In classification problems, we use two types of algorithms (dependent on the kind of output it creates):

1. **Class output**: Algorithms like SVM and KNN create a class output. For instance, in a binary classification problem, the outputs will be either 0 or 1. However, today we have algorithms that can convert these class outputs to probability.
2. **Probability output**: Algorithms like Logistic Regression, Random Forest, Gradient Boosting, Adaboost, etc. give probability outputs. Converting probability outputs to class output is just a matter of creating a threshold probability.

Introduction

By using different metrics for performance evaluation, we should be in a position to improve the overall predictive power of our model before we roll it out for production on unseen data.
Without doing a proper evaluation of the ML model using different metrics, and depending only on accuracy, it can lead to a problem when the respective model is deployed on unseen data and can result in poor predictions.

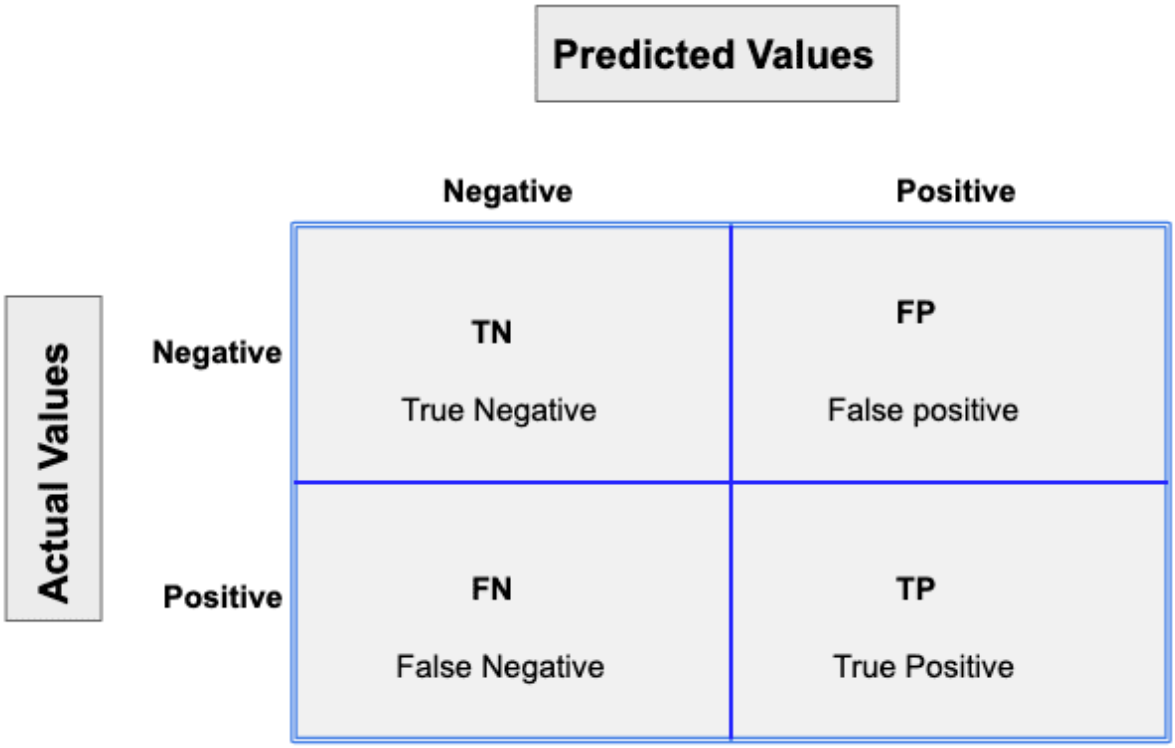
This happens because, in cases like these, our models don't learn but instead memorize; hence, they cannot generalize well on unseen data.

Model Evaluation Metrics

It aims to estimate the generalization accuracy of a model on the future (unseen/out-of-sample) data.

Confusion Matrix

A confusion matrix is a matrix representation of the prediction results of any binary testing that is often used to **describe the performance of the classification model** (or "classifier") on a set of test data for which the true values are known.
The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.



Confusion matrix with 2 class labels.

Each prediction can be one of the four outcomes, based on how it matches up to the actual value:

- **True Positive (TP):** Predicted True and True in reality.
- **True Negative (TN):** Predicted False and False in reality.
- **False Positive (FP):** Predicted True and False in reality.
- **False Negative (FN):** Predicted False and True in reality.

Now let us understand this concept using hypothesis testing.

A **Hypothesis** is speculation or theory based on insufficient evidence that lends itself to further testing and experimentation. With further testing, a hypothesis can usually be proven true or false.
A **Null Hypothesis** is a hypothesis that says there is no statistical significance between the two variables in the hypothesis. It is the hypothesis that the researcher is trying to disprove.
We would always reject the null hypothesis when it is false, and we would accept the null hypothesis when it is indeed true.

Even though hypothesis tests are meant to be reliable, **there are two types of errors that can occur.**
These errors are known as **Type 1 and Type II errors.**
For example, when examining the effectiveness of a drug, the null hypothesis would be that the drug does not affect a disease.

Type I Error:- equivalent to False Positives(FP).
The first kind of error that is possible involves the rejection of a null hypothesis that is true.

Let's go back to the example of a drug being used to treat a disease. If we reject the null hypothesis in this situation, then we claim that the drug does have some effect on a disease. But if the null hypothesis is true, then, in reality, the drug does not combat the disease at all. The drug is falsely claimed to have a positive effect on a disease.

Type II Error:- equivalent to False Negatives(FN).
The other kind of error that occurs when we accept a false null hypothesis. This sort of error is called a type II error and is also referred to as an error of the second kind.

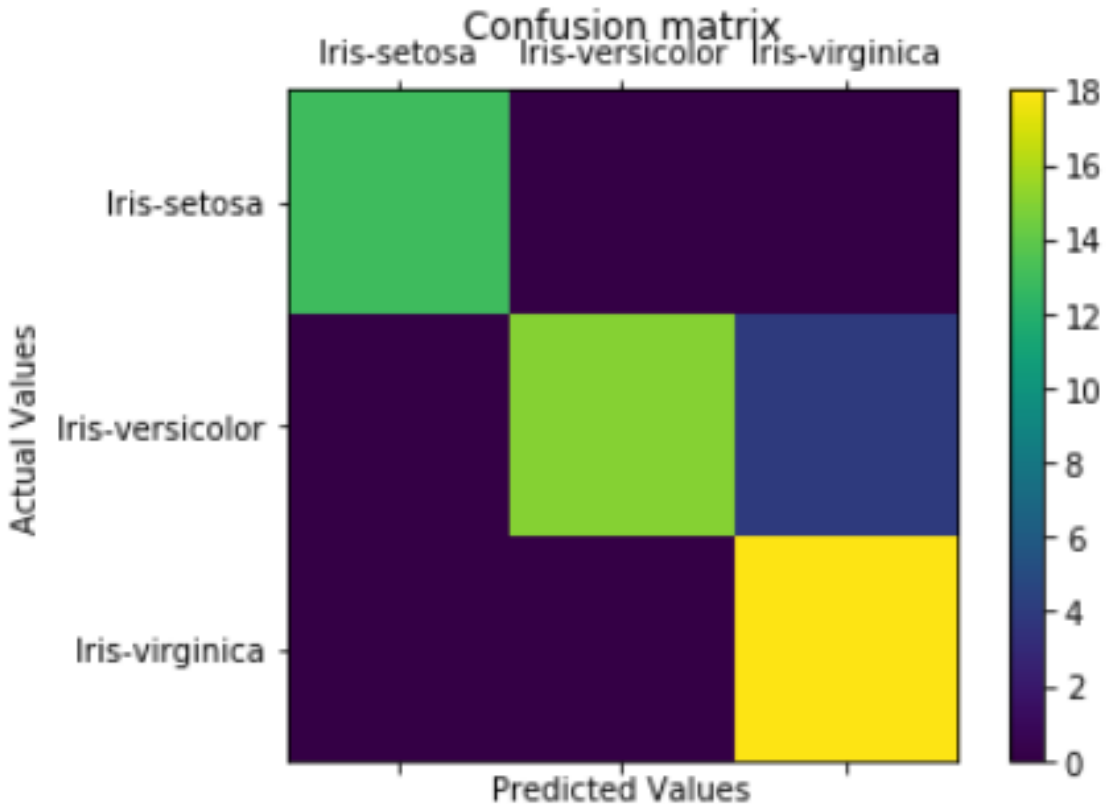
If we think back again to the scenario in which we are testing a drug, what would a type II error look like? A type II error would occur if we accepted that the drug hs no effect on disease, but in reality, it did.

A sample python implementation of the Confusion matrix.

```
import warnings
import pandas as pd
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline

#ignore warnings
warnings.filterwarnings('ignore')
# Load digits dataset
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url)
# df = df.values
X = df.iloc[:,0:4]
y = df.iloc[:,4]
#test size
test_size = 0.33
```

```
#generate the same set of random numbers
seed = 7
#Split data into train and test set.
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
test_size=test_size, random_state=seed)
#Train Model
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)

#Construct the Confusion Matrix
labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
cm = confusion_matrix(y_test, pred, labels)
print(cm)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

```
[[13  0  0]
 [ 0 15  4]
 [ 0  0 18]]
```



Confusion matrix with 3 class labels.

The diagonal elements represent the number of points for which the predicted label is equal to the true label, while anything off the diagonal was mislabeled by the classifier. Therefore, the higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

In our case, the classifier predicted all the 13 setosa and 18 virginica plants in the test data perfectly. However, it incorrectly classified 4 of the versicolor plants as virginica.

There is also a list of rates that are often computed from a confusion matrix for a binary classifier:

1. Accuracy

Overall, how often is the classifier correct?
Accuracy = (TP+TN)/total

When our classes are roughly equal in size, we can use accuracy**,** which will give us correctly classified values.
Accuracy is a common evaluation metric for classification problems. It's the number of correct predictions made as a ratio of all predictions made.

Misclassification Rate(Error Rate): Overall, how often is it wrong.
Since accuracy is the percent we correctly classified (success rate), it follows that our error rate (the percentage we got wrong) can be calculated as follows:
Misclassification Rate = (FP+FN)/total

We use the sklearn module to compute the accuracy of a classification task, as shown below.

```
#import modules
import warnings
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import accuracy_score
```

```
#ignore warnings
warnings.filterwarnings('ignore')
# Load digits dataset
iris = datasets.load_iris()
# # Create feature matrix
X = iris.data
# Create target vector
y = iris.target
#test size
test_size = 0.33
#generate the same set of random numbers
seed = 7
#cross-validation settings
kfold = model_selection.KFold(n_splits=10, random_state=seed)
#Model instance
model = LogisticRegression()
#Evaluate model performance
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, y, cv=kfold, scoring=scoring)
print('Accuracy -val set: %.2f%% (%.2f)' % (results.mean()*100, results.std()))

#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
test_size=test_size, random_state=seed)
#fit model
model.fit(X_train, y_train)
#accuracy on test set
result = model.score(X_test, y_test)
print("Accuracy - test set: %.2f%%" % (result*100.0))
```

The classification accuracy is **88%** on the validation set.

2. Precision

When it predicts yes, how often is it correct?

Precision=tp / (tp + fp)

When we have a class imbalance, accuracy can become an unreliable metric for measuring our performance. For instance, if we had a 99/1 split between two classes, A and B, where the rare event, B, is our positive class, we could build a model that was 99% accurate by just saying everything belonged to class A. Clearly, we shouldn't bother building a model if it doesn't do anything to identify class B; thus, we need different metrics that will discourage this behavior. For this, we use precision and recall instead of accuracy.

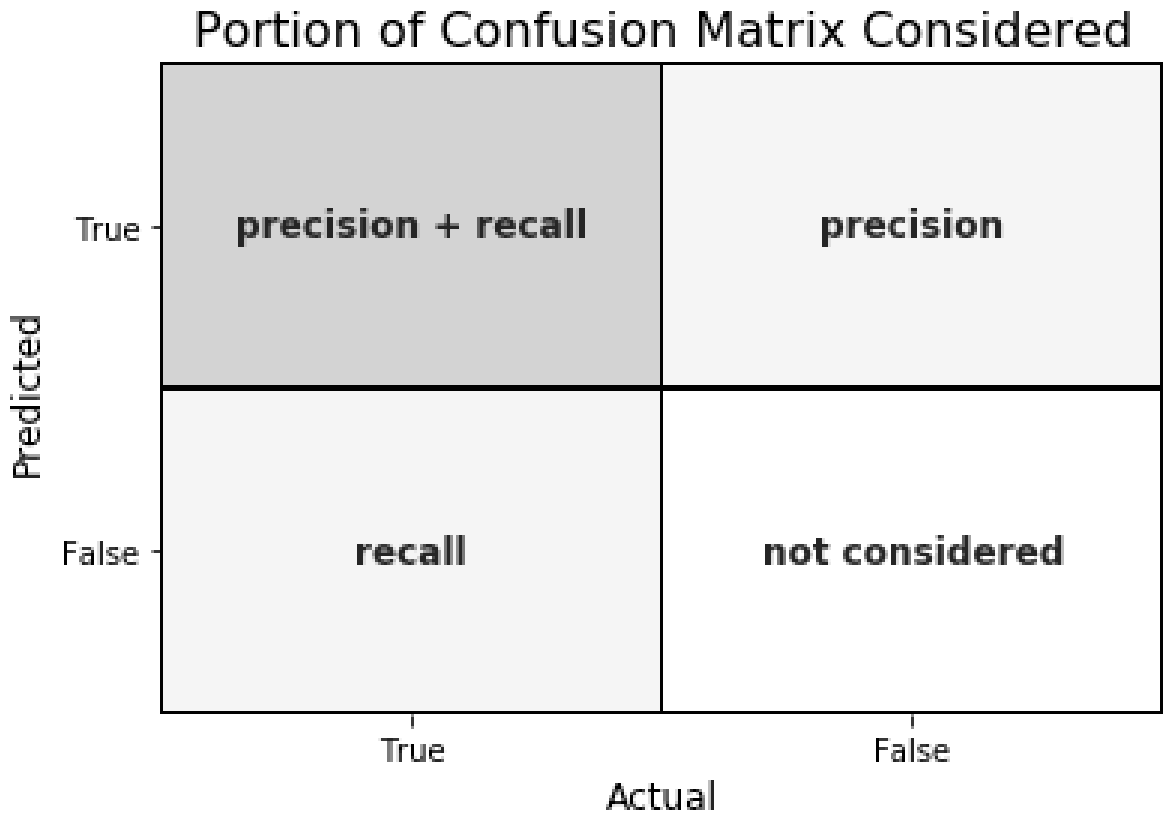
3. Recall or Sensitivity

When it's actually yes, how often does it predict yes?
True Positive Rate = tp / (tp + fn)

Recall gives us the **true positive rate** (**TPR**), which is the ratio of true positives to everything positive.

In the case of the 99/1 split between classes A and B, the model that classifies everything as A would have a recall of 0% for the positive class, B (precision would be undefined — 0/0). Precision and recall provide a better way of evaluating model performance in the face of a class imbalance. They will correctly tell us that the model has little value for our use case.

Just like accuracy, both precision and recall are easy to compute and understand but require thresholds. Besides, precision and recall only consider half of the confusion matrix:



4. F1 Score

The F1 score is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Why harmonic mean? Since the harmonic mean of a list of numbers skews strongly toward the least elements of the list, it tends (compared to the arithmetic mean) to mitigate the impact of large outliers and aggravate the impact of small ones.

An F1 score punishes extreme values more. Ideally, an F1 Score could be an effective evaluation metric in the following classification scenarios:

- *When FP and FN are equally costly — meaning they miss on true positives or find false positives — both impact the model almost the same way, as in our cancer detection classification example*
- *Adding more data doesn't effectively change the outcome effectively*
- *TN is high (like with flood predictions, cancer predictions, etc.)*

A sample python implementation of the F1 score.

```
import warnings
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
from sklearn.metrics import precision_recall_fscore_support as score, precision_score,
recall_score, f1_score

warnings.filterwarnings('ignore')

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-
diabetes.data.csv"
dataframe = pandas.read_csv(url)
dat = dataframe.values
X = dat[:,:-1]
y = dat[:,-1]
test_size = 0.33
seed = 7

model = LogisticRegression()
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
test_size=test_size, random_state=seed)
model.fit(X_train, y_train)
precision = precision_score(y_test, pred)
print('Precision: %f' % precision)
```

```
# recall: tp / (tp + fn)
recall = recall_score(y_test, pred)
print('Recall: %f' % recall)
# f1: tp / (tp + fp + fn)
f1 = f1_score(y_test, pred)
print('F1 score: %f' % f1)
```

```
Precision: 0.696970
Recall: 0.541176
F1 score: 0.609272
```

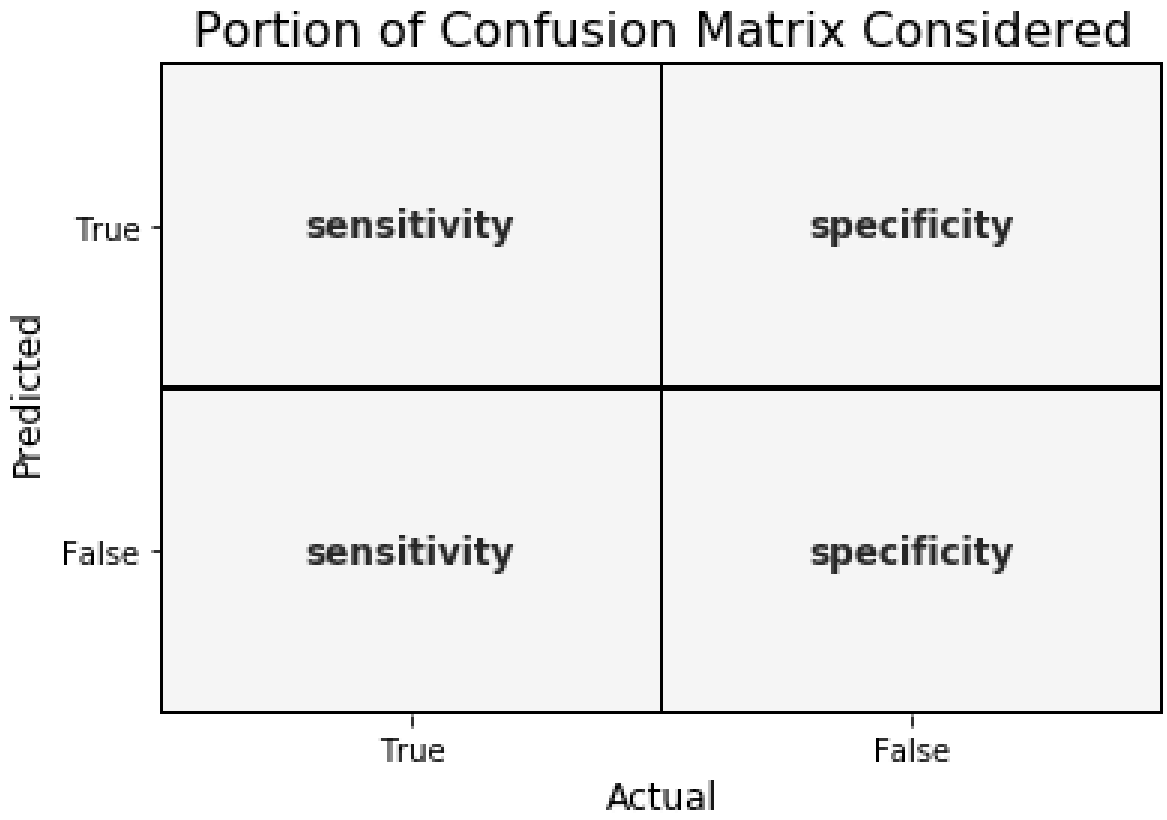
5. Specificity

When it's no, how often does it predict no?

True Negative Rate=TN/actual no

It is the **true negative rate** or the proportion of true negatives to everything that should have been classified as negative.

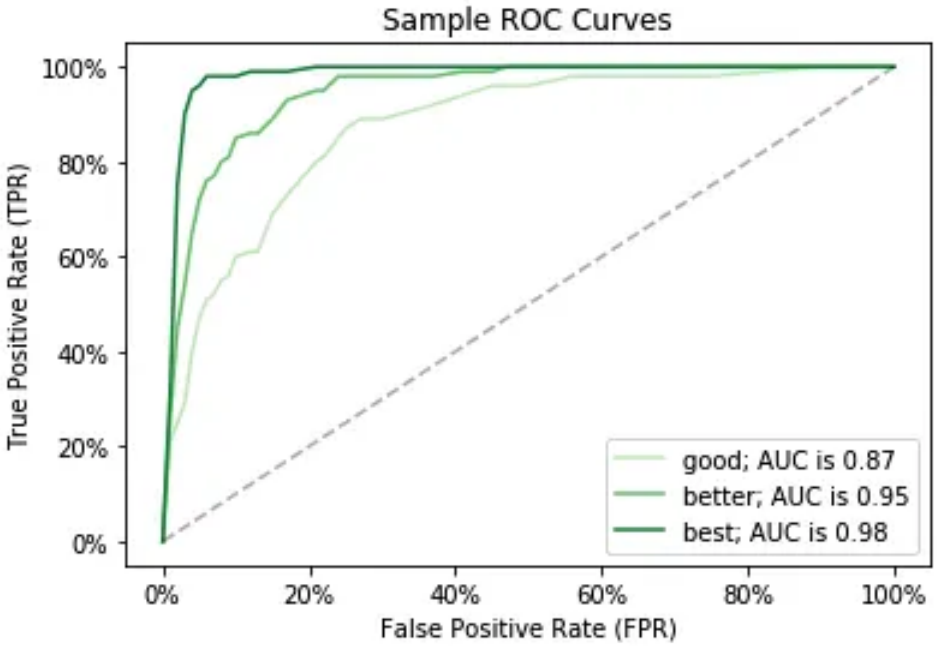
Note that, together, specificity and sensitivity consider the full confusion matrix:



6. Receiver Operating Characteristics (ROC) Curve

Measuring the area under the ROC curve is also a very useful method for evaluating a model. By plotting the true positive rate (sensitivity) versus the false-positive rate (1 — specificity), we get the **Receiver Operating Characteristic** (**ROC**) **curve**. This curve allows us to visualize the trade-off between the true positive rate and the false positive rate.

The following are examples of good ROC curves. The dashed line would be random guessing (no predictive value) and is used as a baseline; anything below that is considered worse than guessing. We want to be toward the top-left corner:

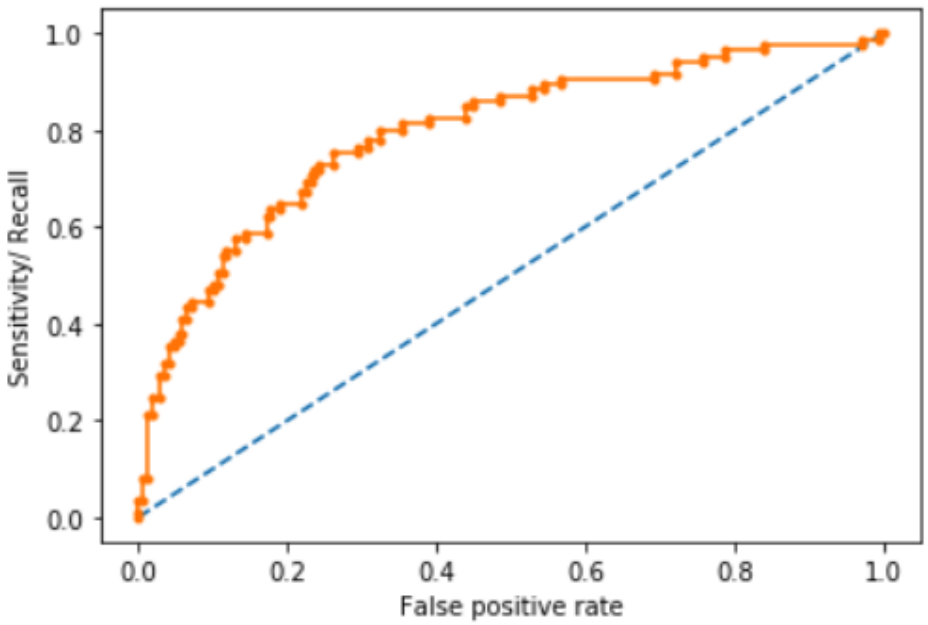


A sample python implementation of the ROC curves.

```
#Classification Area under curve
import warnings
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve

warnings.filterwarnings('ignore')

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
dataframe = pandas.read_csv(url)
dat = dataframe.values
X = dat[:,:-1]
y = dat[:,-1]
seed = 7
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=test_size, random_state=seed)
model.fit(X_train, y_train)

# predict probabilities
```

```
probs = model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

auc = roc_auc_score(y_test, probs)
print('AUC - Test Set: %.2f%%' % (auc*100))

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
plt.xlabel('False positive rate')
plt.ylabel('Sensitivity/ Recall')
# show the plot
plt.show()
```

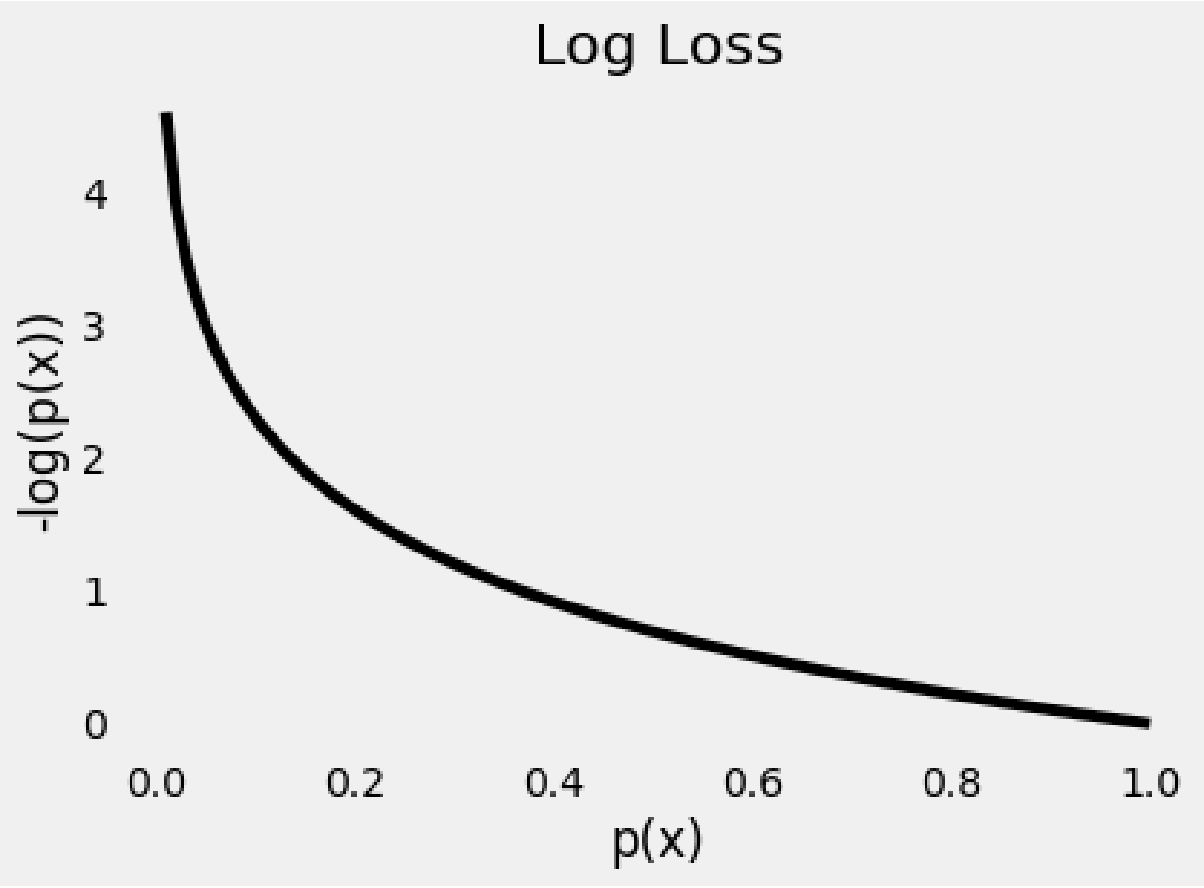
```
AUC - Test Set: 79.65%
```



In the example above, the AUC is relatively close to 1 and greater than 0.5. A perfect classifier will have the ROC curve go along the Y-axis and then along the X-axis.

Log Loss

Log Loss is the most important classification metric based on probabilities.



As the **predicted probability** of the **true class** gets **closer to zero**, the **loss increases exponentially**

It measures the performance of a classification model where the prediction input is a probability value between 0 and 1. Log loss increases as the predicted probability diverge from the actual label. The goal of any machine learning model is to minimize this value. As such, smaller log loss is better, with a perfect model having a log loss of 0.

A sample python implementation of the Log Loss.

```
#Classification LogLoss
import warnings
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

warnings.filterwarnings('ignore')
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
dataframe = pandas.read_csv(url)
dat = dataframe.values
X = dat[:,:-1]
y = dat[:,-1]
seed = 7
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=test_size, random_state=seed)
model.fit(X_train, y_train)
#predict and compute logloss
pred = model.predict(X_test)
accuracy = log_loss(y_test, pred)
print("Logloss: %.2f" % (accuracy))
```

view rawlogg_loss.py hosted with ♥ by GitHub

```
Logloss: 8.02
```