EvaluationUsingPrecisionRecallAndROC

October 14, 2021

1 Classifier Evaluation using Precision, Recall, and ROC

We are going to apply logistic regression using the data available from the Lending Club Corporation. The company provides files containing complete loan data for all loans issued through a certain time period, including the current loan status (Current, Late, Fully Paid, etc.) and latest payment information.

We are going to use a small subset of the available data which were previously preprocessed.

The goal of this notebook is to explore the different ways we can use to evaluate the result of classification.

As the very first step, we load all the relevant libraries.

[ ]:
```
# Usual libraries for tools
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn import model_selection
import matplotlib.pyplot as plt

# Libraries for the evaluation
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler

%matplotlib inline
```

Then, we load the data.

[ ]:
```
# loans = pd.read_csv('LoansData01.csv')
loans = pd.read_csv('LoansNumerical.csv')
```

[ ]:
```
target = 'safe_loans'
features = loans.columns[loans.columns!=target]

features
```

[ ]:
```
Index(['sub_grade_num', 'short_emp', 'emp_length_num', 'dti',
       'payment_inc_ratio', 'delinq_2yrs', 'delinq_2yrs_zero',
       'inq_last_6mths', 'last_delinq_none', 'last_major_derog_none',
       'open_acc', 'pub_rec', 'pub_rec_zero', 'revol_util',
       'total_rec_late_fee', 'int_rate', 'total_rec_int', 'annual_inc',
       'funded_amnt', 'funded_amnt_inv', 'installment', 'num_term',
       'grade_num', 'loan_amnt'],
      dtype='object')
```

From the data dictionary available on the Lending Club Corporation website, we have the description of the variables:

- sub_grade_num, the sub-grade of the loan as a number from 0 to 1
- short_emp, one year or less of employment
- emp_length_num, number of years of employment
- dti, debt to income ratio
- payment_inc_ratio, ratio of the monthly payment to income
- delinq_2yrs, number of delinquincies
- delinq_2yrs_zero, no delinquincies in last 2 years
- inq_last_6mths, number of creditor inquiries in last 6 months
- last_delinq_none, has borrower had a delinquincy
- last_major_derog_none, has borrower had 90 day or worse rating
- open_acc, number of open credit accounts
- pub_rec, number of derogatory public records
- pub_rec_zero, no derogatory public records
- revol_util, percent of available credit being used
- total_rec_late_fee, total late fees received to day
- int_rate, interest rate of the loan
- total_rec_int, interest received to date
- annual_inc, annual income of borrower
- funded_amnt, amount committed to the loan
- funded_amnt_inv, amount committed by investors for the loan
- installment, monthly payment owed by the borrower
- num_term, number of payments on the loan. Values are in months and can be either 36 or 60
- grade_num, LC assigned loan grade as a number
- loan_amnt, the listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

The target variable (the class) is safe_loans that is +1 if the loan is safe -1 if it is risky

The input variables x are the columns corresponding to the features, the output variable y is the column corresponding to the target variable.

[ ]:
```
X = loans[features]
y = loans[target]

# we need to normalize X for applying logistic regression
Xn = StandardScaler().fit_transform(X)
```

First we apply plain logistic regression without regularization (thus α is zero). The Scikit-learn function does not allow to specify α but it uses a parameter C=$1/\alpha$. Accordingly, to have no regularization we need to specify a huge value of C. And we evaluate the model using plain cross-validation.

[ ]:
```
simple_logistic = linear_model.LogisticRegression(C=10e10)
simple_logistic.fit(Xn,y);
simple_eval = model_selection.cross_val_score(simple_logistic, Xn, y,␣
 ↪cv=StratifiedKFold(10,shuffle=True,random_state=1234))
print("Simple Logistic Regression\t%3.2f\t%3.2f" % (np.average(simple_eval), np.
 ↪std(simple_eval)))
```

```
Simple Logistic Regression      0.81    0.00
```

1.1 Evaluating Classification - Accuracy

Accuracy is the most known and most widely used measure of classification performance. It measure the percentage of correct classification achieved by the model.

$accuracy = \frac{number of correclty classified examples}{number of examples}$

1.2 Confusion Matrix

Accuracy evaluates the percentage of correct predictions but not the errors are equal. To have a better sense of how a classifier model performs the confusion matrix is usually examined. The confusion matrix includes

- the true positives, the number of examples labeled as positives and prediceted as positives
- the true negatives, the number of examples labeled as negatives and prediceted as negatives
- the false positives, the number of examples labeled as negatives and prediceted as positives
- the false negatives, the number of examples labeled as positives and prediceted as negatives

The confusion matrix is typically shown as,

	Predicted Positive	Predicted Negative
Labeled Positive	TP	FN
Labeled Negative	FP	TN

We can compute the confusion matrix for the three models we developed.

[ ]:
```
yp = simple_logistic.predict(X);

def PrintConfusionMatrix(model, true_y, predicted_y, positive=1, negative=-1):
```

```
    cm = confusion_matrix(true_y,predicted_y)
    print("\t"+str(model.classes_[0])+"\t"+str(model.classes_[1]))
    print(str(model.classes_[0]) + "\t",cm[0][0],"\t",cm[0][1])
    print(str(model.classes_[1]) + "\t",cm[1][0],"\t",cm[1][1])

print("Confusion Matrix - Simple Logistic")
PrintConfusionMatrix(simple_logistic, y, yp)
```

```
Confusion Matrix - Simple Logistic
        -1      1
-1       413     22709
1        888     98452
```

1.3 Precision and Recall

Precision and recall are alternative measures to plain accuracy introduced in the area of information retrieval and search engine. Precision focuses on the percentage of correctly classified positive examples or in the information retrieval context represents the percentage of actually good documents that have been shown as a result. Recall focuses on the percentage of positively classified examples with respect to the number of existing good documents or in the information retrieval context, recall represents the percentage of good documents shown with respect to the existing ones.

$precision = \frac{TP}{TP+FP}$

$recall = \frac{TP}{TP+FN}$

[ ]:
```
print("Precision %3.2f" % precision_score(y,yp))
print("Recall    %3.2f" % recall_score(y,yp))
```

```
Precision 0.81
Recall    0.99
```

1.4 Probabilistic Models & Classification Thresholds

Up to now we used logistic regression to predict classifier labels, however, logistic regression typically returns a probability that an example should be label using a certain class. For example, the first example, is associated to two probabilities one corresponding to label -1 and one corresponding to label +1.

[ ]:
```
X.head(1)
```

[ ]:
```
   sub_grade_num  short_emp  emp_length_num    dti  payment_inc_ratio  \
0            0.4          0              11  27.65             8.1435

   delinq_2yrs  delinq_2yrs_zero  inq_last_6mths  last_delinq_none  \
0          0.0               1.0             1.0                 1

   last_major_derog_none  …  total_rec_late_fee  int_rate  total_rec_int  \
0                      1  …                 0.0     10.65         861.07
```

```
   annual_inc  funded_amnt  funded_amnt_inv  installment  num_term  grade_num  \
0     24000.0         5000             4975       162.87        36          5

   loan_amnt
0       5000

[1 rows x 24 columns]
```

[ ]:
```
yp = simple_logistic.predict(Xn)
yprob = simple_logistic.predict_proba(Xn)
print("Example %d:\n\tP(-1|x[%d])=%3.2f\n\tP(+1|x[%d])=%3.2f\n\t=> Labeled as␣
 ↪%d" %(0 , 0, yprob[0,0], 0, yprob[0,1], yp[0]))
```

```
Example 0:
        P(-1|x[0])=0.20
        P(+1|x[0])=0.80
        => Labeled as 1
```

The class assignment is based on the label with the largest probability so it is equivalent to using a threshold of 0.5 to decide which class to assign to an example. However, we might decide to use a different threshold and for instance label as positive only examples with a P(+1|x)>0.75 this would label as positive only cases for which we are more confident that should be labeled as positive.

[ ]:
```
def ClassifyWithThreshold(probabilities, threshold):
    return [+1 if x>=threshold else -1 for x in probabilities]

yp_confident = ClassifyWithThreshold(yprob[:,1],0.75)

print("Confusion Matrix - Simple Logistic (0.75)")
PrintConfusionMatrix(simple_logistic, y, yp_confident)

print("\n")
print("Precision %3.2f" % precision_score(y,yp_confident))
print("Recall    %3.2f" % recall_score(y,yp_confident))
```

```
Confusion Matrix - Simple Logistic (0.75)
        -1      1
-1       10837   12285
1        18038   81302


Precision 0.87
Recall    0.82
```

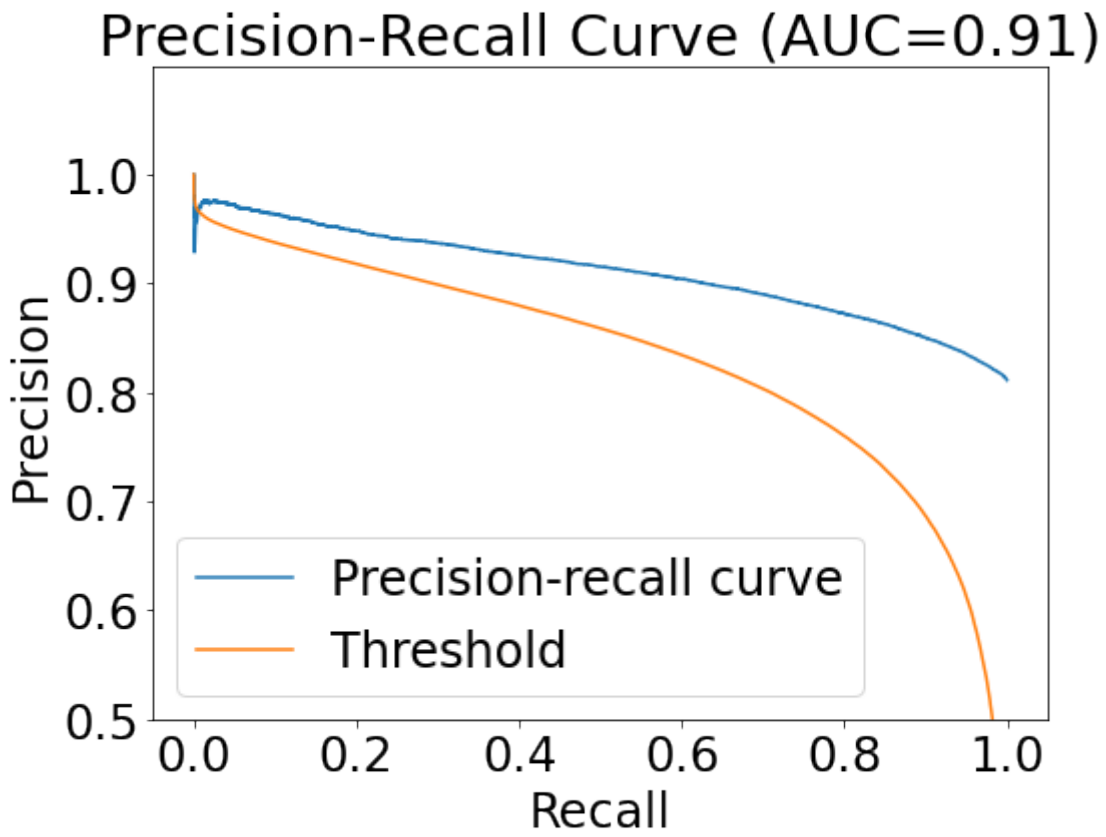
By increasing the threshold we increased the precision and thus we decreased the number of false positives while we increased the number of false negatives. Thus by modifying the threshold we can change the way our classifier makes mistakes.

1.5 Precision-Recall Curve

By modifying the threshold we can optimize our classifier to focus on one of the two metrics (precision or recall). To analyze precision varies depending on the threshold we can use the precision-recall curve which is computed by computing precision and recall from the threshold of 1 to the 0 threshold and plotting the result.

[ ]:
```
y_true = y
yp = simple_logistic.predict(Xn)
yprob = simple_logistic.predict_proba(Xn)
precision, recall, thresholds = precision_recall_curve(y_true=y,␣
 ↪probas_pred=yprob[:,1])
auc = average_precision_score(y, yprob[:,1])
```

[ ]:
```
plt.figure(1, figsize=(8, 6));
font = {'family':'sans', 'size':24};
plt.rc('font', **font);
plt.plot(recall, precision, label="Precision-recall curve");
plt.xlabel('Recall');
plt.ylabel('Precision');
plt.ylim([0.5,1.1])
plt.yticks(np.arange(0.5,1.01,.1))
plt.title('Precision-Recall Curve (AUC=%3.2f)'%auc);
plt.plot(recall[:-1],thresholds, label="Threshold");
plt.legend()
plt.show()
```



1.6 Receiver Operating Characteristic (ROC) Curves

Similar to precision-recall curves, they plot the True Positive Rate (TPR) against the False Positive Rate (FPR)

$TPR = \frac{TP}{TP+FN}$

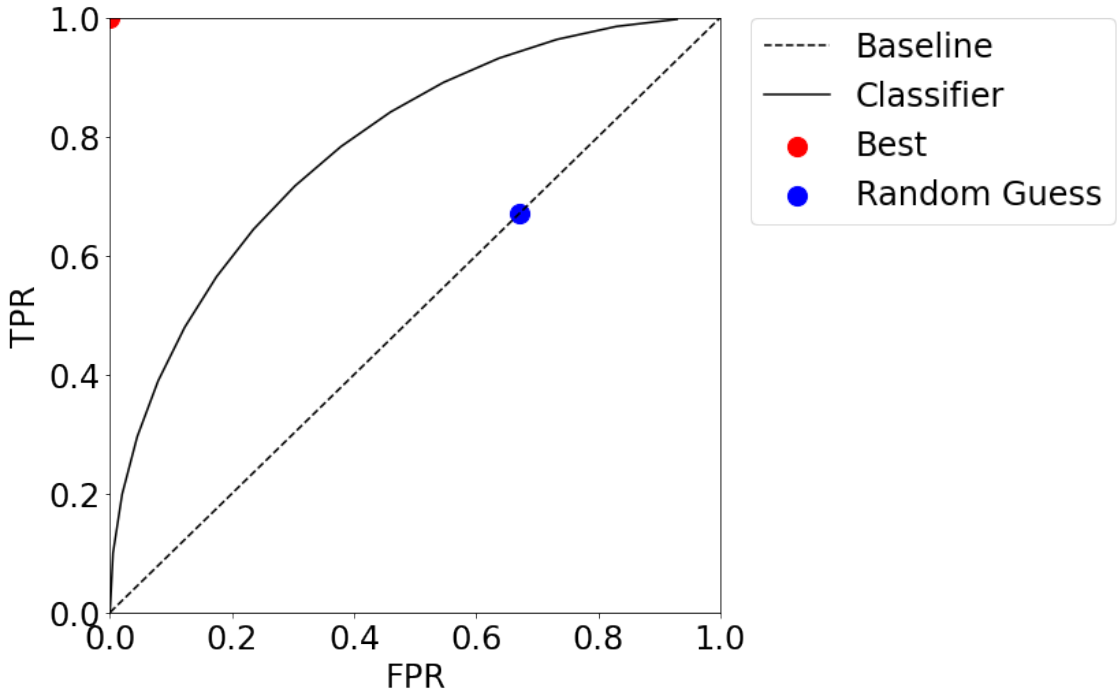
$FPR = \frac{FP}{TN+FP}$

[ ]:
```
import math

plt.figure(1, figsize=(8, 8));
font = {'family':'sans', 'size':24};
plt.rc('font', **font);
plt.xlabel('FPR');
plt.ylabel('TPR');
plt.ylim([0.0,1.0])
plt.xlim([0.0,1.0])
plt.plot([0.0,1.0],[0.0,1.0],color='black',ls='--',label='Baseline')
angle=np.arange(math.pi, math.pi/2,-0.1)
fpr = np.cos(angle)+1.
```

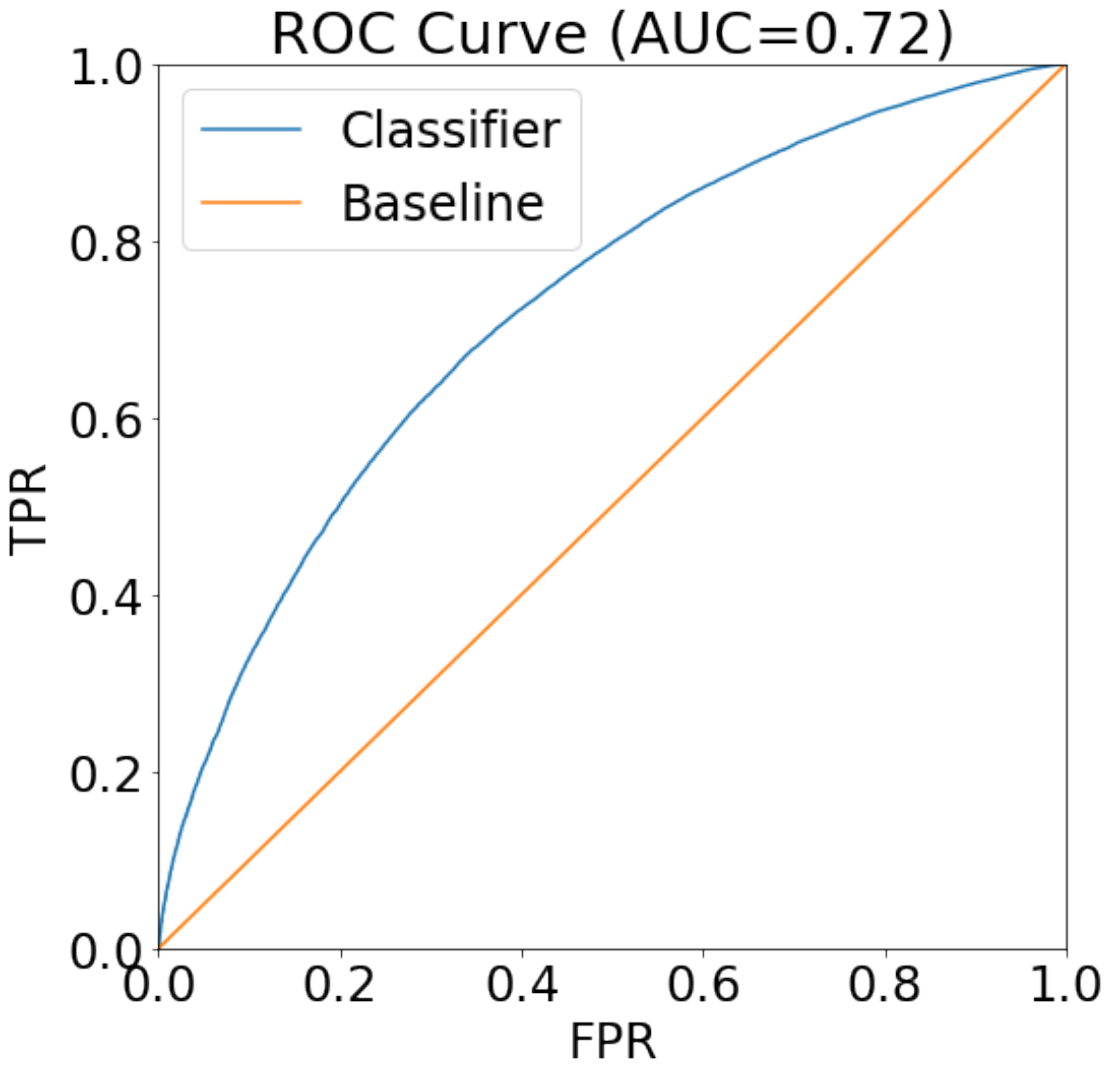
```
tpr = np.sin(angle)
plt.plot(fpr,tpr,label='Classifier',color='black')
plt.yticks(np.arange(0.0,1.01,.2))
plt.scatter([0.0],[1.0], s=200, color='red', label='Best')
plt.scatter([0.67],[0.67], s=200, color='blue', label='Random Guess')


# plt.legend()
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



[ ]:
```
fpr, tpr, thresholds = roc_curve(y_true=y, y_score = yprob[:,1], pos_label=1)
roc_auc = roc_auc_score(y_true=y, y_score = yprob[:,1])
plt.figure(1, figsize=(8, 8));
font = {'family':'sans', 'size':24};
plt.rc('font', **font);
plt.xlabel('FPR');
plt.ylabel('TPR');
plt.plot(fpr,tpr,label='Classifier')
# plt.plot(fpr,thresholds,label='Thresholds')
plt.plot([0.0,1.0],[0.0,1.0],label='Baseline')
plt.yticks(np.arange(0.0,1.01,.2))
plt.title('ROC Curve (AUC=%3.2f)'%roc_auc)
plt.ylim([0.0,1.0])
```

```
plt.xlim([0.0,1.0])
plt.legend()
plt.show();
```



As for the precision-recall curves, we can use the threshold value to search for the best TPR/FPR tradeoff.

The ideal value of AUC for a ROC curve is one however, it never happens. According, we look for classifiers with an AUC as large as possible and at least greater than Ideally, the AUC for ROC ion-recall curves, thus when selecting with two models we compare their AUC.

[ ]: