MACHINE LEARNING & CLASSIFICATION ALGORITHMS

Loan Approval Using Machine Learning Algorithm

Classification algorithms to determine application outcome

Sarit Maitra
Oct 16, 2020 · 8 min read ★

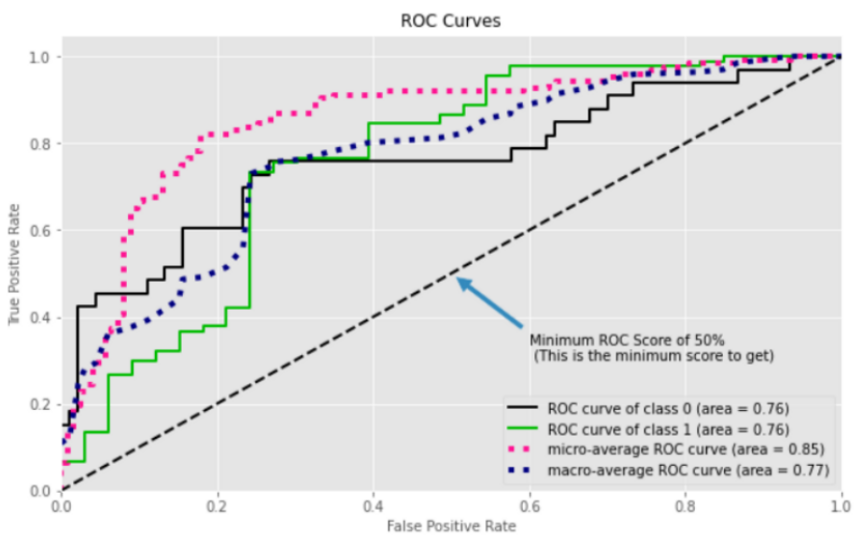


Image by author

Loans in terms of financial pay outs is an important aspect of banking business system. Several loan applications are scanned based on certain inputs to validate the eligibility for loan. Here our use-case is that, we want to automate the loan eligibility process (real time) based on customer detail obtained during loan application. This will lead to improved service and customer satisfaction.

Let us load the available data to check the information it contain.

Loading training data:

```
# Import data
df = pd.read_csv("loan_train.csv")
print('Information on dataset:')
df.info()
```

```
Information on dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
```

```
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

Here that the dependent / target variable is the Loan_Status and we need to develop a model using the rest of the features to predict the target variable.

```
df.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

Data Pre-processing:

```
df.Dependents.value_counts()
```

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

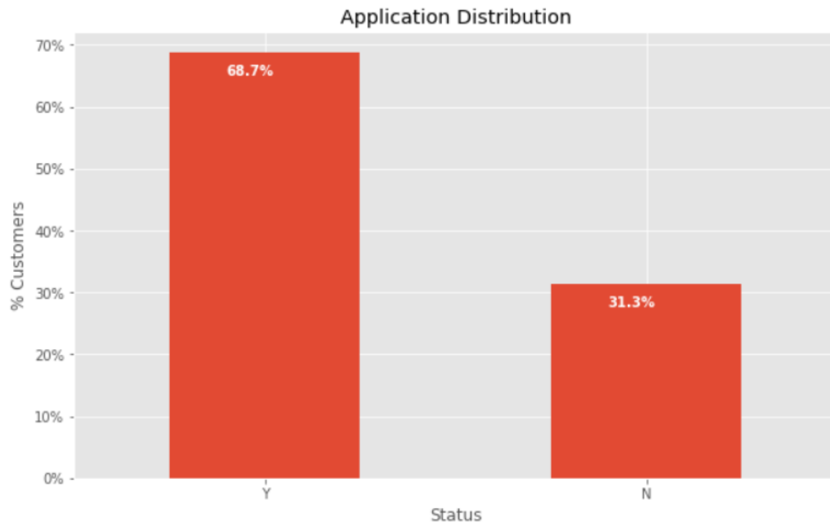
```
# replacing 3+ in Dependents variable with 3
df['Dependents'].replace('3+', 3, inplace=True)
df.Dependents.value_counts()
```

```
0    345
1    102
2    101
3     51
Name: Dependents, dtype: int64
```

Bar plot to visualize application outcome:

```
ax = (df['Loan_Status'].value_counts()*100.0
/len(df)).plot(kind='bar', stacked = True, rot = 0)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('% Loan application')
ax.set_xlabel('Status')
ax.set_ylabel('% Customers')
ax.set_title('Application Distribution')
totals = []

# finding the values and append to list
for i in ax.patches:
```

```
  totals.append(i.get_width())
total = sum(totals)
for i in ax.patches:
  ax.text(i.get_x()+.15, i.get_height()-3.5, \
str(round((i.get_height()/total), 1))+'%', color='white', weight =
'bold')
```



Missing value and outlier treatment:

```
# check for missing values
df.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

- numerical variables: imputation using mean or median
- categorical variables: imputation using mode
- There are very less missing values in Gender, Married, Dependents,
- Credit_History and Self_Employed features so we fill them using the mode of the features.
- If an independent variable in our dataset has huge amount of missing data e.g. 80% missing values in it, then we would drop the variable from the dataset.

```
# replace missing values with the mode
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0],
inplace=True)
```

```
df['Credit_History'].fillna(df['Credit_History'].mode()[0],
inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0],
inplace=True)

# replace missing values with the median value due to outliers
df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)

# replacing Y and N in Loan_Status variable with 1 and 0
df['Loan_Status'].replace('N', 0, inplace=True)
df['Loan_Status'].replace('Y', 1, inplace=True)
```

Loading test data:

```
# Import data
df1 = pd.read_csv("loan_test.csv")
print('Information on dataset:')
df1.info()
```

```
Information on dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            367 non-null    object
 1   Gender             356 non-null    object
 2   Married            367 non-null    object
 3   Dependents         357 non-null    object
 4   Education          367 non-null    object
 5   Self_Employed      344 non-null    object
 6   ApplicantIncome    367 non-null    int64
 7   CoapplicantIncome  367 non-null    int64
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

Test data processing:

```
# replacing 3+ in Dependents variable with 3
df1['Dependents'].replace('3+', 3, inplace=True)

# replace missing values in Test set with mode/median from Training
set
df1['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df1['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df1['Self_Employed'].fillna(df['Self_Employed'].mode()[0],
inplace=True)
df1['Credit_History'].fillna(df['Credit_History'].mode()[0],
inplace=True)
df1['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0],
inplace=True)
df1['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)
```

Model Development and Evaluation:

```
# drop Loan_ID
train = df.drop('Loan_ID', axis=1) # train
test = df1.drop('Loan_ID', axis=1) # test

# drop "Loan_Status" and assign it to target variable
X = train.drop('Loan_Status', 1)
y = train.Loan_Status
```

```
# adding dummies to the dataset
X = pd.get_dummies(X)
train = pd.get_dummies(train)
test = pd.get_dummies(test)

print(X.shape, train.shape, test.shape)
```

```
(614, 20) (614, 21) (367, 20)
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((491, 20), (123, 20), (491,), (123,))
```

Overfitting:

Less than half of applications classified as not accepted, therefore perform stratified cv would be a better approach to avoid overfitting. With this approach, we will have not one estimate but 5 estimates for the generalization error.

We shall try several classification algorithms and select the best one based on performance score.

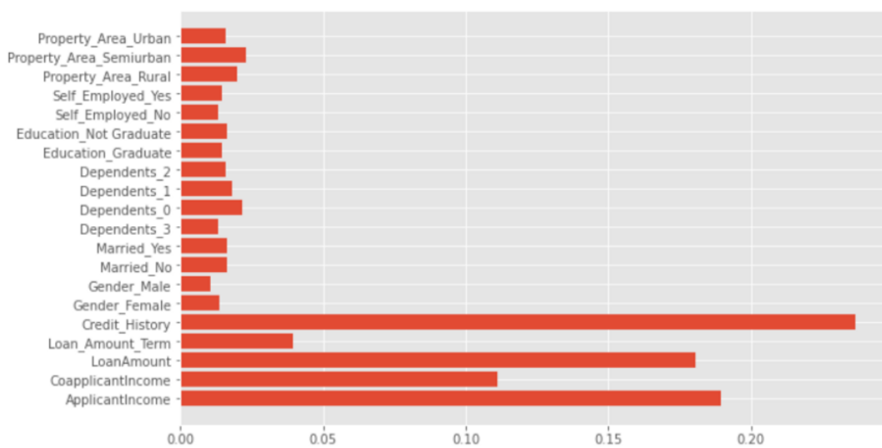
```
kfold = StratifiedKFold(n_splits=5, random_state=42)

# Logistic Regression
log_reg = LogisticRegression(solver='lbfgs', max_iter=5000)
log_scores = cross_val_score(log_reg, x_train, y_train, cv=kfold)
log_reg_mean = log_scores.mean()

# SVC
svc_clf = SVC(gamma='auto')
svc_scores = cross_val_score(svc_clf, x_train, y_train, cv=kfold)
svc_mean = svc_scores.mean()

# KNearestNeighbors
knn_clf = KNeighborsClassifier()
knn_scores = cross_val_score(knn_clf, x_train, y_train, cv=kfold)
knn_mean = knn_scores.mean()

# Decision Tree
tree_clf = tree.DecisionTreeClassifier()
tree_scores = cross_val_score(tree_clf, x_train, y_train, cv=kfold)
tree_mean = tree_scores.mean()

# Gradient Boosting Classifier
grad_clf = GradientBoostingClassifier()
grad_scores = cross_val_score(grad_clf, x_train, y_train, cv=kfold)
grad_mean = grad_scores.mean()

# Random Forest Classifier
rand_clf = RandomForestClassifier(n_estimators=100)
rand_scores = cross_val_score(rand_clf, x_train, y_train, cv=kfold)
rand_mean = rand_scores.mean()

# NeuralNet Classifier
neural_clf = MLPClassifier(alpha=1)
neural_scores = cross_val_score(neural_clf, x_train, y_train,
cv=kfold)
neural_mean = neural_scores.mean()

# Naives Bayes
nav_clf = GaussianNB()
nav_scores = cross_val_score(nav_clf, x_train, y_train, cv=kfold)
nav_mean = neural_scores.mean()

# Create a Dataframe with the results.
d = {'Classifiers': ['Logistic Reg.', 'SVC', 'KNN', 'Dec Tree', 'Grad
B CLF', 'Rand FC', 'Neural Classifier', 'Naives Bayes'],
'Crossval Mean Scores': [log_reg_mean, svc_mean, knn_mean, tree_mean,
grad_mean, rand_mean, neural_mean, nav_mean]}
```

```
result_df = pd.DataFrame(data=d)

# Log Reg performed best
result_df = result_df.sort_values(by=['Crossval Mean Scores'],
ascending=False)
print(result_df)
```

	Classifiers	Crossval Mean Scores
0	Logistic Reg.	0.806473
5	Rand FC	0.794187
4	Grad B CLF	0.769924
3	Dec Tree	0.698598
1	SVC	0.674129
2	KNN	0.645599
6	Neural Classifier	0.572171
7	Naives Bayes	0.572171

Feature importance:

```
# Random Forest Classifier
rand_clf = RandomForestClassifier(n_estimators=100).fit(x_train,
y_train)

# get importance
plt.barh(X.columns, rand_clf.feature_importances_)
plt.show()
```



We can see here that credit history, loan amount, co-applicant's income and applicant's income plays significant roles in determining the approval. This kind of fits our mental model too.

Confusion Matrix:

The main purpose of a confusion matrix is to see how our model is performing when it comes to classifying potential clients that are likely to subscribe to a term deposit. We will see in the confusion matrix four terms the True Positives, False Positives, True Negatives and False Negatives.

```
# Cross validate Log Reg Classifier
from sklearn.model_selection import cross_val_predict
```

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(log_reg, x_train, y_train, cv=kfold)
from sklearn.metrics import accuracy_score
log_reg.fit(x_train, y_train)
print ("Logistic Regression Classifier accuracy is %2.2f" % accuracy_score(y_train, y_train_pred))
```

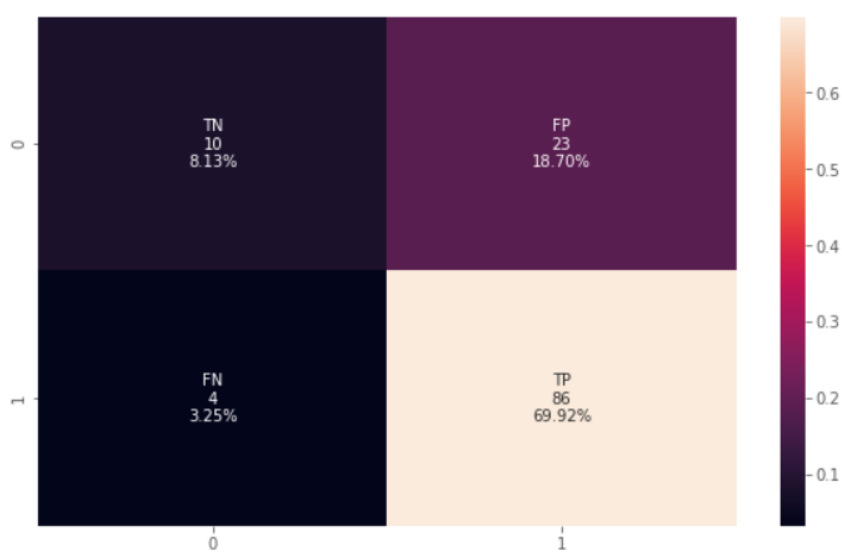
```
Logistic Regression Classifier accuracy is 0.81
```

```
# test data
y_test_pred = cross_val_predict(log_reg, x_test, y_test, cv=kfold)
from sklearn.metrics import accuracy_score
print ("Logistic Regression Classifier accuracy is %2.2f" % accuracy_score(y_test, y_test_pred))
```

```
Logistic Regression Classifier accuracy is 0.78
```

```
cm = confusion_matrix(y_test, y_test_pred)
print('Confusion Matrix :')
print(cm)

print ('Report : ')
print (classification_report(y_test, y_test_pred))

names = ['TN','FP','FN','TP']
counts = ['{0:0.0f}'.format(value) for value in cm.flatten()]
percentages = ['{0:.2%}'.format(value) for value in
cm.flatten()/np.sum(cm)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names, counts,
percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cm/np.sum(cm), annot=labels, fmt='')
plt.show()
```

```
Confusion Matrix :
[[10 23]
 [ 4 86]]

Accuracy: 78.05%

Report :
              precision    recall  f1-score   support

           0       0.71      0.30      0.43        33
           1       0.79      0.96      0.86        90

    accuracy                           0.78       123
   macro avg       0.75      0.63      0.64       123
weighted avg       0.77      0.78      0.75       123
```



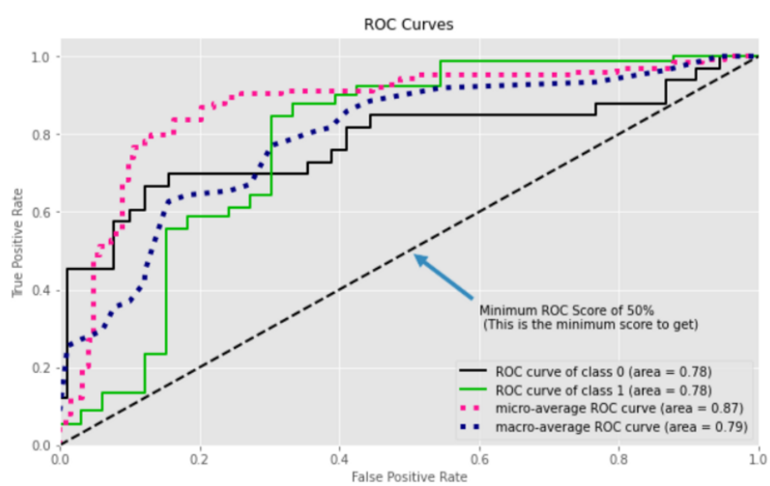
Recall Precision Tradeoff:

- As the precision gets higher the recall gets lower and vice versa. For instance, if we increase the precision from 30% to 60% the model is picking the predictions that the model believes is 60% sure.
- If there is an instance where the model believes that is 58% likely to be a loan approval then the model will classify it as a "No."
- However, that instance was actually a "Yes" (potential client eligible for loan)
- That is why the higher the precision the more likely the model is to miss instances that are actually a "Yes".

ROC Curve (Receiver Operating Characteristic):

- The ROC curve tells us how well our classifier is classifying between yes and no.
- The X-axis is represented by False positive rates (Specificity) and the Y-axis is represented by the True Positive Rate (Sensitivity.)
- As the line moves the threshold of the classification changes giving us different values.
- The closer is the line to our top left corner the better is our model separating both classes.

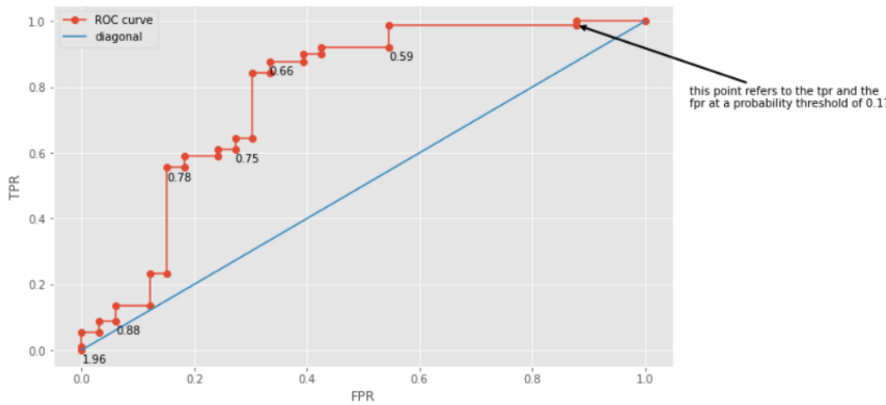
```
# predict probabilities
yhat = log_reg.predict_proba(x_test)
skplt.metrics.plot_roc_curve(y_test, yhat)
plt.annotate('Minimum ROC Score of 50% \n (This is the minimum score
to get)', xy=(0.5, 0.5), xytext=(0.6, 0.3),
arrowprops=dict(shrink=0.05))
plt.show()
```



Threshold for test from ROC-curve:

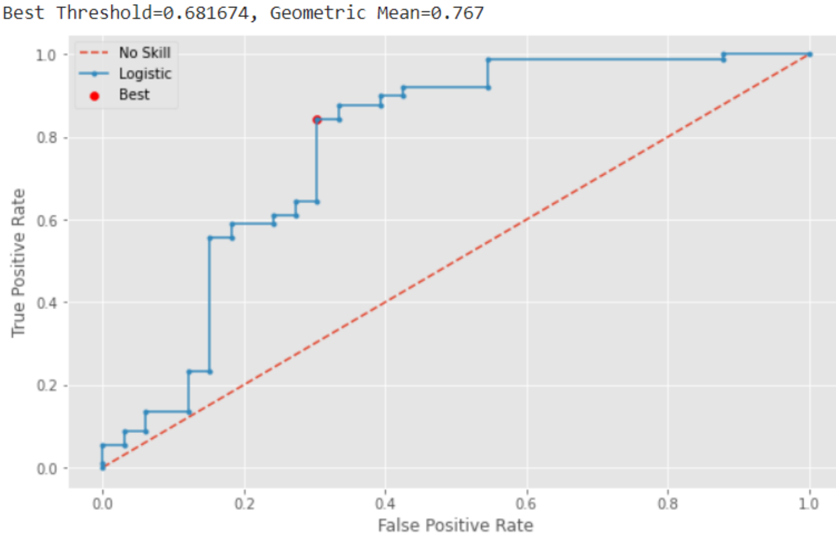
```
# retrieve just the probabilities for the positive class
probs = yhat[:, 1]
fpr, tpr, thresholds = metrics.roc_curve(y_test, probs)

plt.subplots(figsize=(10, 6))
plt.plot(fpr, tpr, 'o-', label="ROC curve")
plt.plot(np.linspace(0,1,10), np.linspace(0,1,10), label="diagonal")
for x, y, txt in zip(fpr[::5], tpr[::5], thresholds[::5]):
  plt.annotate(np.round(txt,2), (x, y-0.04))
```

```
rnd_idx = 27
plt.annotate('this point refers to the tpr and the \nfpr at a
probability threshold of {}'.format(np.round(thresholds[rnd_idx],
2)), xy=(fpr[rnd_idx], tpr[rnd_idx]), xytext=(fpr[rnd_idx]+0.2,
tpr[rnd_idx]-0.25), arrowprops=dict(color='black', lw=2,
arrowstyle='->'))
plt.legend(loc="upper left")
plt.xlabel("FPR"); plt.ylabel("TPR"); plt.show();
```




```
gmeans = sqrt(tpr * (1-fpr))# geometric means=sqrt(sensitivity *
specificity)= sqrt(tpr * (1-fpr))
index = argmax(gmeans) # index of the largest g-mean
print('Best Threshold=%f, G-Mean=%.3f' % (thresholds[ix],
gmeans[ix]))

# plot the roc curve for the model
plt.plot([0,1], [0,1], linestyle='--', label='No Skill')
plt.plot(fpr, tpr, marker='.', label='Logistic')
plt.scatter(fpr[index], tpr[index], marker='o', color='red',
label='Best')

plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate');
plt.legend(); plt.show();
```

```
Best Threshold=0.681674, Geometric Mean=0.767
```



Learning curve (Bias-Variance):

It is in-fact impossible to avoid the relationship between bias and variance for the sheer fact that-

- Increasing the bias will decrease the variance.

- Increasing the variance will decrease the bias.

So, we see there is a trade-off and to strike a balance we need to carefully select the algorithms and configure to work on reducible errors. Our target is to achieve low variance and low bias. Well, the detection is not that difficult, however, real work is to reduce the error to minimum. Several measures can be taken to reduce this error e.g.-

- increase the complexity of the model
- increase input features
- decrease regularization term etc.

```
skplt.estimators.plot_learning_curve(
 log_reg, x_train, y_train, title="Classifier Learning Curve",
 scoring="f1", cv=kfold, shuffle=True, random_state=42, n_jobs=-1,  figsize=(10, 5))
plt.show()
```

Learning curve plotted the prediction accuracy/error vs. the training set size to reflect how better does the model get at predicting the target as training instances are increased. Here both the training and test/validation performance are plotted together so we can diagnose the bias-variance tradeoff. This is contrast to ROC curve which does not show learning. Here y-axis is ‘score’, and higher the score, the better the performance of the model. Training score (red line) and Cross-validation score (green line) almost meeting at some point indicating model a good fit for the given data set.

Report generation

```
report = pd.read_csv("report_loan_approval.csv")
# fill the Loan_ID and Loan_Status
report['Loan_Status'] = y_test_pred
report['Loan_ID'] = df1['Loan_ID']

# replace with "N" and "Y"
report['Loan_Status'].replace(0, 'N', inplace=True)
report['Loan_Status'].replace(1, 'Y', inplace=True)

from IPython.display import HTML
def hover(hover_color="#ffff99"):
  return dict(selector="tr:hover", props=[("background-color", "%s"
% hover_color)])
  styles = [hover(), dict(selector="th", props=[("font-size",
"150%"), ("text-align", "center")]), dict(selector="caption", props=
[("caption-side", "bottom")])]
html = (report.style.set_table_styles(styles).set_caption("Hover to
highlight."))
html
```

	Loan_ID	Loan_Status
0	LP001015	1
1	LP001022	1
2	LP001031	1
3	LP001035	1
4	LP001051	1
5	LP001054	1
6	LP001055	1
7	LP001056	0
8	LP001059	1
9	LP001067	1
10	LP001078	1

Conclusion

Here, several machine learning algorithms were tried e.g. Logistic Regression(LR), Decision Tree (DT) and Random Forest (RF) etc. are applied to predict the loan approval of customers. The experimental results conclude that the accuracy of Logistic Regression algorithm is better as compared others. However, great care is required when selecting the appropriate cross-validation technique.

Connect me here.

Note: The programs described here are experimental and should be used with caution for any commercial purpose. All such use at your own risk.

Classification Algorithms Bias Variance Tradeoff Machine Learning Ai