main

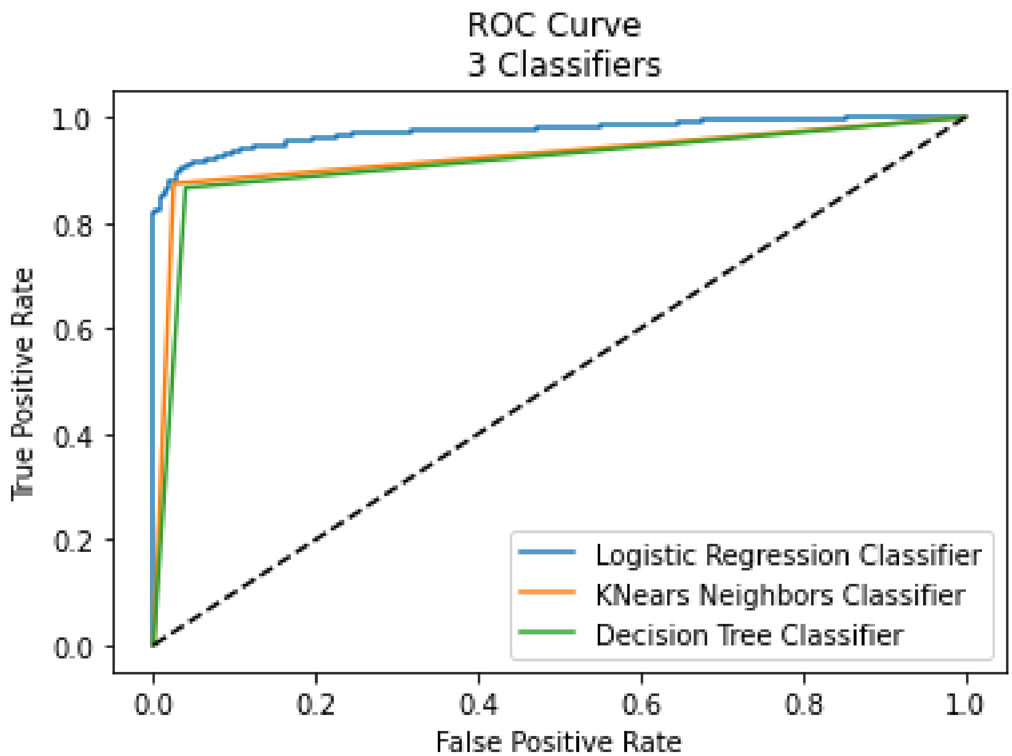
December 3, 2020

[18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import matplotlib.patches as mpatches
import collections
from collections import Counter
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import average_precision_score
import pickle
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings("ignore")
```

[22]:
```
data = pd.read_csv('creditcard.csv')
rob = RobustScaler()
data['rs_amount'] = rob.fit_transform(data['Amount'].values.reshape(-1,1))
data['rs_time'] = rob.fit_transform(data['Time'].values.reshape(-1,1))
data.drop(['Time','Amount'],axis=1, inplace=True)

rs_amount = data['rs_amount']
rs_time = data['rs_time']

data.drop(['rs_amount', 'rs_time'],axis=1, inplace=True)
data.insert(0, 'rs_amount', rs_amount)
data.insert(1, 'rs_time', rs_time)
x = data.drop('Class', axis=1)
y = data['Class']
skf = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for tr_index, te_index in skf.split(x, y):
    orig_x_train, orig_x_test = x.iloc[tr_index], x.iloc[te_index]
    orig_y_train, orig_y_test = y.iloc[tr_index], y.iloc[te_index]
orig_x_train = orig_x_train.values
orig_y_train = orig_y_train.values
orig_x_test = orig_x_test.values
orig_y_test = orig_y_test.values

data= data.sample(frac=1)
F_data = data.loc[data['Class'] == 1]
NF_data = data.loc[data['Class']==0][:492]
sub_data = pd.concat([F_data,NF_data]).sample(frac = 1, random_state = 42)
sub_y = sub_data['Class']
sub_x = sub_data.drop('Class', axis = 1)
train_x, test_x, train_y, test_y = train_test_split(sub_x, sub_y, test_size=0.
 ↪2, random_state=42)
train_x = train_x.values
train_y = train_y.values
test_x = test_x.values
test_y = test_y.values
us_x = data.drop('Class', axis=1)
us_y = data['Class']
# the following make pipelined codes was modified from www.kaggle.com/
 ↪janiobachmann
for tr_index, te_index in skf.split(us_x, us_y):
    us_xtrain, us_xtest = us_x.iloc[tr_index], us_x.iloc[te_index]
    us_ytrain, us_ytest = us_y.iloc[tr_index], us_y.iloc[te_index]

us_xtrain = us_xtrain.values
us_xtest = us_xtest.values
us_ytrain = us_ytrain.values
```

```python
us_ytest = us_ytest.values

Pkl_Filename_LR_U = "Pickle_LR_undersample.pkl"
Pkl_Filename_LR_O = "Pickle_LR_oversample.pkl"
Pkl_Filename_KNN = "Pickle_KNN_undersample.pkl"
Pkl_Filename_DTC = "Pickle_DTC_undersample.pkl"
with open(Pkl_Filename_LR_U, 'rb') as file:
    LOR_clf = pickle.load(file)
with open(Pkl_Filename_LR_O, 'rb') as file:
    best_OS_LOR = pickle.load(file)
with open(Pkl_Filename_KNN, 'rb') as file:
    KNN_clf = pickle.load(file)
with open(Pkl_Filename_DTC, 'rb') as file:
    decision_tree_clf= pickle.load(file)
# the following make pipelined codes was modified from www.kaggle.com/
 ↪janiobachmann
for train, test in skf.split(us_xtrain, us_ytrain):
    us_pipeline =␣
 ↪imbalanced_make_pipeline(NearMiss(sampling_strategy='majority'), LOR_clf)
    us_model = us_pipeline.fit(us_xtrain[train], us_ytrain[train])
    us_predict= us_model.predict(us_xtrain[test])
for train, test in skf.split(us_xtrain, us_ytrain):
    us_pipeline =␣
 ↪imbalanced_make_pipeline(NearMiss(sampling_strategy='majority'),␣
 ↪decision_tree_clf)
    us_model = us_pipeline.fit(us_xtrain[train], us_ytrain[train])
    us_predict= us_model.predict(us_xtrain[test])
for train, test in skf.split(us_xtrain, us_ytrain):
    us_pipeline =␣
 ↪imbalanced_make_pipeline(NearMiss(sampling_strategy='majority'), KNN_clf)
    us_model = us_pipeline.fit(us_xtrain[train], us_ytrain[train])
    us_predict= us_model.predict(us_xtrain[test])
LR_predict = cross_val_predict(LOR_clf, train_x, train_y,␣
 ↪cv=5,method="decision_function")
KNC_predict = cross_val_predict(KNN_clf, train_x, train_y, cv=5)
DTC_predict = cross_val_predict(decision_tree_clf, train_x, train_y, cv=5)

LR_fpr, LR_tpr, LR_thresold = roc_curve(train_y, LR_predict)
KNN_fpr, KNN_tpr, KNN_threshold = roc_curve(train_y, KNC_predict)
DTC_fpr, DTC_tpr, DTC_threshold = roc_curve(train_y, DTC_predict)
plt.title('ROC Curve \n 3 Classifiers')
plt.plot(LR_fpr, LR_tpr, label='Logistic Regression Classifier' )
plt.plot(KNN_fpr, KNN_tpr, label='KNears Neighbors Classifier' )
plt.plot(DTC_fpr, DTC_tpr, label='Decision Tree Classifier' )
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.legend()
plt.show()
```



[24]:
```
os_y_res = best_OS_LOR.decision_function(orig_x_test)

os_avg_precision = average_precision_score(orig_y_test, os_y_res)
#OS during Cross Validation
print('Average precision-recall of oversample by logestic regression(imprtant␣
 ↪metric): {0:0.2f}'.format(
      os_avg_precision))

us_y_res = LOR_clf.decision_function(orig_x_test)

us_avg_precision = average_precision_score(orig_y_test, us_y_res)

US_predict= LOR_clf.predict(orig_x_test)
y_predict_LOR = LOR_clf.predict(test_x)
y_predict_KNN = KNN_clf.predict(test_x)
y_predict_DTC = decision_tree_clf.predict(test_x)
OS_predict = best_OS_LOR.predict(orig_x_test)
```

```
print('Average precision-recall of undersample by logestic regression(imprtant␣
↪metric): {0:0.2f}'.format(
      us_avg_precision))
print("misleading:")
print("MSE of unersample LOR: \n{}\n".format(mean_squared_error(test_y,␣
↪y_predict_LOR)))
print("MSE of unersample KNN: \n{}\n".format(mean_squared_error(test_y,␣
↪y_predict_KNN)))
print("MSE of unersample DTC: \n{}\n".format(mean_squared_error(test_y,␣
↪y_predict_DTC)))
print("MSE of oversample LOR of original data: \n{}\n".
↪format(mean_squared_error(orig_y_test, OS_predict)))
print("MSE of undersample LOR of original data: \n{}\n".
↪format(mean_squared_error(orig_y_test, US_predict)))
```

```
Average precision-recall of oversample by logestic regression(imprtant metric):
0.77
Average precision-recall of undersample by logestic regression(imprtant metric):
0.02
misleading:
MSE of unersample LOR:
0.17258883248730963

MSE of unersample KNN:
0.1065989847715736

MSE of unersample DTC:
0.4263959390862944

MSE of oversample LOR of original data:
0.020417478625726373

MSE of undersample LOR of original data:
0.25031161672021207
```

```
[ ]:
```

EE660 Project

December 3, 2020

[169]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import matplotlib.patches as mpatches
import collections
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
```

[170]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold, StratifiedKFold
```

[171]:
```python
data = pd.read_csv('creditcard.csv')
data
```

[171]:
```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
…              …          …         …         …         …         …
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  …       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274
```

```
4       0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278
…            …         …         …         …    …        …         …
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…            …         …         …         …         …         …       …
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
…         …
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

[173]:

```
Total_value = len(data)
NF_value = data['Class'].value_counts()[0]
F_value = data['Class'].value_counts()[1]
print("No fraud number persentage", round(NF_value/Total_value*100,2))
print("fraud number persentage", round(F_value/Total_value*100,2))
```

```
No fraud number persentage 99.83
fraud number persentage 0.17
```

[174]:

```
sb.countplot('Class', data=data)
plt.title('Class Distributions non fraud and fraud', fontsize=10)
```

[174]: Text(0.5, 1.0, 'Class Distributions non fraud and fraud')

[175]:
```
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
rob = RobustScaler()
data['rs_amount'] = rob.fit_transform(data['Amount'].values.reshape(-1,1))
data['rs_time'] = rob.fit_transform(data['Time'].values.reshape(-1,1))
data.drop(['Time','Amount'],axis=1, inplace=True)

rs_amount = data['rs_amount']
rs_time = data['rs_time']

data.drop(['rs_amount', 'rs_time'],axis=1, inplace=True)
data.insert(0, 'rs_amount', rs_amount)
data.insert(1, 'rs_time', rs_time)

data
```

[175]:
```
   rs_amount   rs_time        V1        V2        V3        V4  \
0   1.783274 -0.994983 -1.359807 -0.072781  2.536347  1.378155
1  -0.269825 -0.994983  1.191857  0.266151  0.166480  0.448154
2   4.983721 -0.994972 -1.358354 -1.340163  1.773209  0.379780
3   1.418291 -0.994972 -0.966272 -0.185226  1.792993 -0.863291
4   0.670579 -0.994960 -1.158233  0.877737  1.548718  0.403034
```

```
…          …        …         …          …        …        …
284802  -0.296653  1.034951 -11.881118  10.071785 -9.834783 -2.066656
284803   0.038986  1.034963  -0.732789  -0.055080  2.035030 -0.738589
284804   0.641096  1.034975   1.919565  -0.301254 -3.249640 -0.557828
284805  -0.167680  1.034975  -0.240440   0.530483  0.702510  0.689799
284806   2.724796  1.035022  -0.533413  -0.189733  0.703337 -0.506271

              V5        V6        V7        V8  …       V20       V21  \
0      -0.338321  0.462388  0.239599  0.098698  …  0.251412 -0.018307
1       0.060018 -0.082361 -0.078803  0.085102  … -0.069083 -0.225775
2      -0.503198  1.800499  0.791461  0.247676  …  0.524980  0.247998
3      -0.010309  1.247203  0.237609  0.377436  … -0.208038 -0.108300
4      -0.407193  0.095921  0.592941 -0.270533  …  0.408542 -0.009431
…            …        …        …        …  …        …        …
284802 -5.364473 -2.606837 -4.918215  7.305334  …  1.475829  0.213454
284803  0.868229  1.058415  0.024330  0.294869  …  0.059616  0.214205
284804  2.630515  3.031260 -0.296827  0.708417  …  0.001396  0.232045
284805 -0.377961  0.623708 -0.686180  0.679145  …  0.127434  0.265245
284806 -0.012546 -0.649617  1.577006 -0.414650  …  0.382948  0.261057

             V22       V23       V24       V25       V26       V27       V28  \
0       0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1      -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2       0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3       0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4       0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
…            …        …        …        …        …        …        …
284802  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
284803  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
284804  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
284805  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
284806  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649

        Class
0           0
1           0
2           0
3           0
4           0
…       …
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

[236]:
```python
#splitting data before undersample and oversample
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import train_test_split
x = data.drop('Class', axis=1)
y = data['Class']
skf = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for tr_index, te_index in skf.split(x, y):
    orig_x_train, orig_x_test = x.iloc[tr_index], x.iloc[te_index]
    orig_y_train, orig_y_test = y.iloc[tr_index], y.iloc[te_index]
orig_x_train = orig_x_train.values
orig_y_train = orig_y_train.values
orig_x_test = orig_x_test.values
orig_y_test = orig_y_test.values
```

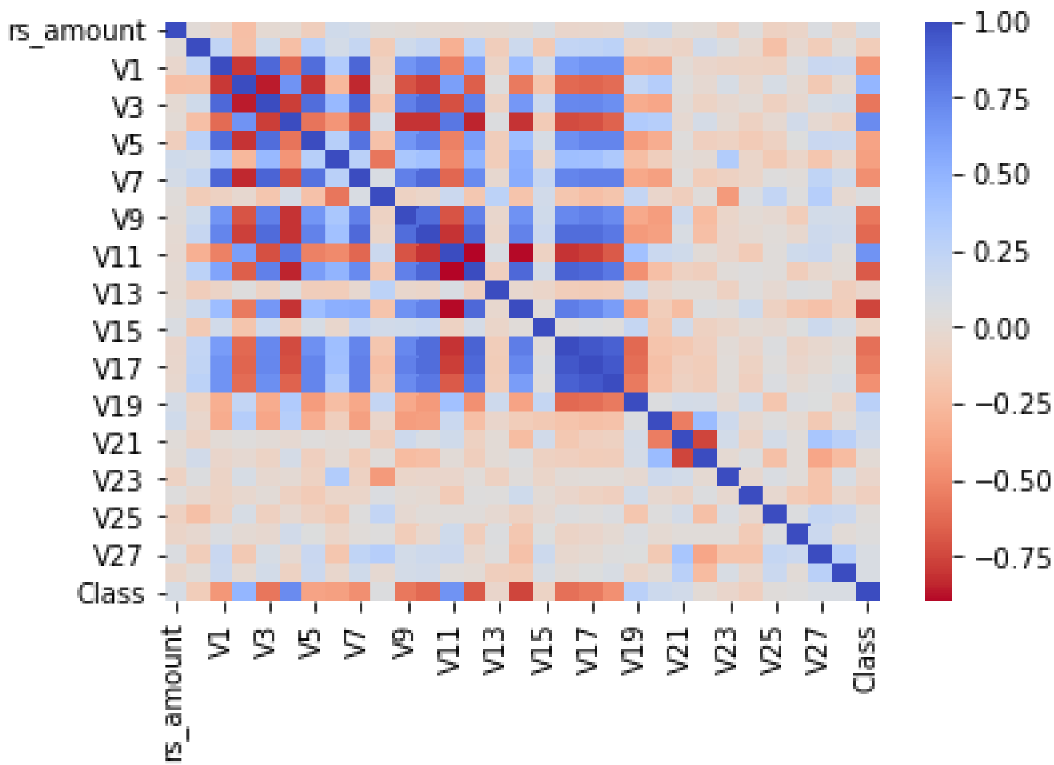
[253]:
```python
data= data.sample(frac=1)
F_data = data.loc[data['Class'] == 1]
NF_data = data.loc[data['Class']==0][:492]
sub_data = pd.concat([F_data,NF_data]).sample(frac = 1, random_state = 42)

print('subsample distribution')
print(sub_data['Class'].value_counts()/len(sub_data))
sub_data
```

```
subsample distribution
1    0.5
0    0.5
Name: Class, dtype: float64
```

[253]:
```
        rs_amount   rs_time        V1        V2        V3        V4        V5  \
9262    -0.098232 -0.838297  1.226878  0.063952  0.795418  0.109193 -0.491986
6427    -0.293440 -0.905579  0.725646  2.300894 -5.329976  4.007683 -1.730411
133014   0.139733 -0.052656  1.135322  0.142540  0.122711  0.474670 -0.003014
249963  -0.296653  0.821967 -0.679521  4.672553 -6.814798  7.143500  0.928654
204079   1.208831  0.592230  1.862102 -0.124052 -1.989752  0.382609  0.473032
…             …         …         …         …         …         …         …
144754   4.216726  0.019784 -0.670238  0.945206  0.610051  2.640065 -2.707775
247673   3.156012  0.810172 -5.192496  3.164721 -5.047679  2.246597 -4.011781
61367   -0.279746 -0.409908  1.252850  0.293765 -0.170780  0.243902  0.528910
151730  -0.042479  0.134435 -1.952933  3.541385 -1.310561  5.955664 -1.003993
74794    4.051003 -0.339901 -6.003422 -3.930731 -0.007045  1.714669  3.414667

              V6        V7        V8  …       V20       V21       V22  \
9262   -0.345963 -0.409757 -0.090610  … -0.003457 -0.297851 -0.635229
6427   -1.732193 -3.968593  1.063728  …  0.504646  0.589669  0.109541
133014 -0.078456 -0.068025  0.101293  … -0.035654 -0.213125 -0.688959
249963 -1.873013 -2.306689  0.993702  …  0.872006  0.566849 -0.321691
```

```
204079 -0.674517  0.298621 -0.282416  …  0.150727 -0.204158 -0.511441
…             …         …         …   …  …        …         …
144754  1.952611 -1.624608 -5.229908  …  1.474929 -2.504450  1.436472
247673 -0.638908 -2.873463  1.576318  … -1.850470  1.167244 -1.006617
61367   0.413733 -0.052919  0.101599  … -0.001047 -0.273833 -0.738917
151730  0.983049 -4.587235 -4.892184  …  1.965030 -1.998091  1.133706
74794  -2.329583 -1.901512 -2.746111  … -4.128186  1.101671 -0.992494

             V23       V24       V25       V26       V27       V28  Class
9262    0.094170 -0.000176  0.027330  0.748993 -0.089141 -0.001460      0
6427    0.601045 -0.364700 -1.843078  0.351909  0.594550  0.099372      1
133014  0.079722 -0.355197  0.143186  0.124495 -0.017226  0.018885      0
249963 -0.281325 -1.120256 -0.073394  0.553530  0.760542  0.386742      1
204079  0.077874  0.388335  0.007896 -0.120980 -0.019579  0.006155      1
…            …         …         …         …         …         …    …
144754  0.351542  0.648467  0.579681  0.075738  0.346717  0.282209      1
247673  0.774562  0.063397 -0.390658  1.884741 -1.742558 -0.082216      1
61367  -0.009814 -1.162879  0.301826  0.175835 -0.005637  0.007501      0
151730 -0.041461 -0.215379 -0.865599  0.212545  0.532897  0.357892      1
74794  -0.698259  0.139898 -0.205151 -0.472412  1.775378 -0.104285      1

[984 rows x 31 columns]
```

[254]:
```python
sb.heatmap(sub_data.corr(), cmap='coolwarm_r', annot_kws={'size':20})
plt.show()
```



[255]:
```
from sklearn.model_selection import train_test_split
sub_y = sub_data['Class']
sub_x = sub_data.drop('Class', axis = 1)
train_x, test_x, train_y, test_y = train_test_split(sub_x, sub_y, test_size=0.
 ↪2, random_state=42)
train_x = train_x.values
train_y = train_y.values
test_x = test_x.values
test_y = test_y.values
```

[274]:
```
#baseline 1
from sklearn.model_selection import cross_val_score
DTC = DecisionTreeClassifier()
DTC.fit(train_x, train_y)
train_score = cross_val_score(DTC, train_x, train_y, cv=5)
avg_score = train_score.mean()
print("decision tree Classifier training score ", round(avg_score, 2)*100)
```

```
decision tree Classifier training score  89.0
```

[275]:
```
from sklearn.model_selection import GridSearchCV
#thanks to internet source, i got valid parameter choices
para_tree = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
              "min_samples_leaf": list(range(5,7,1))}
decision_tree = GridSearchCV(DecisionTreeClassifier(), para_tree)
decision_tree.fit(train_x, train_y)
decision_tree_clf = decision_tree.best_estimator_
decision_tree_score = cross_val_score(decision_tree_clf, train_x, train_y, cv=5)
avg_tree_score = decision_tree_score.mean()
print("decision tree Classifier cross validation score",␣
 ↪round(avg_tree_score*100, 2).astype(str))
```

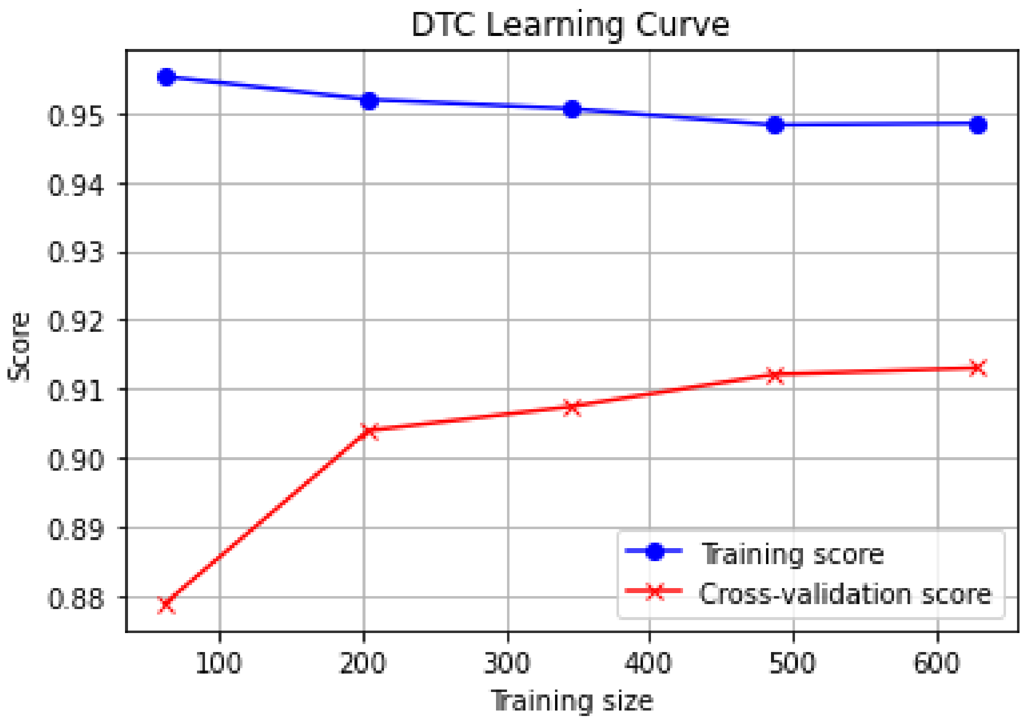
```
decision tree Classifier cross validation score 91.87
```

[276]:
```
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
us_x = data.drop('Class', axis=1)
us_y = data['Class']
# the following make pipelined codes was modified from www.kaggle.com/
 ↪janiobachmann
for tr_index, te_index in skf.split(us_x, us_y):
    us_xtrain, us_xtest = us_x.iloc[tr_index], us_x.iloc[te_index]
```

```
    us_ytrain, us_ytest = us_y.iloc[tr_index], us_y.iloc[te_index]

us_xtrain = us_xtrain.values
us_xtest = us_xtest.values
us_ytrain = us_ytrain.values
us_ytest = us_ytest.values
# the following make pipelined codes was modified from www.kaggle.com/
 ↪janiobachmann
for train, test in skf.split(us_xtrain, us_ytrain):
    us_pipeline =␣
 ↪imbalanced_make_pipeline(NearMiss(sampling_strategy='majority'),␣
 ↪decision_tree_clf)
    us_model = us_pipeline.fit(us_xtrain[train], us_ytrain[train])
    us_predict= us_model.predict(us_xtrain[test])
```

[277]:

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import learning_curve
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=233)
tr_sizes=np.linspace(.1, 1.0, 5)
tr_sizes, tr_scores, te_scores = learning_curve(
       decision_tree_clf , train_x, train_y, cv=cv, n_jobs=1,␣
 ↪train_sizes=tr_sizes)
tr_scores_avg = np.mean(tr_scores,axis = 1)
te_scores_avg = np.mean(te_scores,axis = 1)
plt.plot(tr_sizes, tr_scores_avg, 'o-', color="blue",
             label="Training score")
plt.plot(tr_sizes, te_scores_avg, 'x-', color="red",
             label="Cross-validation score")
plt.title("DTC Learning Curve")
plt.xlabel('Training size')
plt.ylabel('Score')
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



[278]:
```
from sklearn.metrics import roc_curve
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import roc_auc_score
DTC_predict = cross_val_predict(decision_tree_clf, train_x, train_y, cv=5)
print('DTC: ', roc_auc_score(train_y, DTC_predict))
```

```
DTC:  0.916631116282076
```

[318]:
```
print("Best Parameters of DTC: \n{}\n".format(decision_tree.best_params_))
```

```
Best Parameters of DTC:
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 5}
```

[283]:
```
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

y_predict_DTC = decision_tree_clf.predict(test_x)

print('Recall of DTC : {:.2f}'.format(recall_score(test_y, y_predict_DTC)))
print('Precision of DTC : {:.2f}'.format(precision_score(test_y,␣
 ↪y_predict_DTC)))
```

```
print('F1 of DTC: {:.2f}'.format(f1_score(test_y,y_predict_DTC)))
print('Accuracy of DTC : {:.2f}'.format(accuracy_score(test_y, y_predict_DTC)))
```

```
Recall of DTC : 0.97
Precision of DTC : 0.66
F1 of DTC: 0.79
Accuracy of DTC : 0.71
```

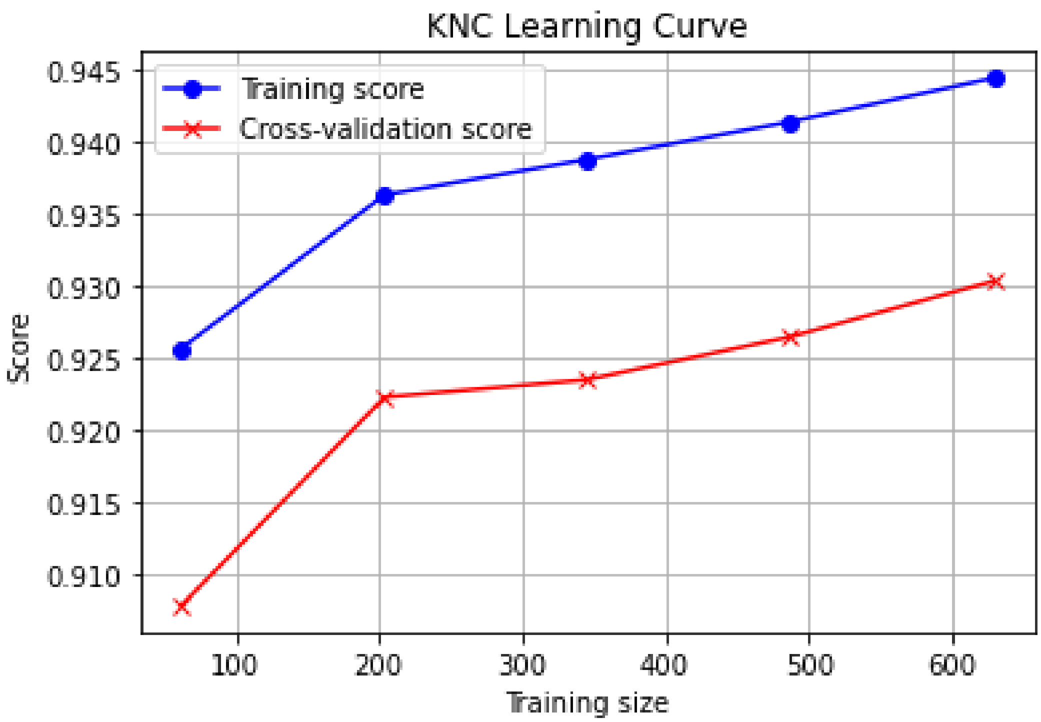
[284]:
```
#baseline2
KNC = KNeighborsClassifier()
KNC.fit(train_x, train_y)
train_score_KNC = cross_val_score(KNC, train_x, train_y, cv=5)
avg_score_KNC = train_score.mean()
print("KNN Classifier training score ", round(avg_score_KNC*100, 2))
#thanks to internet source, i got valid parameter choices
para_KNC = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['auto',␣
↪'ball_tree', 'kd_tree', 'brute']}
KNN = GridSearchCV(KNeighborsClassifier(), para_KNC)
KNN.fit(train_x, train_y)
KNN_clf = KNN.best_estimator_
KNN_score = cross_val_score(KNN_clf, train_x, train_y, cv=5)
avg_KNN_score = KNN_score.mean()
print("KNN Classifier cross validation score", round(avg_KNN_score*100, 2).
↪astype(str))
```

```
KNN Classifier training score  88.69
KNN Classifier cross validation score 92.88
```

[285]:
```
# the following make pipelined codes was modified from www.kaggle.com/
↪janiobachmann
for train, test in skf.split(us_xtrain, us_ytrain):
    us_pipeline =␣
↪imbalanced_make_pipeline(NearMiss(sampling_strategy='majority'), KNN_clf)
    us_model = us_pipeline.fit(us_xtrain[train], us_ytrain[train])
    us_predict= us_model.predict(us_xtrain[test])
```

[286]:
```
tr_sizes_KNN=np.linspace(.1, 1.0, 5)
tr_sizes_KNN, tr_scores_KNN, te_scores_KNN = learning_curve(
       KNN_clf , train_x, train_y, cv=cv, n_jobs=1, train_sizes=tr_sizes_KNN)
tr_scores_KNN_avg = np.mean(tr_scores_KNN,axis = 1)
te_scores_KNN_avg = np.mean(te_scores_KNN,axis = 1)
plt.plot(tr_sizes, tr_scores_KNN_avg, 'o-', color="blue",
             label="Training score")
plt.plot(tr_sizes, te_scores_KNN_avg, 'x-', color="red",
             label="Cross-validation score")
plt.title("KNC Learning Curve")
plt.xlabel('Training size')
```

```
plt.ylabel('Score')
plt.grid(True)
plt.legend(loc="best")
plt.show()
KNC_predict = cross_val_predict(KNN_clf, train_x, train_y, cv=5)
print('KNNC: ', roc_auc_score(train_y, KNC_predict))
```




```
KNNC:  0.927370564281559
```

[290]:

```
y_predict_KNN = KNN_clf.predict(test_x)

print('Recall of KNN : {:.2f}'.format(recall_score(test_y, y_predict_KNN)))
print('Precision of KNN : {:.2f}'.format(precision_score(test_y,␣
↪y_predict_KNN)))
print('F1 of KNN: {:.2f}'.format(f1_score(test_y,y_predict_KNN)))
print('Accuracy of KNN : {:.2f}'.format(accuracy_score(test_y, y_predict_KNN)))
```

```
Recall of KNN : 0.95
Precision of KNN : 0.91
F1 of KNN: 0.93
Accuracy of KNN : 0.92
```

[325]:

```
print("Best Parameters of KNN: \n{}\n".format(KNN.best_params_))
```

```
Best Parameters of KNN:
{'algorithm': 'auto', 'n_neighbors': 4}
```

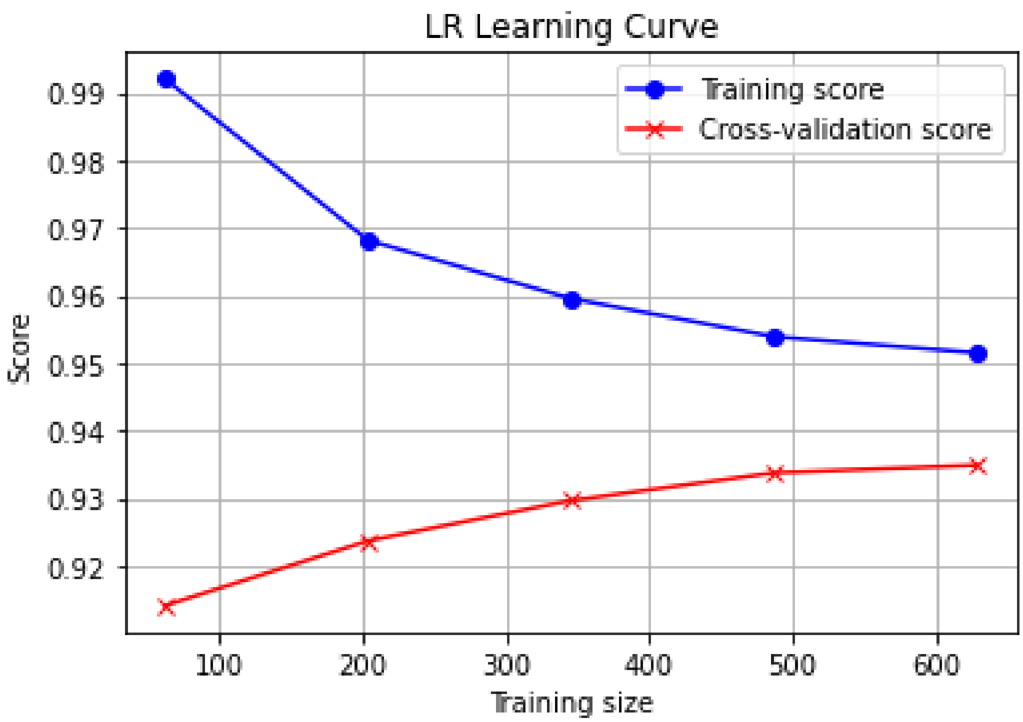
[292]:
```
# good one
LR = LogisticRegression()
LR.fit(train_x, train_y)
train_score_LR = cross_val_score(LR, train_x, train_y, cv=5)
avg_score_LR = train_score.mean()
print("Logistic regression training score ", round(avg_score_LR*100, 2))
#thanks to internet source, i got valid parameter choices
para_LR = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
LOR = GridSearchCV(LogisticRegression(), para_LR)
LOR.fit(train_x, train_y)
LOR_clf = LOR.best_estimator_
LOR_score = cross_val_score(LOR_clf, train_x, train_y, cv=5)
avg_LOR_score = LOR_score.mean()
print("Logistic regression cross validation score", round(avg_LOR_score*100, 2).
 ↪astype(str))
```

```
Logistic regression training score  88.69
Logistic regression cross validation score 93.77
```

[293]:
```
from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.metrics import classification_report_imbalanced
# the following make pipelined codes was modified from www.kaggle.com/
 ↪janiobachmann
for train, test in skf.split(us_xtrain, us_ytrain):
    us_pipeline =␣
 ↪imbalanced_make_pipeline(NearMiss(sampling_strategy='majority'), LOR_clf)
    us_model = us_pipeline.fit(us_xtrain[train], us_ytrain[train])
    us_predict= us_model.predict(us_xtrain[test])
```

[294]:
```
tr_sizes_LR=np.linspace(.1, 1.0, 5)
tr_sizes_LR, tr_scores_LR, te_scores_LR = learning_curve(
       LOR_clf , train_x, train_y, cv=cv, n_jobs=1, train_sizes=tr_sizes_LR)
tr_scores_LR_avg = np.mean(tr_scores_LR,axis = 1)
te_scores_LR_avg = np.mean(te_scores_LR,axis = 1)
plt.plot(tr_sizes_LR, tr_scores_LR_avg, 'o-', color="blue",
             label="Training score")
plt.plot(tr_sizes_LR, te_scores_LR_avg, 'x-', color="red",
             label="Cross-validation score")
plt.title("LR Learning Curve")
plt.xlabel('Training size')
```

```
plt.ylabel('Score')
plt.grid(True)
plt.legend(loc="best")
plt.show()
LR_predict = cross_val_predict(LOR_clf, train_x, train_y,␣
 ↪cv=5,method="decision_function")
print('LORC: ', roc_auc_score(train_y, LR_predict))
```




```
LORC:  0.9726585223967423
```

[295]:
```
y_predict_LOR = LOR_clf.predict(test_x)

print('Recall of LR : {:.2f}'.format(recall_score(test_y, y_predict_LOR)))
print('Precision of LR : {:.2f}'.format(precision_score(test_y, y_predict_LOR)))
print('F1 of LR: {:.2f}'.format(f1_score(test_y,y_predict_LOR)))
print('Accuracy of LR : {:.2f}'.format(accuracy_score(test_y, y_predict_LOR)))
```

```
Recall of LR : 0.98
Precision of LR : 0.69
F1 of LR: 0.81
Accuracy of LR : 0.74
```

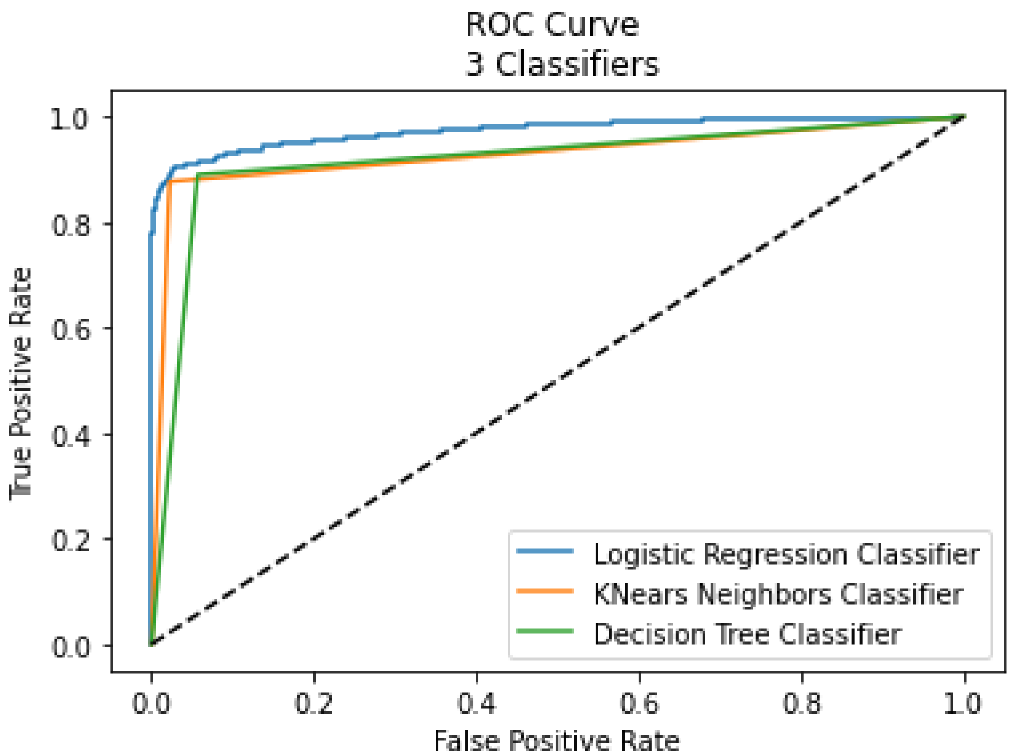
[296]:
```
from sklearn.metrics import average_precision_score
us_y_res = LOR_clf.decision_function(orig_x_test)

us_avg_precision = average_precision_score(orig_y_test, us_y_res)

print('Average precision-recall: {0:0.2f}'.format(
      us_avg_precision))
```

```
Average precision-recall: 0.08
```

[323]:
```
print("Best Parameters of under sample LR: \n{}\n".format(LOR.best_params_))
```

```
Best Parameters of under sample LR:
{'C': 1, 'penalty': 'l2'}
```

[333]:
```
from sklearn.metrics import roc_curve
LR_fpr, LR_tpr, LR_thresold = roc_curve(train_y, LR_predict)
KNN_fpr, KNN_tpr, KNN_threshold = roc_curve(train_y, KNC_predict)
DTC_fpr, DTC_tpr, DTC_threshold = roc_curve(train_y, DTC_predict)
plt.title('ROC Curve \n 3 Classifiers')
plt.plot(LR_fpr, LR_tpr, label='Logistic Regression Classifier' )
plt.plot(KNN_fpr, KNN_tpr, label='KNears Neighbors Classifier' )
plt.plot(DTC_fpr, DTC_tpr, label='Decision Tree Classifier' )
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



[335]:

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report
LOR_sm = LogisticRegression()
rand_LOR = RandomizedSearchCV(LogisticRegression(), para_LR, n_iter=4)
# the following make pipelined codes was modified from www.kaggle.com/
 ↪janiobachmann
for train, test in skf.split(orig_x_train, orig_y_train):
    os_pipeline = imbalanced_make_pipeline(SMOTE(sampling_strategy='minority'),␣
 ↪rand_LOR)
    os_model = os_pipeline.fit(orig_x_train[train], orig_y_train[train])
    best_OS_LOR = rand_LOR.best_estimator_

OS_predict = best_OS_LOR.predict(orig_x_test)
```

[304]:

```
os_y_res = best_OS_LOR.decision_function(orig_x_test)

os_avg_precision = average_precision_score(orig_y_test, os_y_res)
#OS during Cross Validation
print('Average precision-recall of oversample by logestic regression: {0:0.2f}'.
 ↪format(
      os_avg_precision))
```

```
Average precision-recall of oversample by logestic regression: 0.73
```

[324]:
```python
print("Best Parameters of over sample: \n{}\n".format(rand_LOR.best_params_))
```

```
Best Parameters of over sample:
{'penalty': 'l2', 'C': 0.001}
```

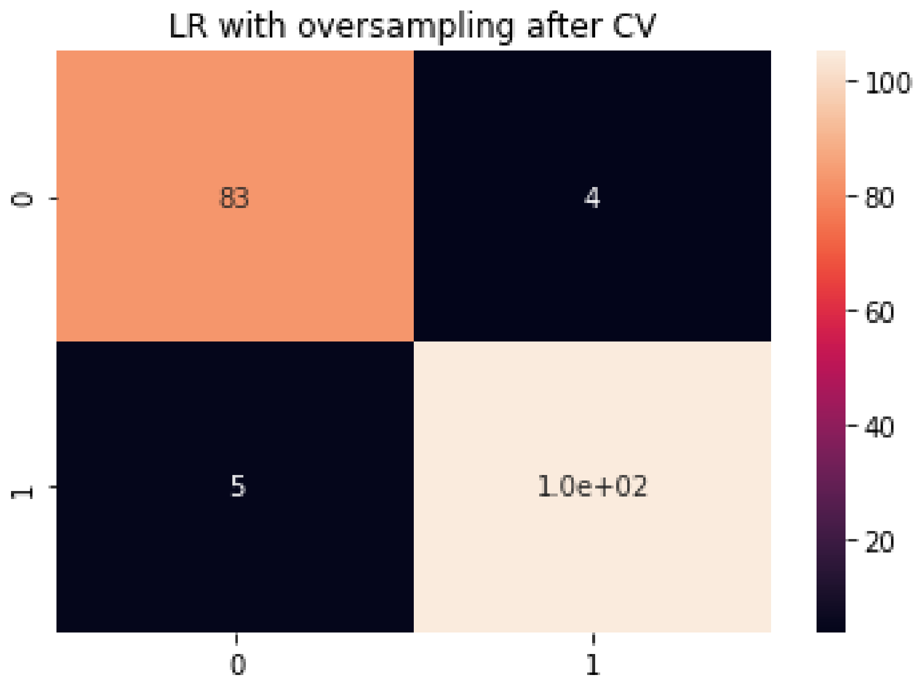
[308]:
```python
#OS after Cross Validation
from imblearn.over_sampling import SMOTE
from sklearn.metrics import confusion_matrix
sm_os = SMOTE(sampling_strategy='minority', random_state=42)
sm_train_x, sm_train_y = sm_os.fit_sample(orig_x_train, orig_y_train)
sm_os_LOR = LOR.best_estimator_
sm_os_LOR.fit(sm_train_x,sm_train_y)
LOR_y_predict = sm_os_LOR.predict(test_x)
confusionM_of_LOR_sm_after_CV = confusion_matrix(test_y, LOR_y_predict)
confusionM_of_KNN = confusion_matrix(test_y, y_predict_KNN)
confusionM_of_DTC = confusion_matrix(test_y, y_predict_DTC)
```

[334]:
```python
from sklearn.metrics import mean_squared_error
US_predict= LOR_clf.predict(orig_x_test)
print("MSE of unersample LOR: \n{}\n".format(mean_squared_error(test_y,␣
 ↪y_predict_LOR)))
print("MSE of unersample KNN: \n{}\n".format(mean_squared_error(test_y,␣
 ↪y_predict_KNN)))
print("MSE of unersample DTC: \n{}\n".format(mean_squared_error(test_y,␣
 ↪y_predict_DTC)))
print("MSE of oversample LOR of original data: \n{}\n".
 ↪format(mean_squared_error(orig_y_test, OS_predict)))
print("MSE of undersample LOR of original data: \n{}\n".
 ↪format(mean_squared_error(orig_y_test, US_predict)))
```

```
MSE of unersample LOR:
0.25888324873096447

MSE of unersample KNN:
0.07614213197969544

MSE of unersample DTC:
0.29441624365482233

MSE of oversample LOR of original data:
0.024630887800424852

MSE of undersample LOR of original data:
0.024683555415108582
```

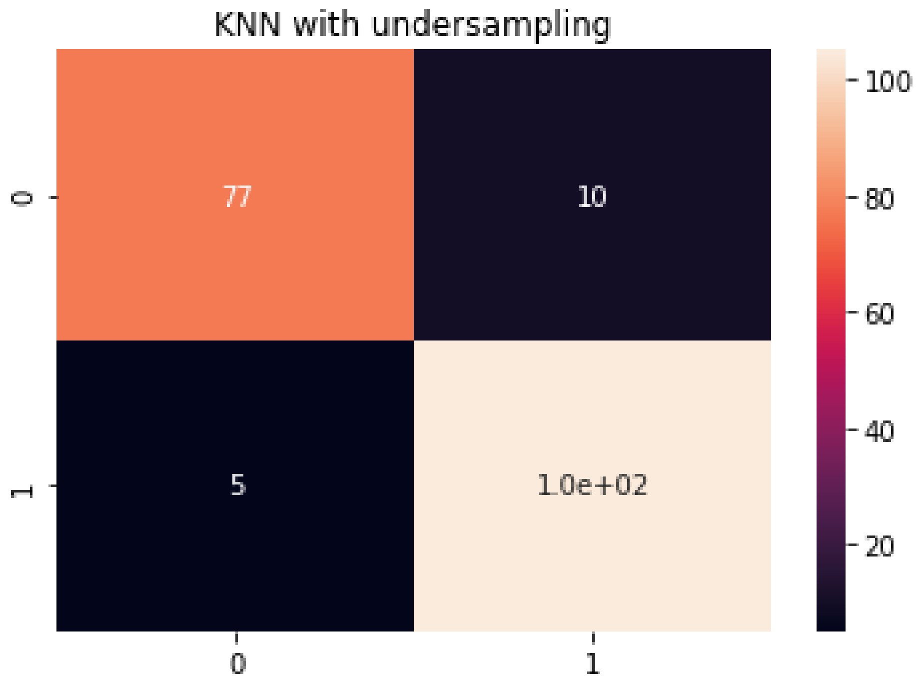
[312]:

```
sb.heatmap(confusionM_of_LOR_sm_after_CV,annot=True)
plt.title("LR with oversampling after CV in subsample result")
plt.show()
```

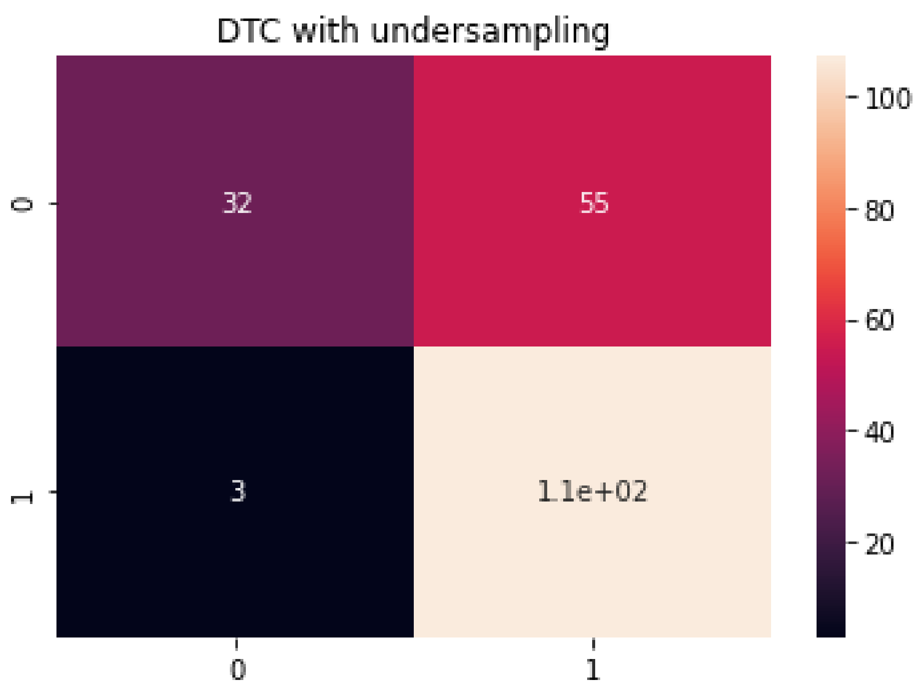


[313]:

```
sb.heatmap(confusionM_of_KNN,annot=True)
plt.title("KNN with undersampling in subsample result")
plt.show()
```

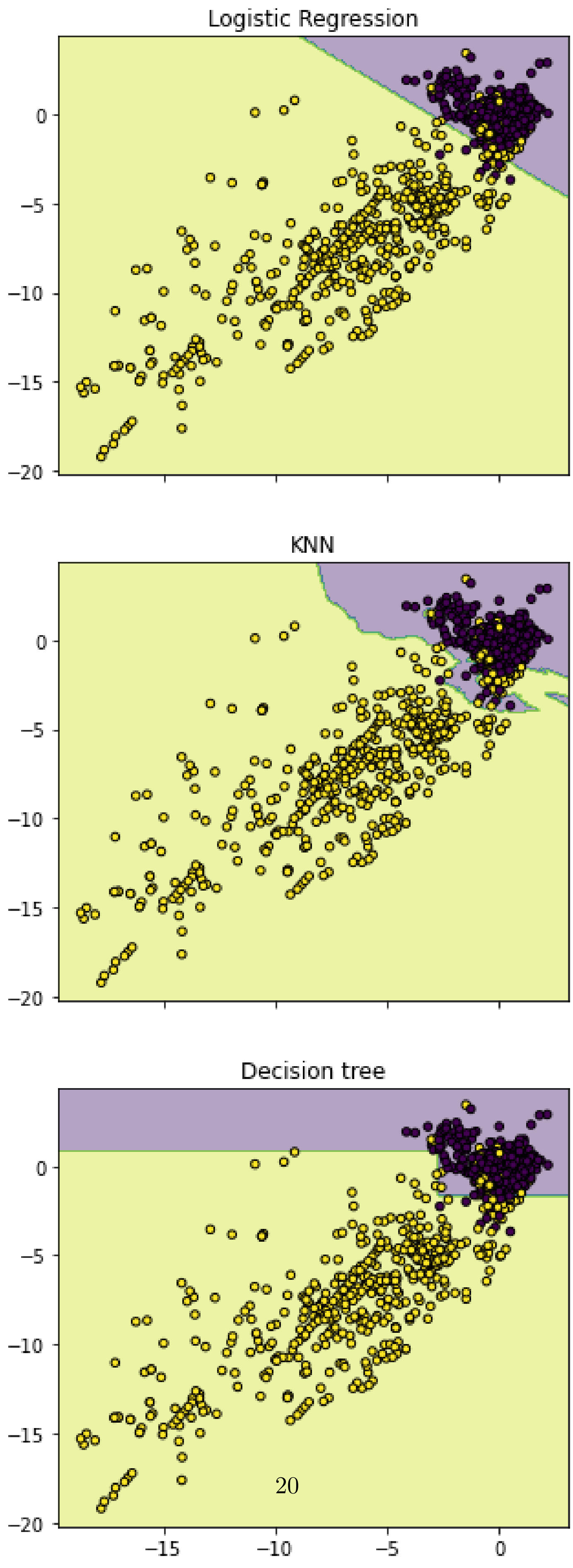


[316]:
```
sb.heatmap(confusionM_of_DTC,annot=True)
plt.title("DTC with undersampling in subsample result")
plt.show()
```



[337]:
```
import pickle
Pkl_Filename_LR_U = "Pickle_LR_undersample.pkl"
with open(Pkl_Filename_LR_U, 'wb') as file:
    pickle.dump(LOR_clf, file)
Pkl_Filename_LR_O = "Pickle_LR_oversample.pkl"
with open(Pkl_Filename_LR_O, 'wb') as file:
    pickle.dump(best_OS_LOR, file)
Pkl_Filename_KNN = "Pickle_KNN_undersample.pkl"
with open(Pkl_Filename_KNN, 'wb') as file:
    pickle.dump(KNN_clf, file)
Pkl_Filename_DTC = "Pickle_DTC_undersample.pkl"
with open(Pkl_Filename_DTC, 'wb') as file:
    pickle.dump(decision_tree_clf, file)
```

[353]:
```
#The folloing plotting code changed from
#https://scikit-learn.org/stable/auto_examples/ensemble/
 ↪plot_voting_decision_regions.html
dbx=sub_x[["V12","V14"]]
dby=sub_y
LOR_clf.fit(dbx,dby)
KNN_clf.fit(dbx,dby)
decision_tree_clf.fit(dbx,dby)

x_min= dbx.to_numpy()[:,0].min() - 1
x_max= dbx.to_numpy()[:,0].max() + 1
y_min, y_max = dbx.to_numpy()[:, 1].min() - 1, dbx.to_numpy()[:, 1].max() + 1
x_x, y_y = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

f, axarr = plt.subplots(3, 1, sharex='col', sharey='row',figsize=(5, 15))

for index, classifier, title in zip([0,1,2],
                        [LOR_clf, KNN_clf, decision_tree_clf],
                        ['Logistic Regression', 'KNN',
                         'Decision tree']):

    Z = classifier.predict(np.c_[x_x.ravel(), y_y.ravel()])
    Z = Z.reshape(x_x.shape)

    axarr[index].contourf(x_x, y_y, Z, alpha=0.4)
    axarr[index].scatter(dbx.to_numpy()[:, 0], dbx.to_numpy()[:, 1], c=dby.
 ↪to_numpy(),
                                  s=20, edgecolor='k')
    axarr[index].set_title(title)
plt.show()
```


Logistic Regression
KNN
Decision tree
0
−5
−10
−15
−20
−15
−10
−5
0

```
[ ]:
```