Credit Card Approval Prediction

Team 30, CS3244

To Approve or Not Approve?

Why choosing this?

- Enormous amount of application.
- Customer with what attributes default easier?

Our definition on credit card approval problem.

- Two-way Classification Problem.

Models Explored

Supervised Learning
- Logistic Regression
- Decision Tree
- Random Forest
- XGBoost
- AdaBoost

Unsupervised Learning
- K-Means Clustering

Labelling + EDA + Data imbalance

- Two files- application record ,the credit record file
- Defining bad customers as having overdue bills for more than two months

- Inner join on customer ID to append labels to the customer attributes

- The number of good customers is 10 times more than bad customers
- F1 score is the metric that we choose

Why Logistic Regression ?

- Binary classification problem
- Interested in predicted probabilities

Steps

- Stratify split data
- Scale features
- Use 10-fold cross validation on training data

Plain Logistic Regression

```
List of possible f1 score: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
training_results
```

	model	roc auc score	F1-score	precision	recall	tn	fp	fn	tp	class 0 accuracy	class 1 accuracy
0	plain	0.5	0.0	0.0	0.0	25805.0	0.0	190.0	0.0	1.0	0.0

```
testing_results
```

	model	roc auc score	F1-score	precision	recall	tn	fp	fn	tp	class 0 accuracy	class 1 accuracy
0	plain	0.5	0.0	0.0	0.0	7168.0	0.0	53.0	0.0	1.0	0.0

Is model learning well on how to separate classes linearly ? No

Does CV help? No

Weighted Logistic Regression

```
List of possible f1 score: [0.014814814814814815, 0.019097222222222224, 0.016625103906899942,
0.017108639863130880, 0.014084507042253521, 0.021998166819431710, 0.017953321364452428, 0.028286189683860232,
0.025359256128486900, 0.017035775127768313]

Maximum f1 score That can be obtained from this model is: 2.828618968386023 %
Minimum f1 score: 1.4084507042253522 %
Average f1 score: 1.9236299697332044 %
Standard Deviation is: 0.004606305756574825
```

training_results

	model	roc auc score	F1-score	precision	recall	tn	fp	fn	tp	class 0 accuracy	class 1 accuracy
0	plain	0.500000	0.000000	0.000000	0.000000	25805.0	0.0	190.0	0.0	1.000000	0.000000
1	plain + balanced	0.615936	0.022693	0.011556	0.626316	15627.0	10179.0	71.0	119.0	0.605557	0.626316

testing_results

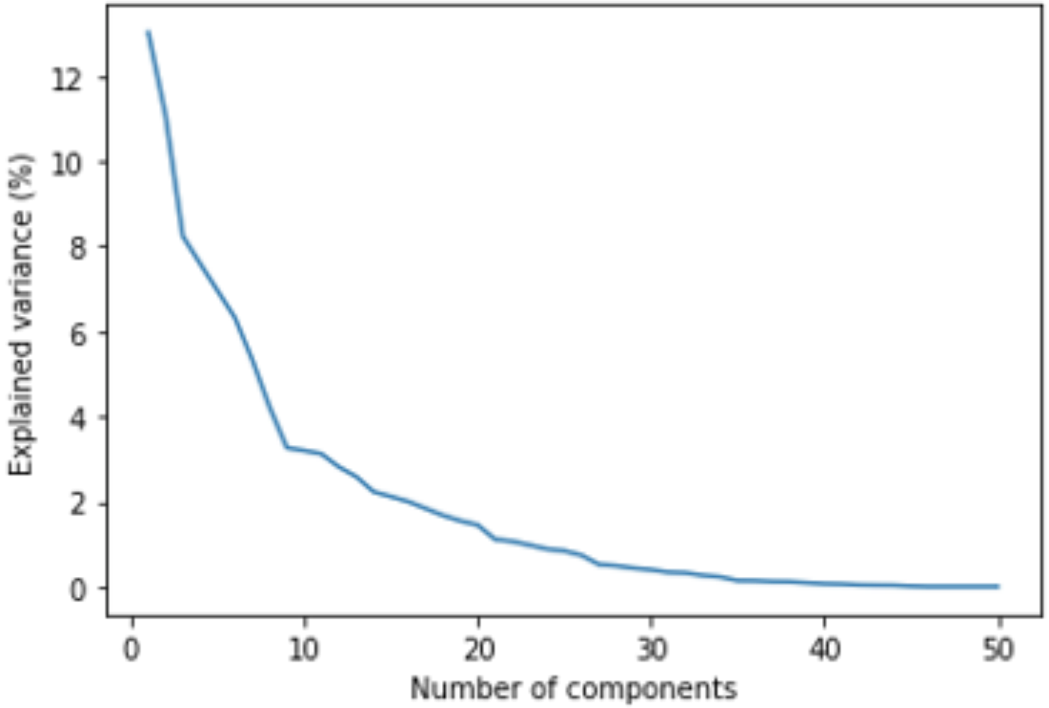
	model	roc auc score	F1-score	precision	recall	tn	fp	fn	tp	class 0 accuracy	class 1 accuracy
0	plain	0.500000	0.000000	0.000000	0.000000	7168.0	0.0	53.0	0.0	1.00000	0.000000
1	plain + balanced	0.524597	0.015981	0.008118	0.509434	3869.0	3299.0	26.0	27.0	0.53976	0.509434

PCA

Why Principal Components Analysis?

- Mitigate overfitting
- Not all 50 features are truly relevant predictors

1st 10 components used as predictors in logistic regression model



Weighted Logistic Regression using PCA components

```
List of possible f1 score: [0.013050570962479609, 0.014574898785425101, 0.019323671497584544,
0.01888276947285602, 0.014790468364831555, 0.018433179723502304, 0.013289036544850497,
0.021943573667711602, 0.01833460656990069, 0.016220600162206]

Maximum f1 score That can be obtained from this model is: 2.1943573667711602 %
Minimum f1 score: 1.3050570962479608 %
Average f1 score: 1.6884337575134793 %
Standard Deviation is: 0.0029406054009804886
```

```
training_results
```

	model	roc auc score	F1-score	precision	recall	tn	fp	fn	tp	class 0 accuracy	class 1 accuracy
0	plain	0.500000	0.000000	0.000000	0.000000	25805.0	0.0	190.0	0.0	1.000000	0.000000
1	plain + balanced	0.615936	0.022693	0.011556	0.626316	15627.0	10179.0	71.0	119.0	0.605557	0.626316
2	PCA + class_weight = balanced	0.584708	0.020104	0.010226	0.589474	14966.0	10840.0	78.0	112.0	0.579943	0.589474

```
testing_results
```

	model	roc auc score	F1-score	precision	recall	tn	fp	fn	tp	class 0 accuracy	class 1 accuracy
0	plain	0.500000	0.000000	0.000000	0.000000	7168.0	0.0	53.0	0.0	1.000000	0.000000
1	plain + balanced	0.524597	0.015981	0.008118	0.509434	3869.0	3299.0	26.0	27.0	0.539760	0.509434
2	PCA + class_weight = balanced	0.473835	0.012686	0.006452	0.377358	4088.0	3080.0	33.0	20.0	0.570312	0.377358

Decision Tree

- Labelling

```
[ ] from sklearn.model_selection import train_test_split

    df_X1 = pd.get_dummies(df[df.columns[df.columns != 'REALTYPE']].copy()) # get columns that are not 'good cx'
    df_X = pd.get_dummies(df_X1[df_X1.columns[df_X1.columns != 'ID']].copy())
    df_y = df['REALTYPE'].copy() # get the column named 'REALTYPE'; this is our label
```

- Split the data

```
[ ] X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.2, random_state=1)

    print ("Number of training instances: ", len(X_train), "\nNumber of test instances: ", len(X_test))

    Number of training instances:  28884
    Number of test instances:  7221
```

Normal Decision Tree

```
Decision Tree accuracy for test set: 0.988090
f1 score: 0.987985
recall score: 0.228070
precision score: 0.236364
AUC-ROC score: 0.611104
```

Near Miss Undersampling

```
Number of training instances:  422
Number of test instances:  106
Decision Tree accuracy for training set: 1.000000
Decision Tree accuracy for test set: 0.688679
f1 score: 0.689095
recall score: 0.606557
precision score: 0.804348
AUC-ROC score: 0.703279
```

- Compared to normal DT

```
Decision Tree accuracy for test set: 0.988090
f1 score: 0.987985
recall score: 0.228070
precision score: 0.236364
AUC-ROC score: 0.611104
```

SMOTE Oversamping

```
> k=1, Mean ROC AUC: 0.673
> k=2, Mean ROC AUC: 0.664
> k=3, Mean ROC AUC: 0.666
> k=4, Mean ROC AUC: 0.671
> k=5, Mean ROC AUC: 0.670
> k=6, Mean ROC AUC: 0.660
> k=7, Mean ROC AUC: 0.669
> k=44, Mean ROC AUC: 0.671
> k=60, Mean ROC AUC: 0.670
> k=99, Mean ROC AUC: 0.665
Decision Tree accuracy for training set: 1.000000
Decision Tree accuracy for test set: 0.688679
f1 score: 0.689095
recall score: 0.606557
precision score: 0.804348
```

- Compared to normal DT

```
Decision Tree accuracy for test set: 0.988090
f1 score: 0.987985
recall score: 0.228070
precision score: 0.236364
AUC-ROC score: 0.611104
```

Using Validation --- Near Miss Undersampling

```
Decision Tree accuracy for validation set: 0.773585
Decision Tree accuracy for test set: 0.641509
f1 score of valid set: 0.776025
f1 score of test set: 0.645479
recall score of valid set: 0.718750
recall score of test set: 0.600000
precision score of valid set: 0.884615
precision score of test set: 0.521739
AUC-ROC score of valid set: 0.787946
AUC-ROC score of test set: 0.633333
```

- Compared to not use validation

```
Number of training instances:  422
Number of test instances:  106
Decision Tree accuracy for training set: 1.000000
Decision Tree accuracy for test set: 0.688679
f1 score: 0.689095
recall score: 0.606557
precision score: 0.804348
AUC-ROC score: 0.703279
```

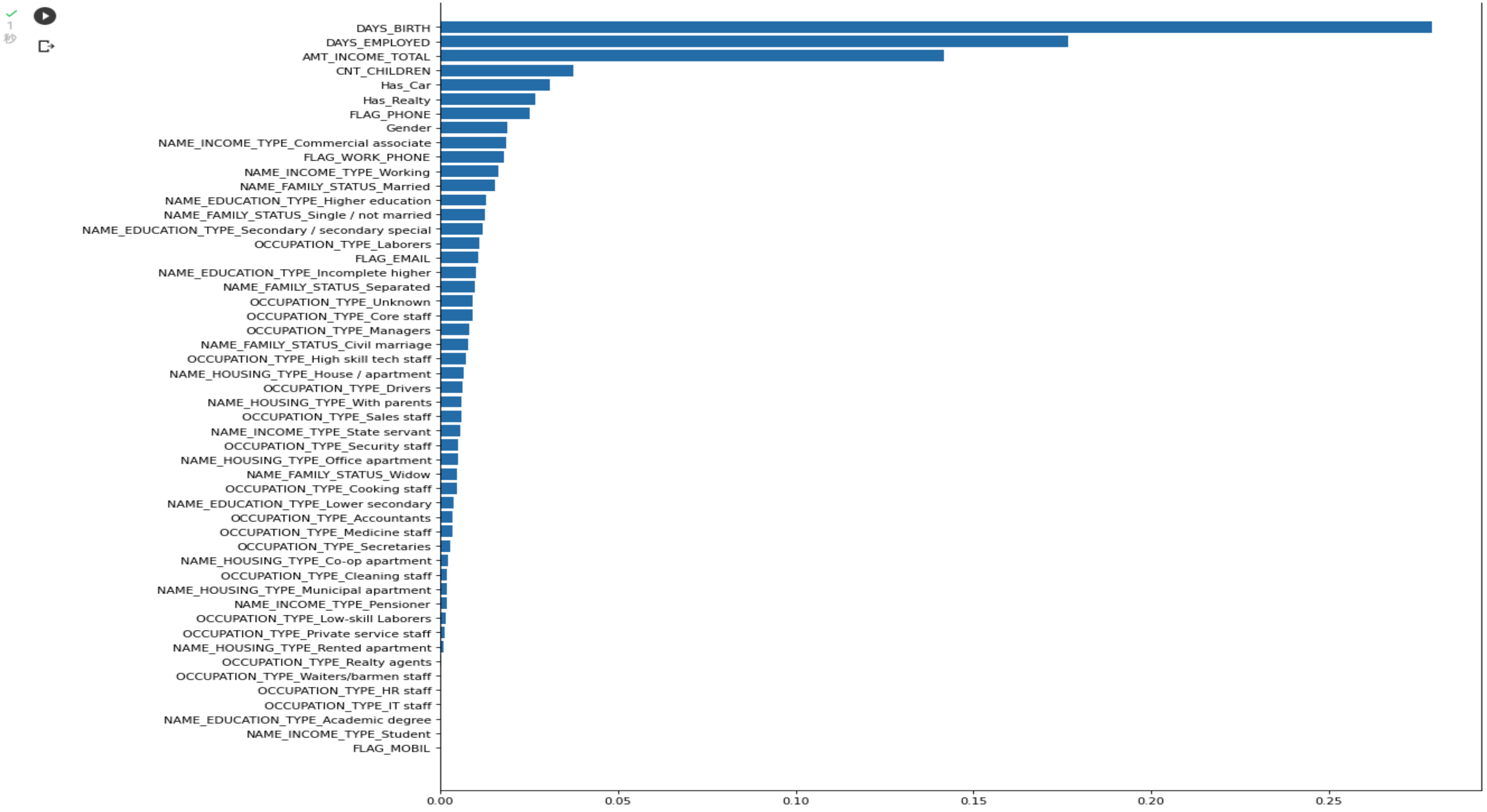
Random Forest - Feature engineering

- correlation matrix

	Unnamed: 0	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	CNT_FAM_MEMBERS	Gender	Has_Car	Has_Realty	REALTYPE
Unnamed: 0	1.000000	0.950773	0.037043	−0.016733	0.058097	−0.040471	nan	0.083650	0.006845	−0.048017	0.035642	0.020863	−0.008080	−0.111125	0.020895
ID	0.950773	1.000000	0.029996	−0.017381	0.055901	−0.038194	nan	0.078888	0.009575	−0.047188	0.027242	0.011766	−0.010905	−0.098977	0.017686
CNT_CHILDREN	0.037043	0.029996	1.000000	0.033238	0.339609	−0.229189	nan	0.048061	−0.016295	0.016385	0.889850	0.078192	0.106422	0.000188	0.007990
AMT_INCOME_TOTAL	−0.016733	−0.017381	0.033238	1.000000	0.066917	−0.168324	nan	−0.038247	0.016857	0.086561	0.023909	0.198129	0.216131	0.032830	−0.002821
DAYS_BIRTH	0.058097	0.055901	0.339609	0.066917	1.000000	−0.615817	nan	0.178106	−0.028719	0.105770	0.304320	0.201920	0.156924	−0.129997	−0.001559
DAYS_EMPLOYED	−0.040471	−0.038194	−0.229189	−0.168324	−0.615817	1.000000	nan	−0.242765	−0.007204	−0.085824	−0.220687	−0.173268	−0.156022	0.094322	0.002494
FLAG_MOBIL	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
FLAG_WORK_PHONE	0.083650	0.078888	0.048061	−0.038247	0.178106	−0.242765	nan	1.000000	0.311589	−0.034111	0.064024	0.065368	0.019884	−0.207539	0.010456
FLAG_PHONE	0.006845	0.009575	−0.016295	0.016857	−0.028719	−0.007204	nan	0.311589	1.000000	0.009984	−0.004320	−0.025526	−0.014698	−0.067176	0.014287
FLAG_EMAIL	−0.048017	−0.047188	0.016385	0.086561	0.105770	−0.085824	nan	−0.034111	0.009984	1.000000	0.014814	−0.003109	0.022022	0.052450	−0.001931
CNT_FAM_MEMBERS	0.035642	0.027242	0.889850	0.023909	0.304320	−0.220687	nan	0.064024	−0.004320	0.014814	1.000000	0.111601	0.151968	−0.005484	0.004753
Gender	0.020863	0.011766	0.078192	0.198129	0.201920	−0.173268	nan	0.065368	−0.025526	−0.003109	0.111601	1.000000	0.362489	−0.049729	−0.008137
Has_Car	−0.008080	−0.010905	0.106422	0.216131	0.156924	−0.156022	nan	0.019884	−0.014698	0.022022	0.151968	0.362489	1.000000	−0.014619	0.001120
Has_Realty	−0.111125	−0.098977	0.000188	0.032830	−0.129997	0.094322	nan	−0.207539	−0.067176	0.052450	−0.005484	−0.049729	−0.014619	1.000000	−0.006756
REALTYPE	0.020895	0.017686	0.007990	−0.002821	−0.001559	0.002494	nan	0.010456	0.014287	−0.001931	0.004753	−0.008137	0.001120	−0.006756	1.000000

Random Forest - Feature engineering

- feature importance plot



Random Forest - Oversampling and undersampling

- Without sampling method

```
F1-score: 0.2247191011235955
Accuracy 0.9923562645397142
Recall: 0.17857142857142858
Precision:  0.30303030303030304
Confusiont Matrix
 [[8948   23]
 [  46   10]]
tn:  8948
fp:  23
fn:  46
tp:  10
```

- Undersampling

```
F1-score: 0.16080402010050251
Accuracy 0.9814999446106126
Recall: 0.2857142857142857
Precision:  0.11188811188811189
Confusiont Matrix
 [[8844  127]
 [  40   16]]
tn:  8844
fp:  127
fn:  40
tp:  16
```

Random Forest - Oversampling and undersampling

- Oversampling

```
F1-score: 0.22900763358778628
Accuracy 0.9888113437465381
Recall: 0.26785714285714285
Precision:  0.2
Confusiont Matrix
 [[8911   60]
 [  41   15]]
tn:  8911
fp:  60
fn:  41
tp:  15
```

- Both

```
F1-score: 0.22413793103448276
Accuracy 0.9900299102691924
Recall: 0.23214285714285715
Precision:  0.21666666666666667
Confusiont Matrix
 [[8924   47]
 [  43   13]]
tn:  8924
fp:  47
fn:  43
tp:  13
```

Random Forest -

- Hyperparameter tuning

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [60, 80, 100, 120 ,140]
random_grid = {'n_estimators': n_estimators,
}
pprint(random_grid)
```

```
{'n_estimators': [60, 80, 100, 120, 140]}
```

```
[179] rf_random.best_params_
```

```
{'n_estimators': 80}
```

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators =80, class_weight = {0:0.2, 1:1})
rf_model = rf.fit(X_train, Y_train);
predictions = rf.predict(X_test)
f = f1_score(y_true = Y_test , y_pred = predictions)

print("F1-score:",f)
print("Accuracy",accuracy_score(y_true = Y_test , y_pred = predictions))
print("Recall:", recall_score(y_true = Y_test , y_pred = predictions))
print("Precision: ",precision_score (y_true = Y_test , y_pred = predictions))
print("Confusiont Matrix \n",confusion_matrix(y_true = Y_test , y_pred = predictions) )
tn, fp, fn, tp = confusion_matrix(y_true = Y_test , y_pred = predictions).ravel()
print("tn: ", tn)
print("fp: ", fp)
print("fn: ",fn )
print("tp: ", tp)
```

```
F1-score: 0.25423728813559326
Accuracy 0.9902514678187659
Recall: 0.26785714285714285
Precision:  0.24193548387096775
Confusiont Matrix
 [[8924   47]
 [  41   15]]
tn:  8924
fp:  47
fn:  41
tp:  15
```

Random Forest

- Lime

```
[172] import lime
      import lime.lime_tabular
```

```
predict_fn_rf = lambda x: rf.predict_proba(x).astype(float)
X = X_train.values
explainer = lime.lime_tabular.LimeTabularExplainer(X,feature_names = X_train.columns,class_names=['Good Customer','Bad Customer'],kernel_width=5)
```

```
choosen_instance = X_test.iloc[144]
exp = explainer.explain_instance(choosen_instance, predict_fn_rf, num_features=5)
exp.show_in_notebook(show_all=True)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  "X does not have valid feature names, but"
```

NAME_FAMILY_STATUS_Widow	0.00
NAME_HOUSING_TYPE_Co-op apartment	0.00
NAME_HOUSING_TYPE_House / apartment	0.00
NAME_HOUSING_TYPE_Municipal apartment	0.00
NAME_HOUSING_TYPE_Office apartment	1.00
NAME_HOUSING_TYPE_Rented apartment	0.00
NAME_HOUSING_TYPE_With parents	0.00
OCCUPATION_TYPE_Accountants	0.00
OCCUPATION_TYPE_Cleaning staff	0.00
OCCUPATION_TYPE_Cooking staff	0.00
OCCUPATION_TYPE_Core staff	0.00
OCCUPATION_TYPE_Drivers	1.00

XGBoost

- **Performed XGBoost using XGBClassifier**
- Previously used GridSearchCV for parameter tuning
- Later used RandomizedSearchCV

XGBClassifier

- GridSearchCV

```
param_test = {
    'scale_pos_weight':[20, 40, 60],
    'learning_rate':[1.0, 1.2, 1.4]
}

grid = GridSearchCV(estimator=XGBClassifier(**best_parameters),
                    param_grid=param_test, n_jobs=-1)
```

```
param_test = {
    'subsample':[0.8,0.9,1.0],
    'colsample_bytree':[0.8,0.9,1.0]
}

grid = GridSearchCV(estimator=XGBClassifier(**best_parameters),
                    param_grid=param_test, n_jobs=-1)
```

```
param_test = {
    'max_depth':[4,5,6],
    'min_child_weight':[2,3,4]
}

grid = GridSearchCV(estimator=XGBClassifier(**best_parameters),
                    param_grid=param_test, n_jobs=-1)
```

```
param_test = {
    'reg_alpha':[0.05,0.06,0.07],
    'gamma':[0,0.2,0.4]
}

grid = GridSearchCV(estimator=XGBClassifier(**best_parameters),
                    param_grid=param_test, n_jobs=-1)
```

XGBClassifier

- RandomSearchCV

```
# Define the search space
param_grid = {
    'scale_pos_weight': [20, 40, 60],
    'learning_rate': [0.6, 0.7, 0.8, 0.9, 1.0],
    'max_depth': [6, 7, 8, 9, 10],
    'min_child_weight': [1, 2, 3, 4, 5],
    'gamma': [0, 0.1, 0.2, 0.3, 0.4],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'reg_alpha': [0.05, 0.1, 0.5],
    'reg_lambda': [0.05, 0.1, 0.5]
    }
random = RandomizedSearchCV(estimator=XGBClassifier(),
                           param_distributions=param_grid,
                           n_iter=50, n_jobs=-1)
random.fit(x_train, y_train)
print(random.best_params_)
```

```
{'scale_pos_weight': 20, 'reg_lambda': 0.5, 'reg_alpha': 0.5, 'min_child_weight': 1, 'max_depth': 7, 'learning_rate': 0.6, 'gamma': 0.2, 'colsample_bytree': 1.0}
```

XGBClassifier

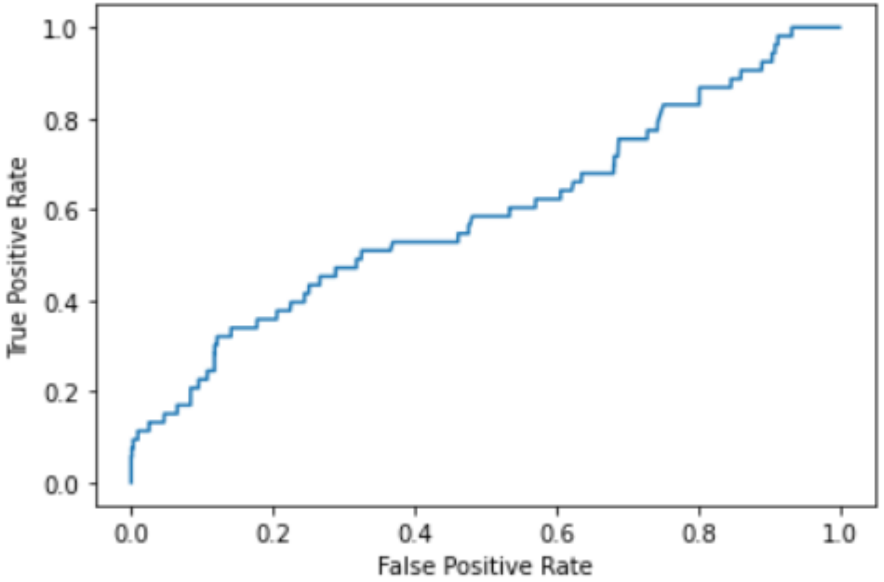
- No Hyperparameters

```
XGB = XGBClassifier()
XGB.fit(x_train, y_train)

preds = XGB.predict(x_test)
y_pred = XGB.predict_proba(x_test)[:,1]
FPR, TPR, _ = metrics.roc_curve(y_test,  y_pred)

plt.plot(FPR, TPR)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print("Train accuracy: ", metrics.accuracy_score(y_train, XGB.predict(x_train)))
print("Test accuracy: ", metrics.accuracy_score(y_test, preds))
print("Area Under ROC Curve: ", metrics.roc_auc_score(y_test, y_pred))
print("F1 score: ", metrics.f1_score(y_test, preds))
print("Recall: ", metrics.recall_score(y_test, preds))
print("Precision: ", metrics.precision_score(y_test, preds))
```




```
Train accuracy:  0.9926949176014402
Test accuracy:  0.9926602963578451
Area Under ROC Curve:  0.5894712874831537
F1 score:  0.0
Recall:  0.0
Precision:  0.0
```

XGBClassifier

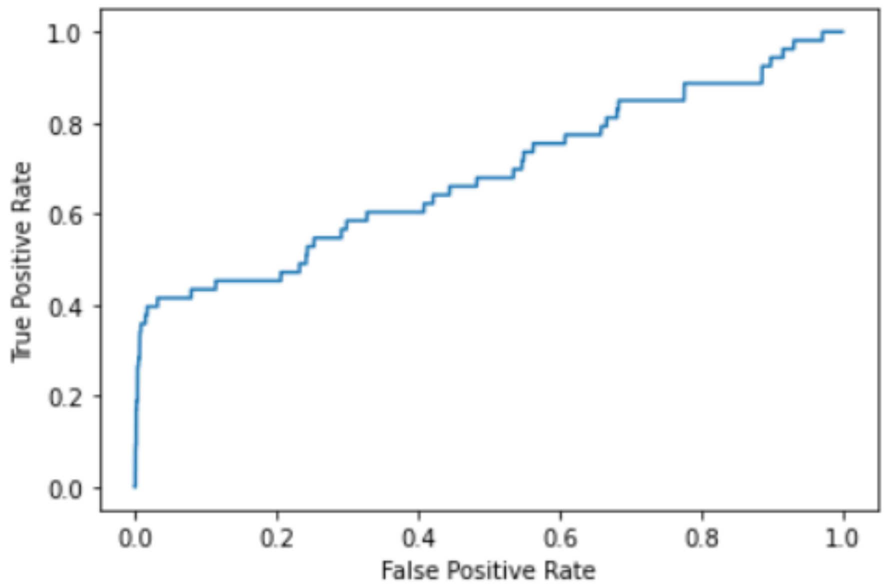
- With Hyperparameters

```
XGB = XGBClassifier(**random.best_params_)
XGB.fit(x_train, y_train)

preds = XGB.predict(x_test)
y_pred = XGB.predict_proba(x_test)[:,1]
FPR, TPR, _ = metrics.roc_curve(y_test,  y_pred)

plt.plot(FPR, TPR)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print("Train accuracy: ", metrics.accuracy_score(y_train, XGB.predict(x_train)))
print("Test accuracy: ", metrics.accuracy_score(y_test, preds))
print("Area Under ROC Curve: ", metrics.roc_auc_score(y_test, y_pred))
print("F1 score: ", metrics.f1_score(y_test, preds))
print("Recall: ", metrics.recall_score(y_test, preds))
print("Precision: ", metrics.precision_score(y_test, preds))
```




```
Train accuracy:  0.9847666528181692
Test accuracy:  0.9793657388173383
Area Under ROC Curve:  0.6849243493092992
F1 score:  0.21164021164021166
Recall:  0.37735849056603776
Precision:  0.14705882352941177
```

XGBClassifier

- Previously Done
  - SMOTE Oversampling
  - Random Undersampling
  - Data Scaling
  - Checking feature importance
  - Removing highly correlated variables

AdaBoost

Why AdaBoost?

- Logistic Regression benefitted from sample weighing
- Utilizes weighing of misclassified instances
- Subsequent weak classifiers to focus on these instances

AdaBoost

Base Classifier with RandomizedSearchCV (AdaBoostBCCV)

Without Base Classifier

```
Accuracy 0.9926949245825939
Precision 0.0
Recall 0.0
F1 0.0
Roc_Auc 0.5810127329337962
```

With Base Classifier and Tuning

Class weights = **{0:0.01, 1:1}**

```
Accuracy 0.9913677968490665
Precision 0.30491129241129244
Recall 0.1121212121212121
F1 0.16010501477038944
Roc_Auc 0.677244261518841
```

AdaBoost-DT

KernelPCA / Recursive Feature Elimination (RFE)

PCA

```
Accuracy 0.9861976941862773
Precision 0.1400958364095410 7
Recall 0.17207792207792202
F1 0.152433524211597 3
Roc_Auc 0.675573973102041
```

KernelPCA

```
Accuracy 0.9889212217066871
Precision 0.26452418891399554
Recall 0.28542568542568536
F1 0.2680150607152683
Roc_Auc 0.7039619992993796
```

Recursive Feature Elimination

```
Accuracy 0.9869939492525378
Precision 0.18270564513205034
Recall 0.2240981240981241
F1 0.1992988749214916
Roc_Auc 0.6831921535695579
```

AdaBoost-DT

KernelPCA / Recursive Feature Elimination

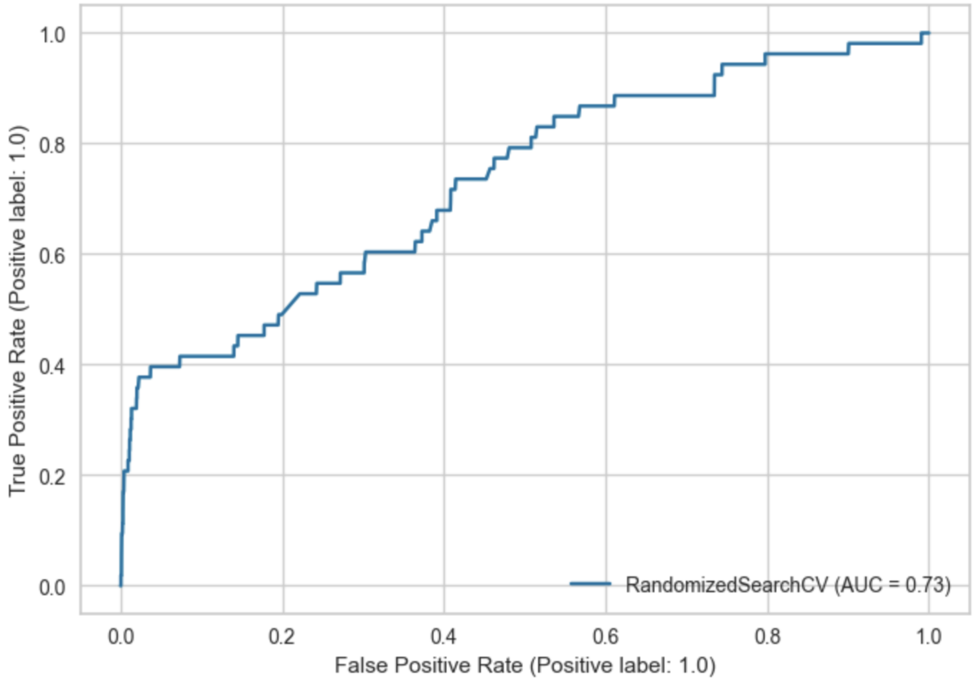
KernelPCA + RFE

```
Accuracy 0.9880672685612668
Precision 0.23197551982663298
Recall 0.2743867243867244
F1 0.2478164826613361
Roc_Auc 0.7068088583023173
```

KernelPCA + RFE + SMOTE

```
Accuracy 0.9833989934667274
Precision 0.9728865205957112
Recall 0.9945593357264254
F1 0.9835915291237955
Roc_Auc 0.9967119072337304
```

Evaluation on Test Set




```
Precision:  0.11242603550295859
Recall:  0.3584905660377358
Confusion Matrix:  [[7018  150]
 [  34   19]]
F1 Score:  0.17117117117117117
ROC AUC Score:  0.6687821133760107
```

K-Means Clustering

Trying unsupervised ML methods.

See the connection of Cluster labels with Ground Truth

K-Means Clustering

Evaluating Matrics used

- Davis Bouldin Score
- Silhouette Coefficient Score

Indicating a good split

```
from sklearn.metrics import davies_bouldin_score
score5 = davies_bouldin_score(df_X, kmeans_1.labels_)
score6 = davies_bouldin_score(X_nm, kmeans_nm.labels_)
print(score5, score6)
```

```
0.35665999644976126 0.29449170174270206
```

```
from sklearn.metrics import silhouette_score
score1 = silhouette_score(df_X,kmeans_1.labels_,metric='euclidean')
score2 = silhouette_score(X_nm, kmeans_nm.labels_,metric='euclidean')
print(score1, score2)
```

```
0.7484033688427337 0.7884076559030081
```

K-Means Clustering

Comparing with Ground Truth

-Rand Score

For minority class

- no better than Random Guess

```
from sklearn.metrics import rand_score

score3 = rand_score(df_y, kmeans_1.labels_)
score4 = rand_score(y_nm, kmeans_nm.labels_)
print(score3, score4)
```

```
0.7141097317132095 0.4990512333965844
```

Thank You!