0           0.0           0.0
3              0.0            1.0            0.0           0.0           0.0
4              0.0            1.0            0.0           0.0           0.0
…              …              …              …             …             …
8950           0.0            0.0            0.0           0.0           1.0
8951           0.0            1.0            0.0           0.0           0.0
8952           0.0            0.0            0.0           0.0           0.0
8953           0.0            1.0            0.0           0.0           0.0
8954           0.0            1.0            0.0           0.0           0.0

      city.city_others
0                  1.0
1                  1.0
2                  1.0
3                  0.0
4                  0.0
…                  …
8950               0.0
8951               0.0
8952               1.0
8953               0.0
8954               0.0
```

```
[8955 rows x 33 columns]
```

Move the target column to the last column of the data frame and show that it has changed

[172]:
```
#Already the last column
targetcol = normData.pop('target')
normData.insert(32, 'target', targetcol)
normData.head()
```

[172]:

	Unnamed: 0	city_development_index	education_level	experience	company_size	last_new_job	training_hours	company_type.Early Stage Startup	company_type.Funded Startup	company_type.NGO	company_type.Other	company_type.Public Sector	company_type.Pvt Ltd	major_discipline.Arts	major_discipline.Business Degree	major_discipline.Humanities
0	0.000000	0.654691	0.0	0.714286	0.285714	1.0	0.137313	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	0.000157	0.636727	0.5	1.000000	0.285714	0.8	0.020896	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.000313	0.626747	0.0	0.619048	0.000000	1.0	0.050746	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
3	0.000365	0.942116	0.0	0.333333	0.285714	0.2	0.134328	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.000522	0.942116	0.0	0.238095	0.857143	0.2	0.319403	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

```
2                                0.0                                0.0
3                                0.0                                0.0
4                                0.0                                0.0

   major_discipline.No Major  major_discipline.Other  major_discipline.STEM  \
0                        0.0                     0.0                    1.0
1                        0.0                     0.0                    1.0
2                        0.0                     0.0                    1.0
3                        0.0                     0.0                    1.0
4                        0.0                     0.0                    1.0

   enrolled_university.Full time course  enrolled_university.Part time course  \
0                                   0.0                                   0.0
1                                   0.0                                   0.0
2                                   0.0                                   0.0
3                                   0.0                                   0.0
4                                   0.0                                   0.0

   enrolled_university.no_enrollment  \
0                                1.0
1                                1.0
2                                1.0
3                                1.0
4                                1.0

   relevent_experience.Has relevent experience  \
0                                          0.0
1                                          1.0
2                                          1.0
3                                          1.0
4                                          1.0

   relevent_experience.No relevent experience  gender.Female  gender.Male  \
0                                         1.0            0.0          1.0
1                                         0.0            0.0          1.0
2                                         0.0            0.0          1.0
3                                         0.0            0.0          1.0
4                                         0.0            0.0          1.0

   gender.Other  city.city_103  city.city_114  city.city_16  city.city_21  \
0           0.0            0.0            0.0           0.0           0.0
1           0.0            0.0            0.0           0.0           0.0
2           0.0            0.0            0.0           0.0           0.0
3           0.0            1.0            0.0           0.0           0.0
4           0.0            1.0            0.0           0.0           0.0

   city.city_others  target
```

```
0               1.0     0.0
1               1.0     0.0
2               1.0     1.0
3               0.0     1.0
4               0.0     0.0
```

5 3.) X/Y and Training/Test Split with stratified sampling and SMOTE

Copy all the features into X and the target to Y

[173]:
```
y = normData['target']
normData = normData.drop(columns = ['target'])
x = normData
```

Show the ratio of 1 and 0 in Y

[174]:
```
print("Total count: ")
print(y.value_counts())
print(" ")
print("Ratios: ")
print("1/0: ", y.value_counts()[1]/y.value_counts()[0])
print("0/1: ", y.value_counts()[0]/y.value_counts()[1])
```

```
Total count:
0.0    7472
1.0    1483
Name: target, dtype: int64

Ratios:
1/0:  0.19847430406852248
0/1:  5.0384356035064055
```

Use sklearn's train_test_split to split the data set into training and test sets. There should be 30% records in the test set. The random_stat should be 0. As we want to have the same ratio of 0 and 1 in the test set, use the stratify parameter to the Y.

[175]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,␣
↪random_state = 0, stratify = y)
```

Show the ratio of 0 and 1 in y_train and then y_test

[176]:
```
print("Total y_train count: ")
print(y_train.value_counts())
print(" ")
print("Ratios: ")
print("1/0: ", y_train.value_counts()[1]/y_train.value_counts()[0])
print("0/1: ", y_train.value_counts()[0]/y_train.value_counts()[1])
```

```
Total y_train count:
0.0    5230
1.0    1038
Name: target, dtype: int64

Ratios:
1/0:  0.19847036328871892
0/1:  5.038535645472062
```

[177]:
```
print("Total y_test count: ")
print(y_test.value_counts())
print(" ")
print("Ratios: ")
print("1/0: ", y_test.value_counts()[1]/y_test.value_counts()[0])
print("0/1: ", y_test.value_counts()[0]/y_test.value_counts()[1])
```

```
Total y_test count:
0.0    2242
1.0     445
Name: target, dtype: int64

Ratios:
1/0:  0.19848349687778769
0/1:  5.038202247191011
```

Use imblearn's SMOTE to balance the x_train

[178]:
```
sm = SMOTE(random_state=54)
#x_res, y_res = sm.fit_resample(x_train, y_train)
x_train2, y_train2 = SMOTE().fit_resample(x_train, y_train)
```

Show the ratio of 0 and 1 in Y_train after rebalancing. (do you have 50% of each class now?)

[179]:
```
print("Total y_train count: ")
print(y_train2.value_counts())
print(" ")
print("Ratios: ")
print("1/0: ", y_train2.value_counts()[1]/y_train2.value_counts()[0])
print("0/1: ", y_train2.value_counts()[0]/y_train2.value_counts()[1])
```

```
Total y_train count:
0.0    5230
1.0    5230
Name: target, dtype: int64

Ratios:
1/0:  1.0
0/1:  1.0
```

Answer: Yes we do.

6 4.) PCA and Logistic Regression

As we have many features now, we would like to do principal component analysis (you have learned it in datacamp). As part of ti, create pipeline to find how many dimensions give you the best logistic regression model.

[180]:
```
# define the pipeline
steps = [('pca', PCA(n_components=10)), ('m', LogisticRegression())]
model = Pipeline(steps=steps)
# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, x_test, y_test, scoring='accuracy', cv=cv,␣
↪n_jobs=-1, error_score='raise')
# report performance
print("Accuracy: ", n_scores.mean(), "(", n_scores.std(), ")")
```

```
Accuracy:  0.8553505705672383 ( 0.013597068084271434 )
```

[181]:
```
# get a list of models to evaluate
def get_models():
        models = dict()
        for i in range(1, 32):
                steps = [('pca', PCA(n_components=i)), ('m',␣
↪LogisticRegression())]
                models[str(i)] = Pipeline(steps=steps)
        return models

# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
        cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
        scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv,␣
↪n_jobs=-1, error_score='raise')
        return scores

# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
        scores = evaluate_model(model, x_train2, y_train2)
        results.append(scores)
        names.append(name)
        print('>%s %.3f (%.3f)' % (name, scores.mean(), scores.std()))
# plot model performance for comparison
plt.boxplot(results, labels=names, showmeans=True)
plt.xticks(rotation=45)
sns.set(rc={'figure.figsize':(10, 3)})
plt.show()
```

>1 0.677 (0.018)
>2 0.688 (0.015)
>3 0.689 (0.015)
>4 0.691 (0.015)
>5 0.704 (0.014)
>6 0.703 (0.014)
>7 0.706 (0.015)
>8 0.719 (0.013)
>9 0.719 (0.014)
>10 0.724 (0.012)
>11 0.726 (0.013)
>12 0.724 (0.012)
>13 0.724 (0.013)
>14 0.721 (0.013)
>15 0.721 (0.013)
>16 0.724 (0.013)
>17 0.724 (0.014)
>18 0.722 (0.014)
>19 0.724 (0.014)
>20 0.724 (0.014)
>21 0.731 (0.012)
>22 0.730 (0.012)
>23 0.730 (0.012)
>24 0.731 (0.012)
>25 0.731 (0.012)
>26 0.731 (0.012)
>27 0.731 (0.012)
>28 0.731 (0.012)
>29 0.731 (0.012)
>30 0.731 (0.012)
>31 0.731 (0.012)

Show the confusion matrix and interpret the numbers in the confusion matrix

[183]:
```
print("Confusion Matrix: ")
tn, fp, fn, tp = confusion_matrix(y_pred, y_test).ravel()
confusion_matrix(y_test, y_pred)
```

```
Confusion Matrix:
```

[183]:
```
array([[1888,  354],
       [ 190,  255]])
```

[184]:
```
print("TP: ", tp)
print("FP: ", fp)
print("TN: ", tn)
print("FN: ", fn)
print("Missclassification count: ", fp+fn)
```

```
TP:  255
FP:  190
TN:  1888
FN:  354
Missclassification count:  544
```

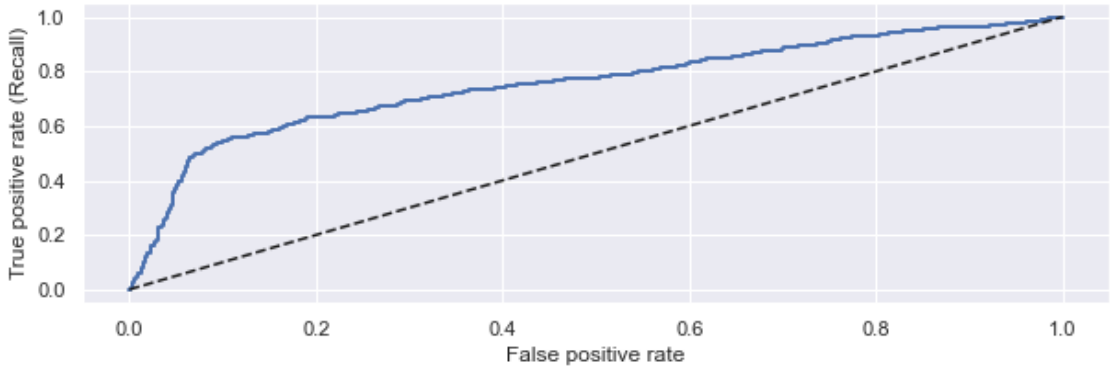
Show precision, recall, and f1 score

[185]:
```
print("Precision Score: ", precision_score(y_test, y_pred))
print("Recall Score: ", recall_score(y_test, y_pred))
print("F1 Score: ", f1_score(y_test, y_pred))
```

```
Precision Score:  0.4187192118226601
Recall Score:  0.5730337078651685
F1 Score:  0.4838709677419355
```

Plot ROC curve and find AUC

[186]:
```
y_scores = cross_val_predict(model, x_test, y_test, cv = 3, method =␣
 ↪"decision_function")
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```

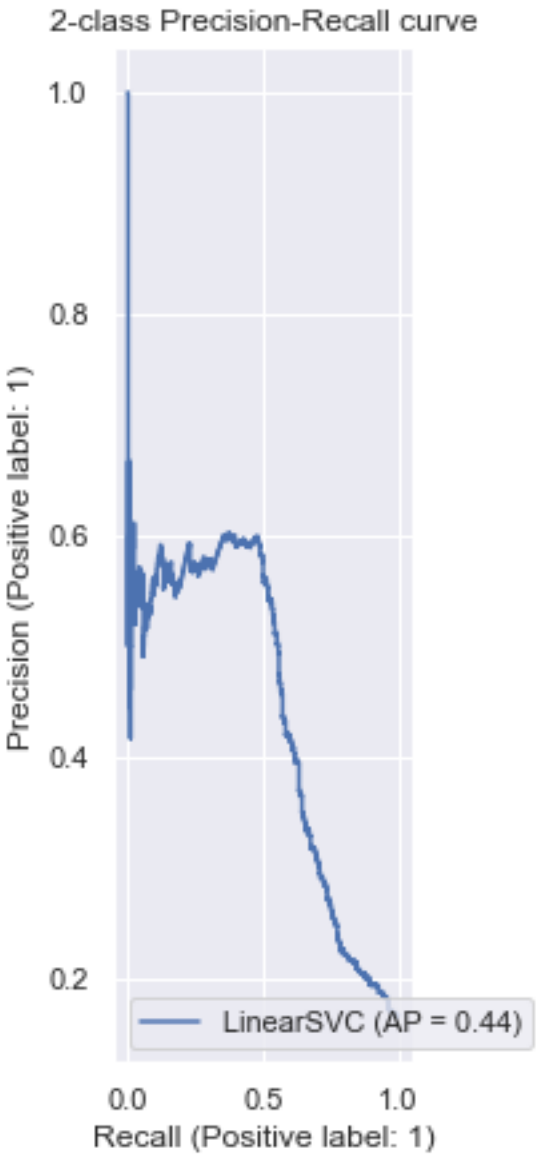


[187]:
```
roc_auc_score(y_test, y_pred)
```

[187]:
```
0.7075694855115315
```

Plot precision-recall curve for different thresholds and discuss the plot

[188]:
```
display = PrecisionRecallDisplay.from_predictions(y_test, y_scores,␣
↪name="LinearSVC")
_ = display.ax_.set_title("2-class Precision-Recall curve")
sns.set(rc={'figure.figsize':(6, 7)})
```



7 Answer:

It appears to perform its best when precision is at around 0.6 while recall is between 0.4 and 0.6

8 5.) Softmaxt regression:

How softmax regression is related to logistic regression? What library can you use for softmax regression?

9 Answer:

TensorFlow is a library that supports softmax regression. Softmax and logistic are both methods of classification. Their differences is that logistic is good for binary classification, while softmax is good for multiple classes.

10 6.) KNN

Use sklearn's KNN classifier to train (with k= 10) and predict the model based on the unbalanced training set and test it and show the confusion matrix and classification report

[189]:
```
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(x_train, y_train)
```

[189]:
```
KNeighborsClassifier(n_neighbors=10)
```

[190]:
```
y_pred1 = classifier.predict(x_test)
print("Confusion Matrix: ")
tn, fp, fn, tp = confusion_matrix(y_pred1, y_test).ravel()
print(confusion_matrix(y_test, y_pred1))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred1))
print("Missclassification count: ", fp+fn)
```

```
Confusion Matrix:
[[2155   87]
 [ 312  133]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.96      0.92      2242
         1.0       0.60      0.30      0.40       445

    accuracy                           0.85      2687
   macro avg       0.74      0.63      0.66      2687
weighted avg       0.83      0.85      0.83      2687

Missclassification count:  399
```

Use sklearn's KNN classifier to train (with k= 10) and predict the model based on the rebalanced training set and test it and show the confusion matrix and classification report

[191]:
```
classifier = KNeighborsClassifier(n_neighbors=10)
classifier.fit(x_train2, y_train2)
```

[191]:
```
KNeighborsClassifier(n_neighbors=10)
```

[192]:
```
y_pred2 = classifier.predict(x_test)
print("Confusion Matrix: ")
tn, fp, fn, tp = confusion_matrix(y_pred2, y_test).ravel()
print(confusion_matrix(y_test, y_pred2))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred2))
print("Missclassification count: ", fp+fn)
```

```
Confusion Matrix:
[[1716  526]
 [ 180  265]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.77      0.83      2242
         1.0       0.34      0.60      0.43       445

    accuracy                           0.74      2687
   macro avg       0.62      0.68      0.63      2687
weighted avg       0.81      0.74      0.76      2687

Missclassification count:  706
```

Use grid search to tune the following hyperparameters of KNN: number of neighbors (between 1 and 20), weights (uniform or distance), and metrics (Euclidean, Manhattan, or Minkowski)istance) to use for KNN. While creating an instance of GridSearchCV, use multiple evaluation metrics such as AUC and accuracy based on the example available

[ ]:
```
seed = 42
knn_params = {
    "n_neighbors": range(1, 30, 2),
    "weights": ["uniform", "distance"],
    "metric": ["euclidean", "manhattan", "minkowski"],
    #"algorithm": ["auto", "ball_tree", "kd_tree", "brute"],
    "leaf_size": range(1, 50, 5)
}

knn = KNeighborsClassifier()

#grid search
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=seed)
grid_search = GridSearchCV(estimator=knn, param_grid=knn_params, n_jobs=1,␣
 ↪cv=cv, scoring="accuracy", error_score=0)
grid_results = grid_search.fit(x_train2, y_train2)

#best model
```

```
final_model = knn.set_params(**grid_results.best_params_)
final_model.fit(x_train2, y_train2)
y_pred3 = final_model.predict(x_test)

#summarize results
print(classification_report(y_test, y_pred3))
print(confusion_matrix(y_test, y_pred3))
print(grid_results.best_params_)
```

The above grid search process can take a couple of minutes. After completing the process, print the best_params_

[193]:
```
print({'leaf_size': 1, 'metric': 'manhattan', 'n_neighbors': 1, 'weights':␣
↪'uniform'})
```

```
{'leaf_size': 1, 'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'uniform'}
```

Based on the result from grid search, use the parameters to train a model, test it with test set, and then print the confusion matrix and classification report. Also, show the AUC of ROC.

[194]:
```
classifier = KNeighborsClassifier(leaf_size = 1, metric = 'manhattan',␣
↪n_neighbors=1,  weights = 'uniform')
classifier.fit(x_train2, y_train2)
```

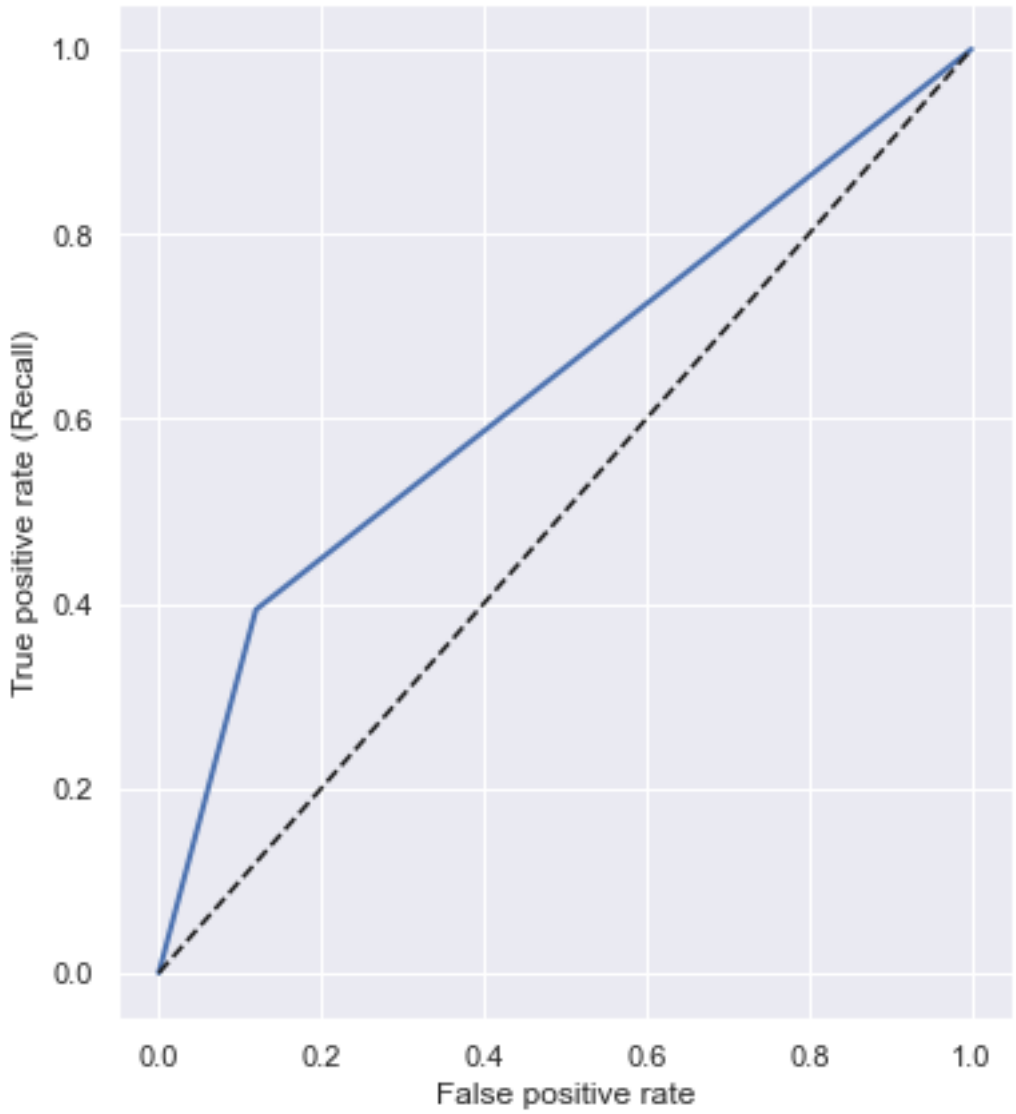
[194]:
```
KNeighborsClassifier(leaf_size=1, metric='manhattan', n_neighbors=1)
```

[195]:
```
y_pred4 = classifier.predict(x_test)
print("Confusion Matrix: ")
tn, fp, fn, tp = confusion_matrix(y_pred4, y_test).ravel()
print(confusion_matrix(y_test, y_pred4))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred4))
print(y_test.value_counts())
print("Missclassification count: ", fp+fn)
```

```
Confusion Matrix:
[[1883  359]
 [ 264  181]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.84      0.86      2242
         1.0       0.34      0.41      0.37       445

    accuracy                           0.77      2687
   macro avg       0.61      0.62      0.61      2687
weighted avg       0.79      0.77      0.78      2687
```

```
0.0    2242
1.0     445
Name: target, dtype: int64
Missclassification count:  623
```

[196]:
```
model = KNeighborsClassifier(leaf_size = 1, metric = 'manhattan',␣
 ↪n_neighbors=1,  weights = 'uniform')
y_scores = cross_val_predict(model, x_test, y_test, cv = 76)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[197]:
```
roc_auc_score(y_test, y_pred4)
```

[197]:
```
0.6233083422706451
```

Use PCA and based on that train model, test it and then print the confusion matrix and classification report. Also, show the AUC of ROC.

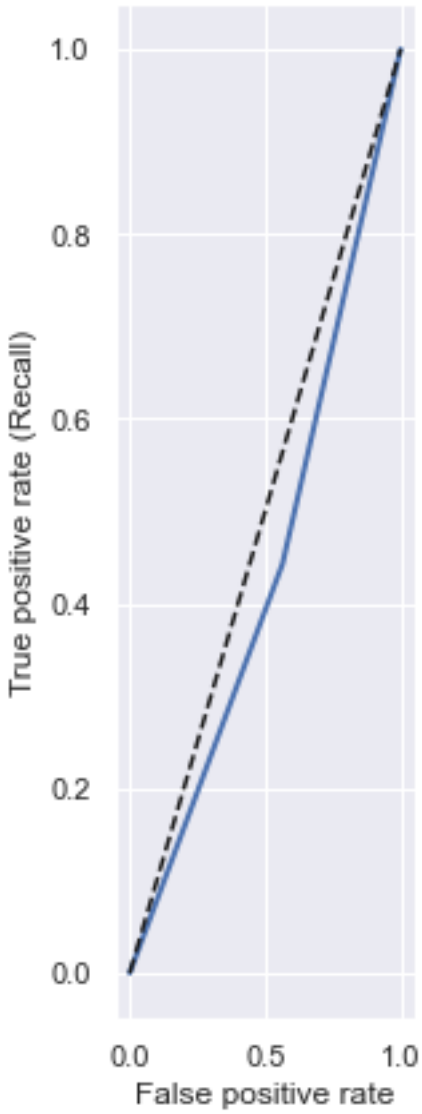
[198]:
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x_train2)
principalComponentstest = pca.fit_transform(x_test)
```

[199]:
```
classifier = KNeighborsClassifier(leaf_size = 1, metric = 'manhattan',␣
↪n_neighbors=1,  weights = 'uniform')
```

```
classifier.fit(principalComponents, y_train2)
y_pred5 = classifier.predict(principalComponentstest)
print("Confusion Matrix: ")
tn, fp, fn, tp = confusion_matrix(y_pred5, y_test).ravel()
print(confusion_matrix(y_test, y_pred5))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred5))
print("Missclassification count: ", fp+fn)
```

```
Confusion Matrix:
[[ 978 1264]
 [ 248  197]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.80      0.44      0.56      2242
         1.0       0.13      0.44      0.21       445

    accuracy                           0.44      2687
   macro avg       0.47      0.44      0.39      2687
weighted avg       0.69      0.44      0.50      2687

Missclassification count:  1512
```

[200]:
```
model = KNeighborsClassifier(leaf_size = 1, metric = 'manhattan',␣
 ↪n_neighbors=1,  weights = 'uniform')
y_scores = classifier.predict(principalComponentstest)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[201]:
```
roc_auc_score(y_test, y_pred5)
```

[201]:
```
0.43945714600727676
```

A short discussion on the 4 models and their differences.

11 Answer

PCA with KNN was absolutely terrible while PCA with logistic seems to have performed the best. None gave very impressive results, though.

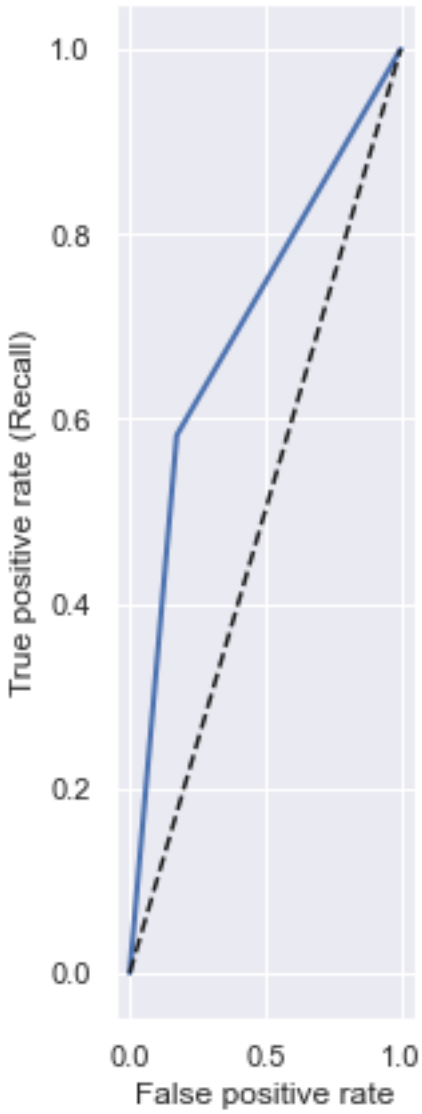
12 7.) Naive Bayes

Train a model with GaussianNB, test it and then print the confusion matrix and classification report. Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclassification.

[202]:
```
clf = GaussianNB()
clf.fit(x_train2, y_train2)
y_pred6 = clf.predict(x_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred6).ravel()
print(confusion_matrix(y_test, y_pred6))
print(classification_report(y_test, y_pred6))
```

```
[[1063 1179]
 [ 107  338]]
              precision    recall  f1-score   support

         0.0       0.91      0.47      0.62      2242
         1.0       0.22      0.76      0.34       445

    accuracy                           0.52      2687
   macro avg       0.57      0.62      0.48      2687
weighted avg       0.79      0.52      0.58      2687
```

[203]:
```
model = GaussianNB()
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[204]:
```
roc_auc_score(y_test, y_pred6)
```

[204]:
```
0.6168404013270655
```

[205]:
```
print("Missclassification count: ", fp+fn)
```

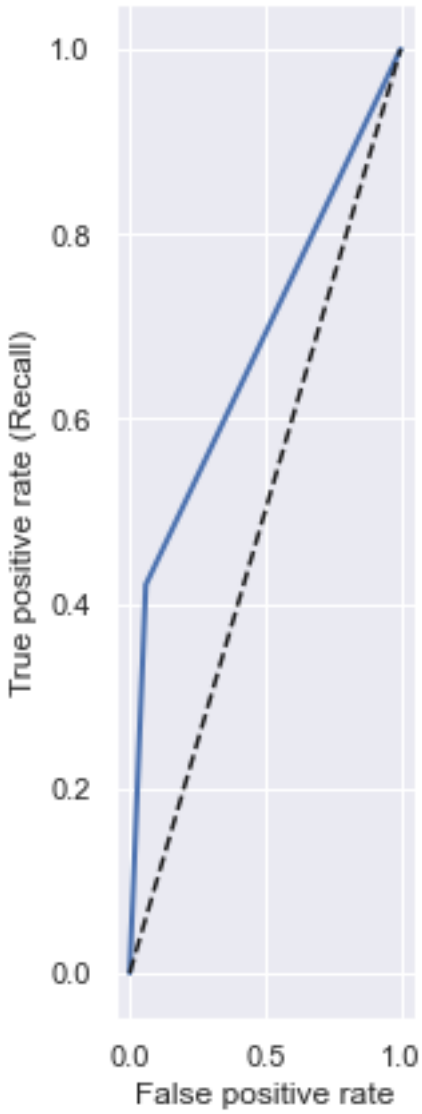
```
Missclassification count:  1286
```

Train a model with CategoricalNB, test it and then print the confusion matrix and classification report. Also, plot ROC curve, and show the AUC of ROC and the count of the number of misclassification.

[206]:
```
clf = CategoricalNB()
clf.fit(x_train2, y_train2)
y_pred7 = clf.predict(x_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred7).ravel()
print(confusion_matrix(y_test, y_pred7))
print(classification_report(y_test, y_pred7))
```

```
[[1905  337]
 [ 205  240]]
              precision    recall  f1-score   support

         0.0       0.90      0.85      0.88      2242
         1.0       0.42      0.54      0.47       445

    accuracy                           0.80      2687
   macro avg       0.66      0.69      0.67      2687
weighted avg       0.82      0.80      0.81      2687
```

[207]:
```
model = CategoricalNB()
y_scores = cross_val_predict(model, x_test, y_test, cv = 3)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[208]:
```
roc_auc_score(y_test, y_pred7)
```

[208]:
```
0.6945068107327927
```

[209]:
```
print("Missclassification rate: ", fp+fn)
```

```
Missclassification rate:  542
```

13 8.) Support Vector Machine

Build a support vector machine model using SVC. Use grid search to tune some parameters and then based on that show the best parameters found

[210]:
```
model = SVC()
model.fit(x_train2, y_train2)
y_pred8 = clf.predict(x_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred8).ravel()
print(confusion_matrix(y_test, y_pred8))
print(classification_report(y_test, y_pred8))
```

```
[[1905  337]
 [ 205  240]]
              precision    recall  f1-score   support

         0.0       0.90      0.85      0.88      2242
         1.0       0.42      0.54      0.47       445

    accuracy                           0.80      2687
   macro avg       0.66      0.69      0.67      2687
weighted avg       0.82      0.80      0.81      2687
```

[ ]:
```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(x_train2, y_train2)
print(grid.best_params_)
print(grid.best_estimator_)
```

[211]:
```
#Best Parameters Are: {'C': 1000, 'gamma': 1, 'kernel': 'rbf'}
```

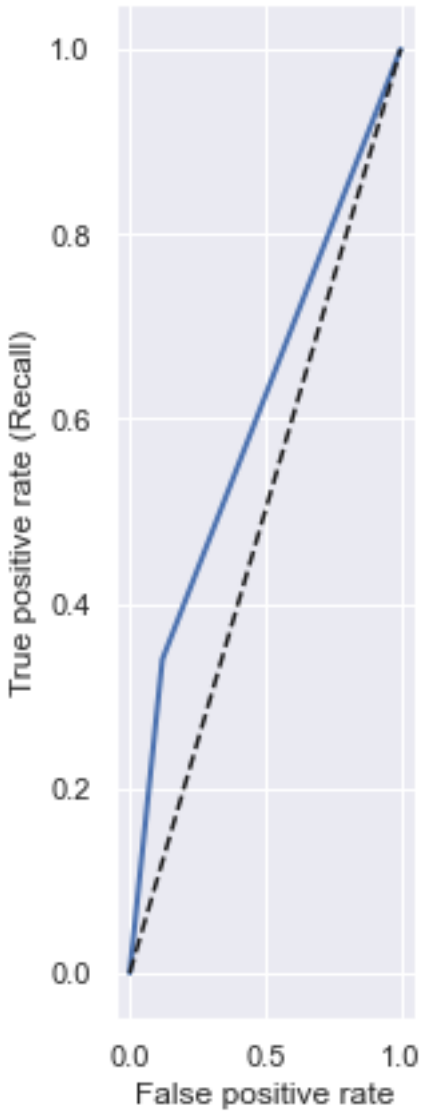
Test the model and print the confusion matrix and classification report. Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclassification.

[212]:
```
model = SVC(C= 1000, gamma = 1, kernel= 'rbf')
SVC(C=1000, gamma=1)
model.fit(x_train2, y_train2)
y_pred9 = clf.predict(x_test)
tn, fp, fn, tp = confusion_matrix(y_test, y_pred9).ravel()
print(confusion_matrix(y_test, y_pred9))
print(classification_report(y_test, y_pred9))
```

```
[[1905  337]
 [ 205  240]]
              precision    recall  f1-score   support

         0.0       0.90      0.85      0.88      2242
```

```
           1.0       0.42      0.54      0.47       445

      accuracy                           0.80      2687
     macro avg       0.66      0.69      0.67      2687
  weighted avg       0.82      0.80      0.81      2687
```

[213]:
```
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[214]:
```
roc_auc_score(y_test, y_pred9)
```

[214]: 0.6945068107327927

[215]:
```
print("Missclassification count: ", fp+fn)
```

```
Missclassification count:  542
```

14 9.) Decision Tree

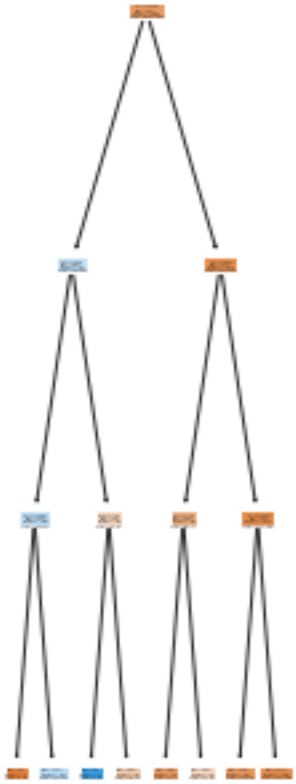
Build a decision tree model using sklearns DecisionTreeClassifier. Use the unbalanced training set, entropy as the criterion. Try with different max_depth (or use grid search). After building model, test it and print the confusion matrix and classification report. Also, plot ROC curve and show

the AUC of ROC, and the count of the number of misclassification. Show the decision tree. (you can simply import tree from sklearn and call tree.plot_tree with your model and the call plt.show. At the beginning of this process, use plt.figure to change the figsize)

[ ]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
param_dict = {"max_depth":range(1, 50)}
grid = GridSearchCV(decision_tree, param_grid=param_dict, cv = 10, verbose = 1,␣
 ↪n_jobs = -1)
grid.fit(x_train, y_train)
```

[216]:
```
#Best max_depth parameter is 3
#grid.best_params_
```

[217]:
```
clf = tree.DecisionTreeClassifier(max_depth = 3)
clf = clf.fit(x_train, y_train)
clf.score(x_train, y_train)
y_pred10 = clf.predict(x_test)
plt.figure()
tree.plot_tree(clf, filled=True)
plt.figure(figsize=(20,20))
plt.show()
```



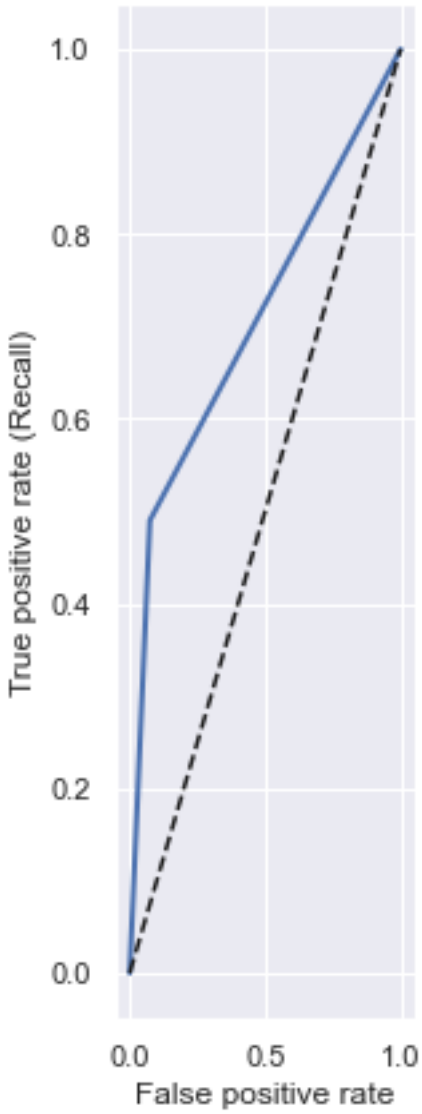
```
<Figure size 1440x1440 with 0 Axes>
```

[218]:
```
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred10))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred10))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred10).ravel()
```

```
Confusion Matrix:
[[2088  154]
 [ 224  221]]

Classification Report:
              precision    recall  f1-score   support
```

```
          0.0       0.90      0.93      0.92      2242
          1.0       0.59      0.50      0.54       445

     accuracy                           0.86      2687
    macro avg       0.75      0.71      0.73      2687
 weighted avg       0.85      0.86      0.85      2687
```

[219]:
```
model = tree.DecisionTreeClassifier(max_depth = 3)
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[220]:
```
roc_auc_score(y_test, y_pred10)
```

[220]: 0.7139702713267648

[221]:
```
print("Missclassification count: ", fp+fn)
```

```
Missclassification count:  378
```

Perform the same tasks as 9.1 with the balanced training set

[ ]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train2, y_train2)
param_dict = {"max_depth":range(1, 1000)}
```

```
grid = GridSearchCV(decision_tree, param_grid=param_dict, cv = 10, verbose = 1,␣
 ↪n_jobs = -1)
grid.fit(x_train2, y_train2)
```

[222]:
```
#Best max_depth parameter is 123
#grid.best_params_
```

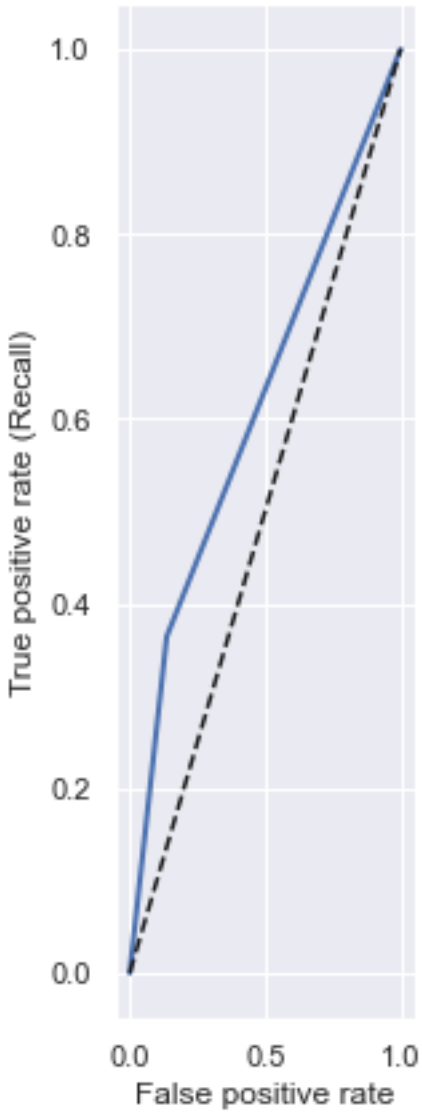
[223]:
```
clf = tree.DecisionTreeClassifier(max_depth = 123)
clf = clf.fit(x_train2, y_train2)
clf.score(x_train2, y_train2)
y_pred11 = clf.predict(x_test)
plt.figure()
tree.plot_tree(clf, filled=True)
plt.figure(figsize=(100,100))
plt.show()
```

```
<Figure size 7200x7200 with 0 Axes>
```

[224]:
```
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred11))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred11))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred11).ravel()
```

```
Confusion Matrix:
[[1862  380]
 [ 269  176]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.83      0.85      2242
         1.0       0.32      0.40      0.35       445

    accuracy                           0.76      2687
   macro avg       0.60      0.61      0.60      2687
weighted avg       0.78      0.76      0.77      2687
```

[225]:
```
model = tree.DecisionTreeClassifier(max_depth = 123)
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[226]:
```
roc_auc_score(y_test, y_pred11)
```

[226]:
```
0.6130070462768996
```

[227]:
```
print("Missclassification count: ", fp+fn)
```

```
Missclassification count:  649
```

Discuss any difference and also discuss part of the tree of 9.2

15 Answer:

The tree with the unscaled data was far smaller and easier to read. It also performed much better with a lesser missclassification count than with the scaled data.

16 10.) Random Forest

Use grid search to tune the max_depth, min_samples_leaf, and n_estimators

[ ]:
```
rfc = RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [200, 500],
    'max_depth' : [4,5,6,7,8],
    'min_samples_leaf': [5,10,20,50,100,200]
}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_train2, y_train2)
```

Print the best estimator

[228]:
```
#Best parameters are: {'max_depth': 8, 'min_samples_leaf': 5, 'n_estimators':␣
↪200}
print("{'max_depth': 8, 'min_samples_leaf': 5, 'n_estimators': 200}")
```

```
{'max_depth': 8, 'min_samples_leaf': 5, 'n_estimators': 200}
```

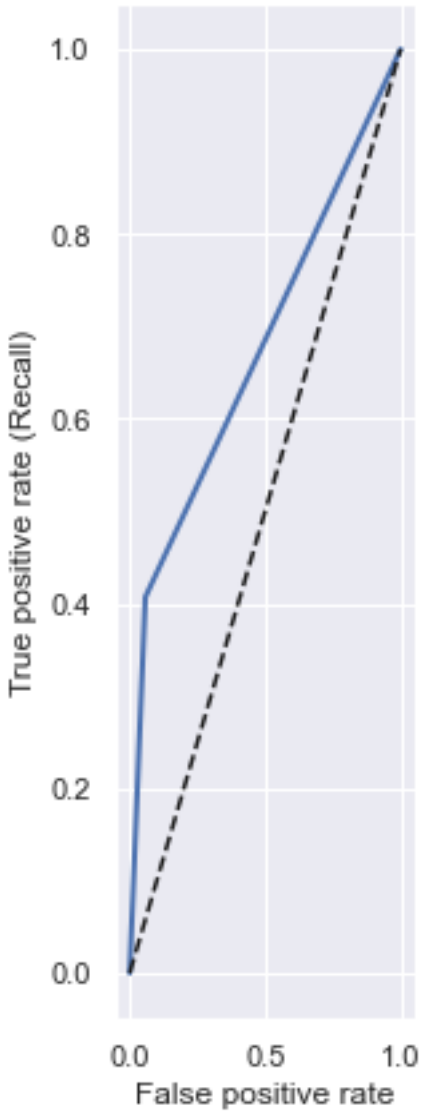
Train the model. After building the model, test it and print the confusion matrix and classification report. Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclassification.

[229]:
```
rfc1=RandomForestClassifier(random_state=42, n_estimators= 200, max_depth=8,␣
↪min_samples_leaf = 5)
rfc1.fit(x_train, y_train)
y_pred12 = rfc1.predict(x_test)
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred12))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred12))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred12).ravel()
```

```
Confusion Matrix:
[[2115  127]
 [ 259  186]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.89      0.94      0.92      2242
```

```
           1.0       0.59      0.42      0.49       445

      accuracy                           0.86      2687
     macro avg       0.74      0.68      0.70      2687
  weighted avg       0.84      0.86      0.85      2687
```

[230]:
```
model = RandomForestClassifier(random_state=42, max_features='auto',␣
 ↪n_estimators= 200, max_depth=8, criterion='gini')
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[231]:
```
roc_auc_score(y_test, y_pred12)
```

[231]: 0.6806658380859787

[232]:
```
print("Missclassification count: ", fp+fn)
```

```
Missclassification count:  386
```

17 11.) Boosting Algorithms

Train an AdaBoostClassifier model with some manual/grid search-based parameters and then test it and then print the confusion matrix and classification report. Also, plot ROC curve and show the AUC of ROC, and the count of the number of misclassification.

[ ]:
```
model = AdaBoostClassifier()
grid = dict()
grid['n_estimators'] = [10, 50, 100, 500]
grid['learning_rate'] = [0.0001, 0.001, 0.01, 0.1, 1.0]
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
g_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
 ↪scoring='accuracy')
g_result = g_search.fit(x_train2, y_train2)
```

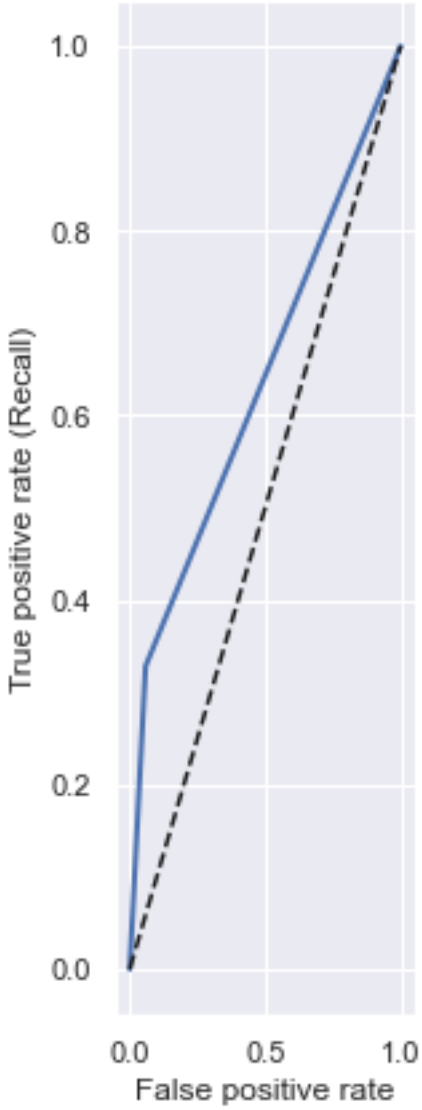
[233]:
```
#Best parameters are: {'learning_rate': 1.0, 'n_estimators': 500}
#g_result.best_params_
```

[234]:
```
model = AdaBoostClassifier(learning_rate = 1.0, n_estimators = 500)
model.fit(x_train2, y_train2)
y_pred13 = model.predict(x_test)
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred13))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred13))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred13).ravel()
```

```
Confusion Matrix:
[[2072  170]
 [ 207  238]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.92      0.92      2242
         1.0       0.58      0.53      0.56       445

    accuracy                           0.86      2687
   macro avg       0.75      0.73      0.74      2687
weighted avg       0.86      0.86      0.86      2687
```

[235]:
```
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
```

```
plt.show()
```



[236]:
```
roc_auc_score(y_test, y_pred13)
```

[236]:
```
0.7295031522817709
```

[237]:
```
print("Missclassification count: ", fp+fn)
```

```
Missclassification count:  377
```

Do the same for Gradient BoostingClassifier

[ ]:
```
parameters = {
    "loss":["deviance"],
    "learning_rate": [0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.2],
    "min_samples_split": np.linspace(0.1, 0.5, 12),
    "min_samples_leaf": np.linspace(0.1, 0.5, 12),
    "max_depth":[3,5,8],
    "max_features":["log2","sqrt"],
    "subsample":[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0],
    "n_estimators":[10]
    }
g_search = GridSearchCV(GradientBoostingClassifier(), parameters, cv=10,␣
 ↪n_jobs=-1)
g_score = g_search.fit(x_train2, y_train2)
```

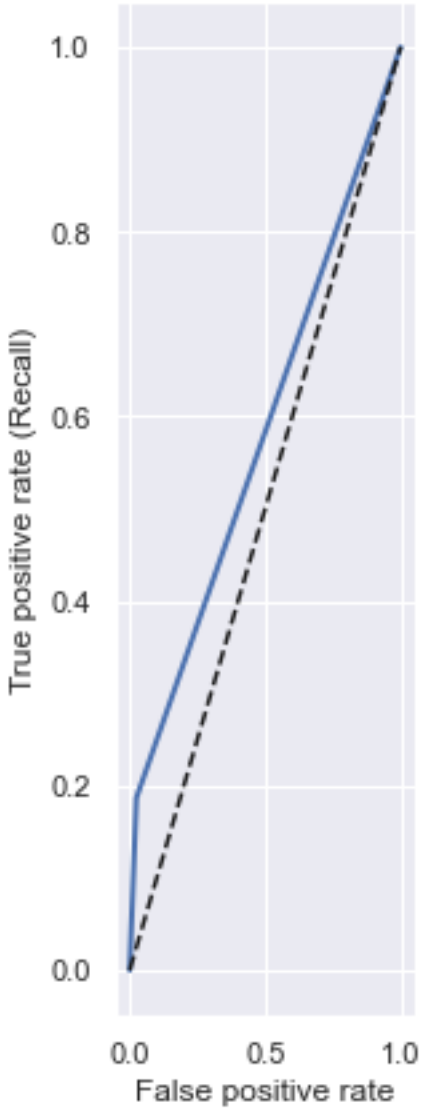
[238]:
```
#Best parameters are: 'learning_rate': 0.2, 'max_depth': 8, 'max_features':␣
 ↪'log2', 'min_samples_leaf': 0.1, 'min_samples_split': 0.2090909090909091,␣
 ↪'n_estimators': 10
#g_score.best_params_
```

{'learning_rate': 0.2, 'loss': 'deviance', 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_leaf': 0.1, 'min_samples_split': 0.13636363636363638, 'n_estimators': 10, 'subsample': 0.95}

[239]:
```
model = GradientBoostingClassifier(learning_rate = 0.2, loss = 'deviance',␣
 ↪max_depth = 5, max_features = 'sqrt', min_samples_leaf = 0.1,␣
 ↪min_samples_split =0.13636363636363638, n_estimators = 10, subsample = 0.95)
model.fit(x_train2, y_train2)
y_pred14 = model.predict(x_test)
print("Confusion Matrix: ")
print(confusion_matrix(y_test, y_pred14))
print()
print("Classification Report: ")
print(classification_report(y_test, y_pred14))
tn, fp, fn, tp = confusion_matrix(y_test, y_pred14).ravel()
```

```
Confusion Matrix:
[[1924  318]
 [ 178  267]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.92      0.86      0.89      2242
         1.0       0.46      0.60      0.52       445

    accuracy                           0.82      2687
   macro avg       0.69      0.73      0.70      2687
weighted avg       0.84      0.82      0.82      2687
```

[240]:
```
y_scores = cross_val_predict(model, x_test, y_test, cv = 6)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
def plot_roc_curve(fpr, tpr, label=None):
 plt.plot(fpr, tpr, linewidth=2, label=label)
 plt.plot([0, 1], [0, 1], 'k--') # dashed diagonal

plot_roc_curve(fpr, tpr)

plt.xlabel("False positive rate")
plt.ylabel("True positive rate (Recall)")
sns.set(rc={'figure.figsize':(2, 7)})
plt.show()
```



[241]:
```
roc_auc_score(y_test, y_pred14)
```

[241]:
```
0.7290811775200714
```

[242]:
```
print("Missclassification count: ", fp+fn)
```

```
Missclassification count:  496
```

18 12.) Finally, briefly discuss your finding such as which model could be most suitable for this given scenario and what could be your future work based on this experiment.

19 Answer

Judging by missclassification count alone, decision tree with the unscaled data had the best one and was the easiest to look at as well. I'd probably like to use at least a couple classification models in the future, but I would lean towards using decision tree classifier if I needed to classify something.

[ ]:
```

```