Logistic_regression

November 8, 2021

In this notebook: * Hyper-parameter tuning for LR using Grid Search cross validation * Applying elastic net regularization (LASSO+Rigde) with 'saga' solver * Running Logistic Regression with selected features from XGBooster algorithm * It should be considered that linear regression assumes input data has a linear relation with the target, there are no outliers, no collinearity, normal distribution, and since it's a distance-based algorithm, predictors should be scaled.

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, roc_curve, roc_auc_score
from sklearn import preprocessing

from imblearn.combine import SMOTETomek

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import std
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31:
FutureWarning: The module is deprecated in version 0.21 and will be removed in
version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144:
FutureWarning: The sklearn.neighbors.base module is  deprecated in version 0.22
and will be removed in version 0.24. The corresponding classes / functions
should instead be imported from sklearn.neighbors. Anything that cannot be
imported from sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
```

```
[ ]: #Import Drive API and authenticate
     from google.colab import drive
     #Mount Drive to the Colab VM
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ]: #Load the dataset into pandas DataFrame
     df = pd.read_csv("/content/drive/MyDrive/Capstone_project/v2_credit_default.
      ↪csv")
```

```
[ ]: #Seperate the independent and dependent variables.
     df_independent = df.drop(['Default'], axis=1)
     df_default = df['Default']
```

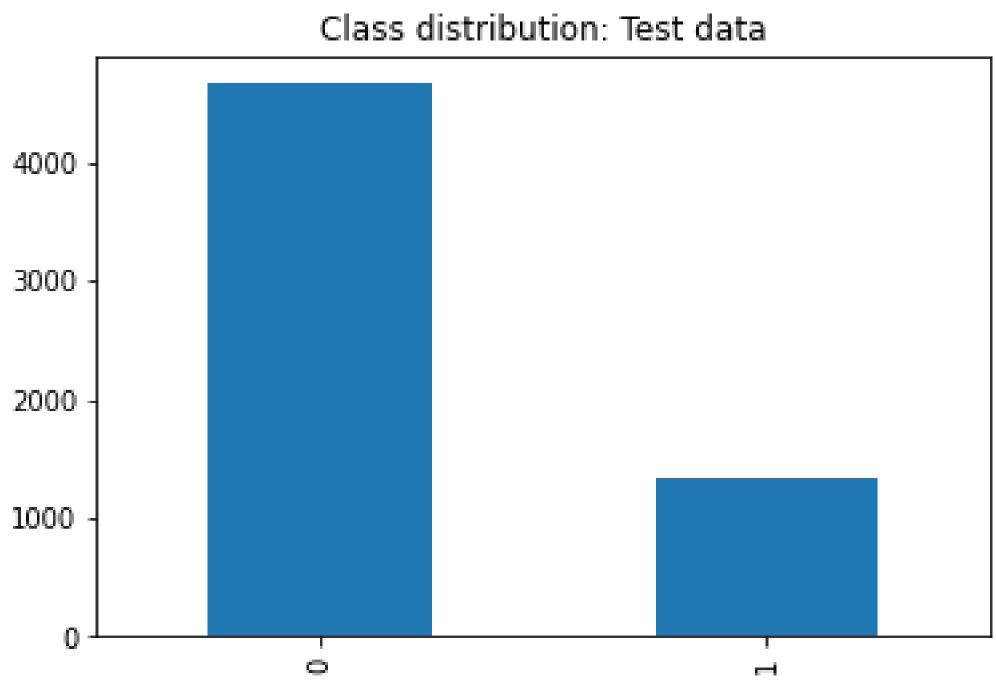
```
[ ]: # split the data into 80% training+validation and 20% test
     X_train, X_test, y_train, y_test = train_test_split(df_independent, df_default,␣
      ↪test_size=0.20, random_state=1)
```

```
[ ]: #Make sure the distribution of the dependent variable is the same in both␣
      ↪training+validation and test sets.
     y_train.value_counts().plot(kind='bar', title='Class distribution: Training +␣
      ↪validation data');
     y_train.value_counts()    #22% defaulters in the training+validation data
```

```
[ ]: 0    18670
     1     5302
     Name: Default, dtype: int64
```


```
[ ]: # Test data
     y_test.value_counts().plot(kind='bar', title='Class distribution: Test data')
     y_test.value_counts() #22% defaulters in the test data
```

```
[ ]: 0    4665
     1    1328
     Name: Default, dtype: int64
```



[ ]:
```
# Scale input variables for training+validation (X_train)
X_train_scaled = preprocessing.MinMaxScaler().fit_transform(X_train)
```

[ ]:
```
# Balancing using SMOTE Tomek
X_smt, y_smt = SMOTETomek(random_state=1).fit_sample(X_train_scaled, y_train.
 ↪squeeze())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

[ ]:
```
# #hyperparameter adjustment with GridSearchCV
# #NOTE 1: The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2␣
 ↪regularization or no regularization. The 'liblinear' solver supports both L1␣
 ↪and L2 regularization.
# #NOTE 2: After running this block of code, the results are recorded (last 2␣
 ↪lines of code), and this section is commented to speed up future runs of the␣
 ↪notebook where other sections are being updated.
# model = LogisticRegression(random_state=1)
```

```
# solver_options = ['newton-cg', 'lbfgs', 'liblinear', 'sag']
# penalty_options = ['l1','l2']
# C_options = [0.1, 0.5, 1, 5, 10, 50, 100]

# param_grid = dict(solver = solver_options, penalty = penalty_options, C =
↪C_options)
# grid = GridSearchCV(model, param_grid, cv=10, scoring = 'f1')
# grid.fit(X_smt,y_smt)
# print (grid.best_params_) # {'C': 50, 'penalty': 'l1', 'solver': 'liblinear'}
# print (grid.best_score_) # output: 0.676
```

[ ]:
```
#Now use the optimized hyperparameters
cv = RepeatedKFold(n_splits=10, n_repeats=5, random_state=1)    #5 repeats of
↪k=10-fold
# create model
model = LogisticRegression(random_state=1, C= 50, penalty= 'l1', solver=
↪'liblinear')
# evaluate model
scores = cross_val_score(model, X_smt, y_smt, scoring='f1', cv=cv, n_jobs=-1)
↪# will have 50 scores (5 iterations x 10-folds)
# report performance
print('f1_score: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
f1_score: 0.676 (0.010)
```

[ ]:
```
# # #hyperparameter optimization with GridSearchCV for 'elasticnet' penalty
↪with 'saga' solver

# model = LogisticRegression(random_state=1)

# solver_options = ['saga']
# penalty_options = ['elasticnet']
# l1_ratio_options = [0.1, .03, 0.5, 0.7, 0.9]
# C_options = [0.1, 0.5, 1, 5, 10, 50, 100]

# param_grid = dict(solver = solver_options, penalty = penalty_options,l1_ratio
↪= l1_ratio_options, C = C_options)
# grid = GridSearchCV(model, param_grid, cv=10, scoring = 'f1')
# grid.fit(X_smt,y_smt)
# print (grid.best_params_) #{'C': 0.1, 'l1_ratio': 0.03, 'penalty':
↪'elasticnet', 'solver': 'saga'}
# print (grid.best_score_)  #0.6914246260933862
```

[ ]:
```
# Now also try elasticnet penalty with saga solver
# Can also use GridSearchCV  to find the optimal l1_ratio and C for elastic net
↪+ saga
```

```
cv = RepeatedKFold(n_splits=10, n_repeats=5, random_state=1)    #5 repeats of␣
 ↪k=10-fold
# create model
model = LogisticRegression(random_state=1, C= 0.1, l1_ratio= 0.03, penalty=␣
 ↪'elasticnet', solver= 'saga')
# evaluate model
scores = cross_val_score(model, X_smt, y_smt, scoring='f1', cv=cv, n_jobs=-1)  ␣
 ↪# will have 50 scores (5 iterations x 10-folds)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

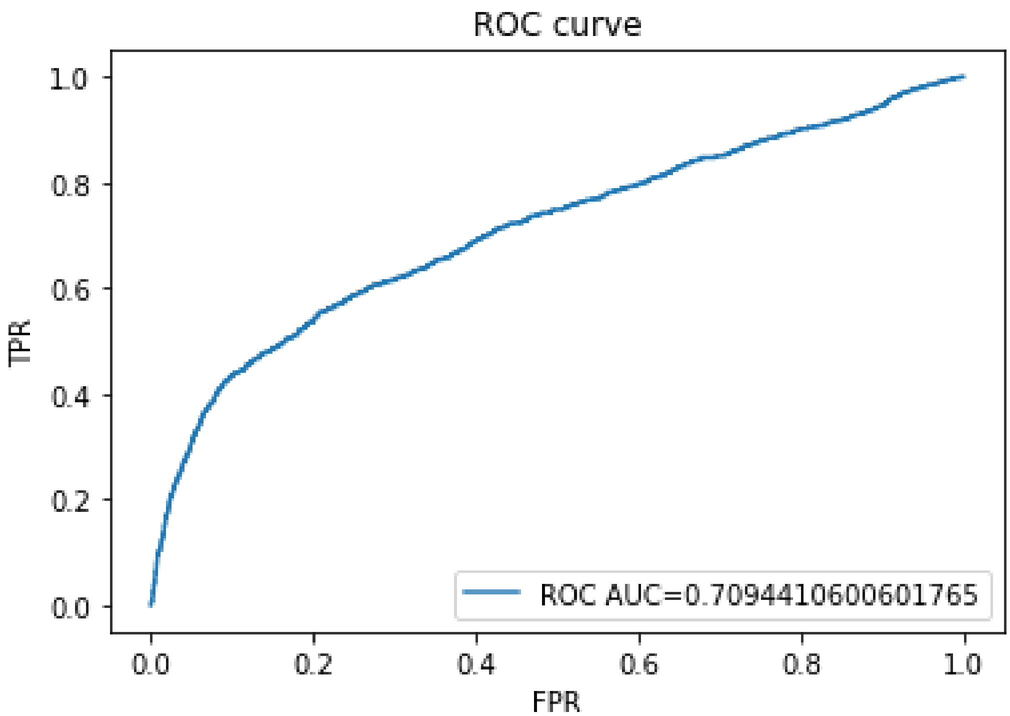
```
Accuracy: 0.670 (0.010)
```

[ ]:
```
#Finally test with the test set (X_test):
# Fit the model
model.fit(X_smt, y_smt)
# Predict using the scaled X_test
X_test_scaled = preprocessing.MinMaxScaler().fit_transform(X_test)
y_pred = model.predict(X_test_scaled)
```

[ ]:
```
# performance metrics
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('accuracy', accuracy_score(y_test, y_pred))
print('precision', precision_score(y_test, y_pred))
print('recall', recall_score(y_test, y_pred))
```

```
[[4065  600]
 [ 709  619]]
accuracy 0.7815785082596363
precision 0.5077932731747334
recall 0.4661144578313253
```

[ ]:
```
# ROC curve, and ROC AUC
y_pred_proba = model.predict_proba(X_test_scaled)[::,1]
FPR, TPR, _ = roc_curve(y_test, y_pred_proba)            #roc_curve(y_true,␣
 ↪y_score)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(FPR,TPR,label="ROC AUC="+str(auc))
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend(loc=4)
plt.show()
```

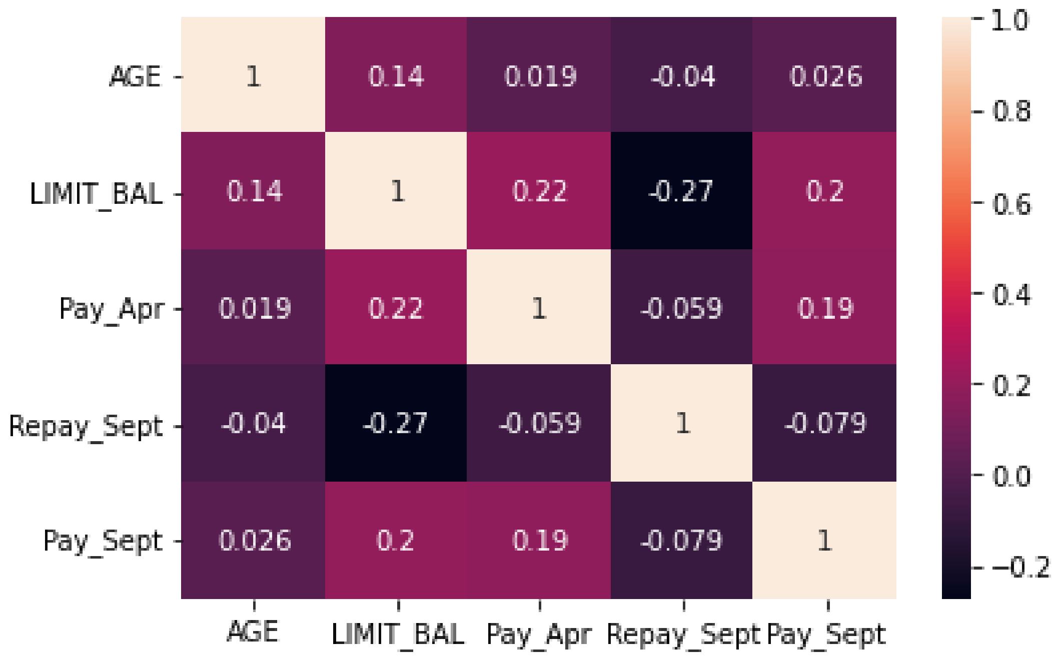


0.0.1 Now run LR with selected features from XGBoost

[ ]:
```
#In XGBoost feature importance, top 5 features appeared in the top-10 list for␣
↪both weight and gain. These features are selected here.
df_selected = df[['AGE', 'LIMIT_BAL', 'Pay_Apr', 'Repay_Sept',␣
↪'Pay_Sept','Default' ]]
```

[ ]:
```
#Seperate the independent and dependent variables.
df_independent = df_selected.drop(['Default'], axis=1)
df_default = df_selected['Default']
```

[ ]:
```
corr_matrix = df_independent.corr()
sns.heatmap(corr_matrix, annot=True);
# there is some correlation. But not a very strong correlation close to +1 or␣
↪-1.
```




```
[ ]: # split the data into 70% training + 30% test
     X_train, X_test, y_train, y_test = train_test_split(df_independent, df_default,␣
      ↪test_size=0.30, random_state=1)
```

```
[ ]: # Scale input variables for training
     X_train_scaled = preprocessing.MinMaxScaler().fit_transform(X_train)
```

```
[ ]: # Balancing the training data using SMOTE Tomek
     X_smt, y_smt = SMOTETomek(random_state=1).fit_sample(X_train_scaled, y_train.
      ↪squeeze())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

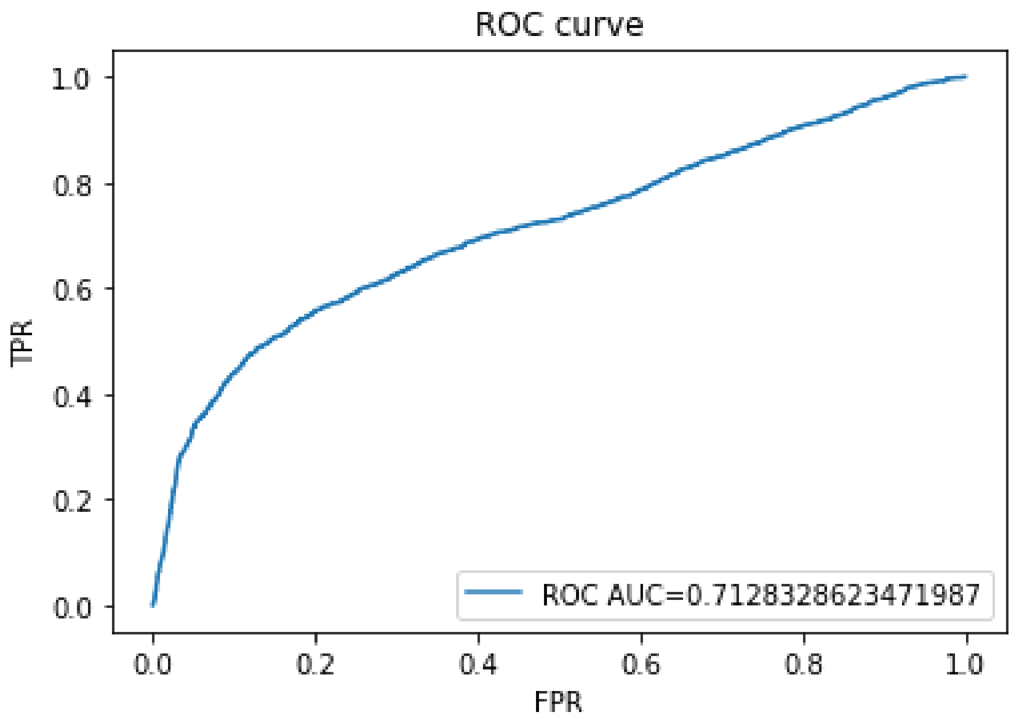
```
[ ]: # create model
     model = LogisticRegression(random_state=1, C= 50, penalty= 'l1', solver=␣
      ↪'liblinear')   #optimized parameters from GridSearchCV
     # Fit the model
     model.fit(X_smt, y_smt)
```

```
# Predict using the scaled X_test
X_test_scaled = preprocessing.MinMaxScaler().fit_transform(X_test)
y_pred = model.predict(X_test_scaled)
```

[ ]:
```
# performance metrics
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('accuracy', accuracy_score(y_test, y_pred))
print('precision', precision_score(y_test, y_pred))
print('recall', recall_score(y_test, y_pred))
```

```
[[5271 1710]
 [ 829 1180]]
accuracy 0.7175750834260289
precision 0.4083044982698962
recall 0.587356893977103
```

[ ]:
```
# ROC curve, and ROC AUC
y_pred_proba = model.predict_proba(X_test_scaled)[::,1]
FPR, TPR, _ = roc_curve(y_test, y_pred_proba)             #roc_curve(y_true,␣
 ↪y_score)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(FPR,TPR,label="ROC AUC="+str(auc))
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend(loc=4)
plt.show()
```



Feature selection: Use top-3 features

```
[ ]: # looking at Pearson correlations with the dependent variable, and XGBoost␣
     ↪feature importance 3 predictors always come up:
     df_selected2 = df[['LIMIT_BAL','Repay_Sept', 'Pay_Sept','Default' ]]
```

```
[ ]: #Seperate the independent and dependent variables.
     df_independent = df_selected2.drop(['Default'], axis=1)
     df_default = df_selected2['Default']
```

```
[ ]: # split the data into 70% training + 30% test
     X_train, X_test, y_train, y_test = train_test_split(df_independent, df_default,␣
     ↪test_size=0.30, random_state=1)
```

```
[ ]: # Scale input variables for training
     X_train_scaled = preprocessing.MinMaxScaler().fit_transform(X_train)
```

```
[ ]: # Balancing the training data using SMOTE Tomek
     X_smt, y_smt = SMOTETomek(random_state=1).fit_sample(X_train_scaled, y_train.
     ↪squeeze())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
```

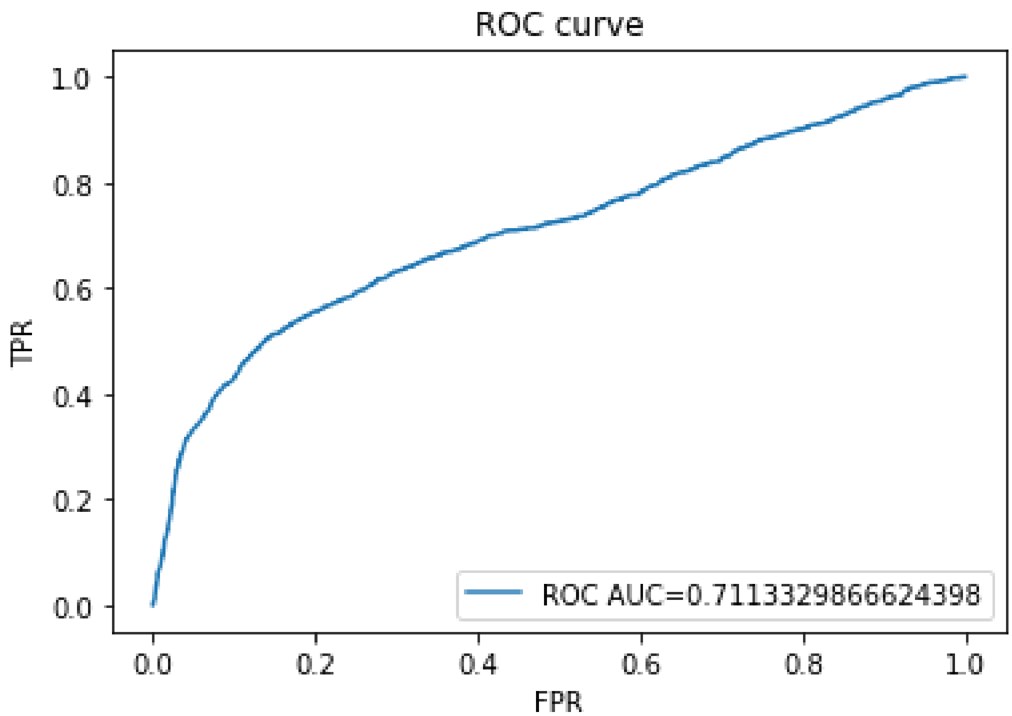
```
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

[ ]:
```
# create model
model = LogisticRegression(random_state=1, C= 50, penalty= 'l1', solver=␣
↪'liblinear')    #optimized parameters from GridSearchCV
# Fit the model
model.fit(X_smt, y_smt)
# Predict using the scaled X_test
X_test_scaled = preprocessing.MinMaxScaler().fit_transform(X_test)
y_pred = model.predict(X_test_scaled)
```

[ ]:
```
# performance metrics
cm = confusion_matrix(y_test, y_pred)
print(cm)
print('accuracy', accuracy_score(y_test, y_pred))
print('precision', precision_score(y_test, y_pred))
print('recall', recall_score(y_test, y_pred))
# Insight: can see the performance with these top 3 features is the same as top␣
↪5 features
```

```
[[5490 1491]
 [ 873 1136]]
accuracy 0.7370411568409344
precision 0.43243243243243246
recall 0.565455450472872
```

[ ]:
```
# ROC curve, and ROC AUC
y_pred_proba = model.predict_proba(X_test_scaled)[::,1]
FPR, TPR, _ = roc_curve(y_test, y_pred_proba)            #roc_curve(y_true,␣
↪y_score)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(FPR,TPR,label="ROC AUC="+str(auc))
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend(loc=4)
plt.show()
```


ROC curve
1.0
0.8
0.6
0.4
0.2
0.0
TPR
ROC AUC=0.7113329866624398
0.0
0.2
0.4
0.6
0.8
1.0
FPR