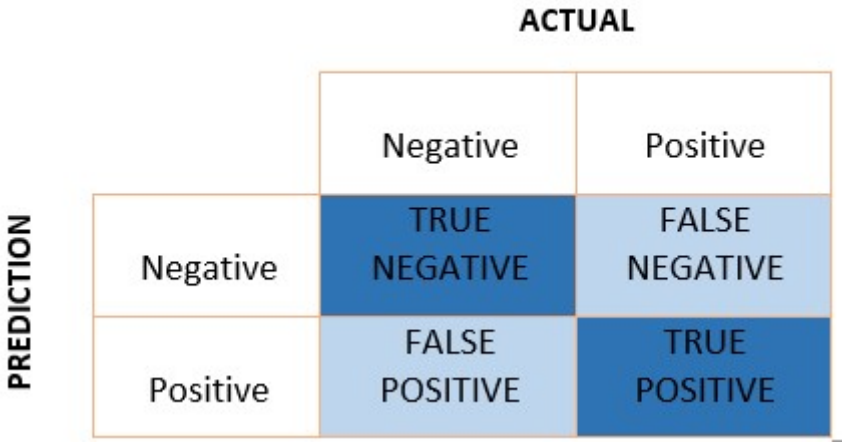
Performance Metrics: Confusion matrix, Precision, Recall, and F1 Score

Accuracy performance metrics can be decisive when dealing with imbalanced data. In this tutorial, we will learn about the Confusion matrix and its associated terms, which looks confusing but are trivial. The confusion matrix, precision, recall, and F1 score gives better intuition of prediction results as compared to accuracy. To understand the concepts, we will limit this article to binary classification only.

What is a confusion matrix?

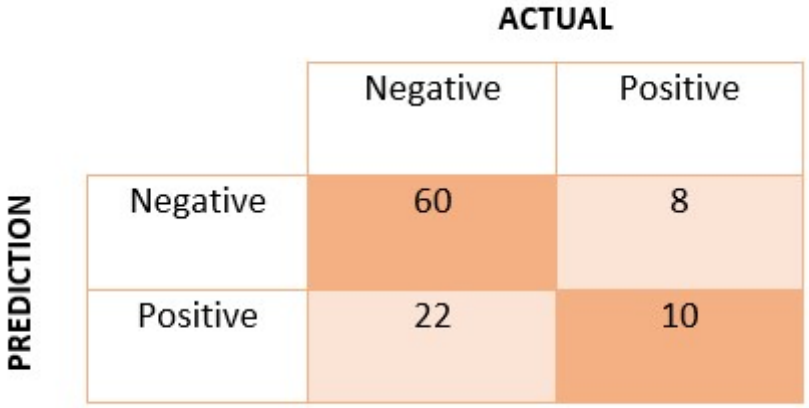
It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another.



Confusion Matrix

EXAMPLE

A machine learning model is trained to predict tumor in patients. The test dataset consists of 100 people.



Confusion Matrix for tumor detection

True Positive (TP) — model correctly predicts the positive class (prediction and actual both are positive). In the above example, **10 people** who have tumors are predicted positively by the model.

True Negative (TN) — model correctly predicts the negative class (prediction and actual both are negative). In the above example, **60 people** who don't have tumors are predicted negatively by the model.

False Positive (FP) — model gives the wrong prediction of the negative class (predicted-positive, actual-negative). In the above example, **22 people** are predicted as positive of having a tumor, although they don't have a tumor. FP is also called a **TYPE I** error.

False Negative (FN) — model wrongly predicts the positive class (predicted-negative, actual-positive). In the above example, **8 people** who have tumors are predicted as negative. FN is also called a **TYPE II** error.

With the help of these four values, we can calculate **True Positive Rate (TPR)**, **False Negative Rate (FPR)**, **True Negative Rate (TNR)**, and **False Negative Rate (FNR)**.

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

$$FNR = \frac{FN}{Actual\ Positive} = \frac{FN}{TP + FN}$$

$$TNR = \frac{TN}{Actual\ Negative} = \frac{TN}{TN + FP}$$

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

Even if data is imbalanced, we can figure out that our model is working well or not. For that, **the values of TPR and TNR should be high, and FPR and FNR should be as low as possible.**

With the help of TP, TN, FN, and FP, other performance metrics can be calculated.

Precision, Recall

Both precision and recall are crucial for information retrieval, where positive class mattered the most as compared to negative. **Why?**

While searching something on the web, the model does not care about something **irrelevant** and **not retrieved** (this is the true negative case). Therefore only TP, FP, FN are used in Precision and Recall.

Precision

Out of all the positive predicted, what percentage is truly positive.

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

The precision value lies between 0 and 1.

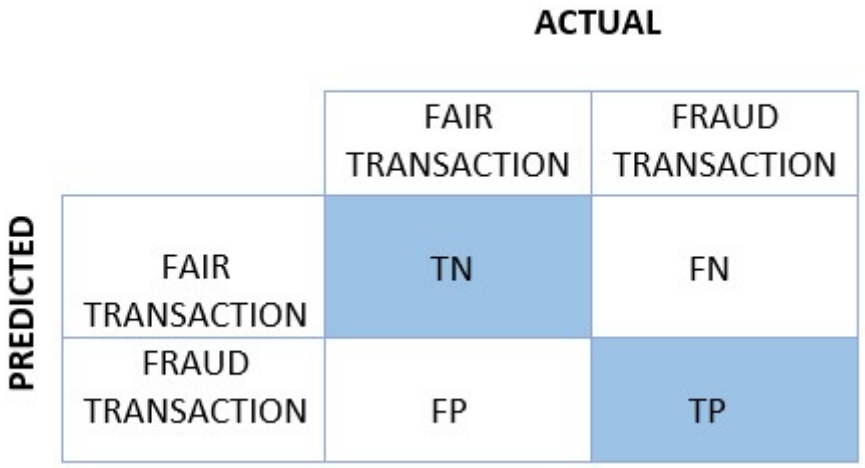
Recall

Out of the total positive, what percentage are predicted positive. It is the same as TPR (true positive rate).

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

How are precision and recall useful? Let's see through examples.

EXAMPLE 1- Credit card fraud detection



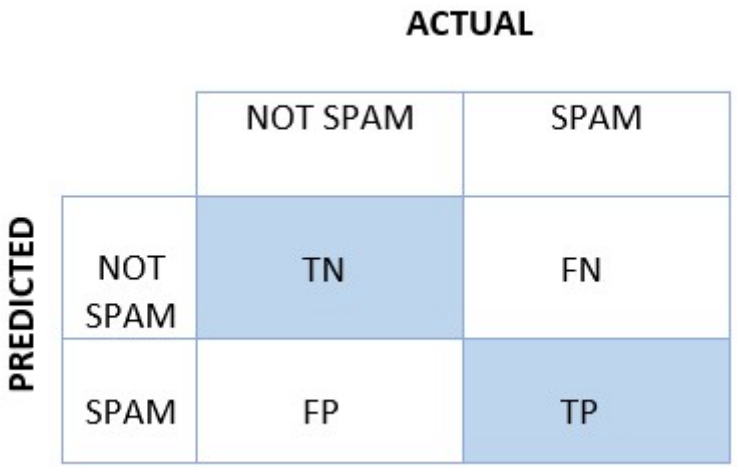
Confusion Matrix for Credit Card Fraud Detection

Let's understand the confusing terms in the confusion matrix: **true positive, true negative, false negative, and false positive** with an example.

We do not want to **miss any fraud transactions**. Therefore, we **want False-Negative to be as low as possible.** In these situations, we can compromise with the low precision, but recall should be high. Similarly, in the medical application, we don't want to miss any patient. Therefore we focus on having a high recall.

So far, we have discussed when the recall is important than precision. **But, when is the precision more important than recall?**

EXAMPLE 2 — Spam detection



Confusion Matrix for Spam detection

In the detection of spam mail, it is okay if any spam mail remains undetected (false negative), but what if we miss any critical mail because it is classified as spam (false positive). In this situation, **False Positive** should be as low as possible. Here, precision is more vital as compared to recall.

When comparing different models, it will be difficult to decide which is better (high precision and low recall or vice-versa). Therefore, there should be a metric that combines both of these. One such metric is the F1 score.

F1 Score

It is the harmonic mean of precision and recall. It takes both false positive and false negatives into account. Therefore, it performs well on an imbalanced dataset.

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

F1 score gives the same weightage to recall and precision.

There is a **weighted F1 score** in which we can give different weightage to recall and precision. As discussed in the previous section, different problems give different weightage to recall and precision.

$$F_\beta = (1 + \beta^2) * \frac{(Precision * Recall)}{(\beta^2 * Precision) + Recall}$$

Beta represents how many times recall is more important than precision. If the recall is twice as important as precision, the value of Beta is 2.

Confusion matrix, precision, recall, and F1 score provides better insights into the prediction as compared to accuracy performance metrics. Applications of precision, recall, and F1 score is in information retrieval, word segmentation, named entity recognition, and many more.

Specificity, Sensitivity, Accuracy, Precision, etc.

Here we go with an example:

A school is running a machine learning primary diabetes scan on all of its students.

The output is either **diabetic (+ve)** or **healthy (-ve).**

There are only 4 cases any student *X* could end up with.
We'll be using the following as a reference later, So don't hesitate to re-read it if you get confused.

- **True positive (*TP*): Prediction is +ve and X is diabetic, *we want that***
- **True negative (*TN*): Prediction is -ve and X is healthy, *we want that too***
- **False positive (*FP*): Prediction is +ve and X is healthy, *false alarm, bad***
- **False negative (*FN*): Prediction is -ve and X is diabetic, *the worst***

To remember that, there are 2 tricks
- If it **starts with True** then the prediction was correct whether diabetic or not, so true positive is a diabetic person correctly predicted & a true negative is a healthy person correctly predicted.

Oppositely, if it **starts with False** then the prediction was incorrect, so false positive is a healthy person incorrectly predicted as diabetic(+) & a false negative is a diabetic person incorrectly predicted as healthy(-).

- **Positive or negative indicates** the output of our program. While true or false judges this output whether correct or incorrect.

True positives & true negatives are always good.
We love the news the word true brings.

Which leaves **false positives** and **false negatives**.

In our example, **false positives are just a false alarm**. May be, in a second more detailed scan it'll be corrected.
But a **false negative** label, this means that they think they're healthy when they're not, which is — in our problem — the worst case of the four.

Whether ***FP*** & ***FN*** are equally bad or if one of them is worse than the other depends on your problem. This piece of information has a great impact on your choice of the performance metric, So give it a thought before you continue.

Which performance metric to choose?

Accuracy--
It's the ratio of the correctly labeled subjects to the whole pool of subjects.
Accuracy is the most intuitive one.
Accuracy answers the following question: How many students did we correctly label out of all the students?
Accuracy = (TP+TN)/(TP+FP+FN+TN)
numerator: all correctly labeled subject (All trues)
denominator: all subjects

Precision--
Precision is the ratio of the correctly +ve labeled by our program to all +ve labeled.
Precision answers the following: How many of those who we labeled as diabetic are actually diabetic?
Precision = TP/(TP+FP)
numerator: +ve labeled diabetic people.
denominator: all +ve labeled by our program (whether they're diabetic or not in reality).

Recall (aka Sensitivity)--
Recall is the ratio of the correctly +ve labeled by our program to all who are diabetic in reality.
Recall answers the following question: Of all the people who are diabetic, how many of

those we correctly predict?
Recall = TP/(TP+FN)
numerator: +ve labeled diabetic people.
denominator: all people who are diabetic (whether detected by our program or not)

F1-score (aka F-Score / F-Measure)--
F1 Score considers both precision and recall.
It is the harmonic mean(average) of the precision and recall.
F1 Score is best if there is some sort of balance between precision (p) & recall (r) in the system.
Oppositely F1 Score isn't so high if one measure is improved at the expense of the other.
For example, *if P is 1 & R is 0, F1 score is 0.*
F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Specificity--
Specificity is the correctly -ve labeled by the program to all who are healthy in reality.
Specifity answers the following question: Of all the people who are healthy, how many of those did we correctly predict?
Specificity = TN/(TN+FP)
numerator: -ve labeled healthy people.
denominator: all people who are healthy in reality (whether +ve or -ve labeled)

General Notes

Yes, **accuracy is a great measure but only when you have symmetric datasets (false negatives & false positives counts are close)**, also, **false negatives & false positives have similar costs**.

If the cost of false positives and false negatives are different then F1 is your savior. **F1 is best if you have an uneven class distribution**.

Precision is how sure you are of your true positives whilst recall is how sure you are that you are not missing any positives.

Choose **Recall if the idea of false positives is far better than false negatives, in other words, if the occurrence of false negatives is unaccepted/intolerable**, that you'd rather get some extra false positives(false alarms) over saving some false negatives, like in our diabetes example.
You'd rather get some healthy people labeled diabetic over leaving a diabetic person labeled healthy.

Choose **precision if you want to be more confident of your true positives**. for example, Spam emails. You'd rather have some spam emails in your inbox rather than some regular emails in your spam box. So, the email company wants to be extra sure that email *Y* is spam before they put it in the spam box and you never get to see it.

Choose **Specificity if you want to cover all true negatives, meaning you don't want any false alarms, you don't want any false positives**. for example, you're running a drug test in which all people who test positive will immediately go to jail, you don't want anyone drug-free going to jail. False positives here are intolerable.

Bottom Line is

— Accuracy value of 90% means that 1 of every 10 labels is incorrect, and 9 is correct.
— Precision value of 80% means that on average, 2 of every 10 diabetic labeled student by our program is healthy, and 8 is diabetic.
— Recall value is 70% means that 3 of every 10 diabetic people in reality are missed by our program and 7 labeled as

diabetic.
— Specificity value is 60% means that 4 of every 10 healthy people in reality are miss-labeled as diabetic and 6 are correctly labeled as healthy.

Confusion Matrix

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

Predicted

Actual

Predicted

Actual

	Positive	Negative	
Positive	True positive(TP)	False Negative(FN)	Sensitivity or Recall or True Positive Rate=TP/(TP+FN)
Negative	False Positive (FP)	True Negative(TN)	Specificity or True Negative Rate=TN/(TN+FP)
	Precision or Positive Predictive Value=TP/(TP+FP)	Negative Predictive Value=FN/(FN+TN)	Accuracy=TP+TN/TP+TN+FP+FN

Receiver Operating Characteristic (ROC) and Area Under the Curve (AUC)

What is ROC Curve?

ROC or Receiver Operating Characteristic plot is used to visualise the performance of a binary classifier. It gives us the trade-off between the **True Positive Rate (TPR)** and the **False Positive Rate (FPR)** at different classification thresholds.

True Positive Rate:

True Positive Rate is the proportion of observations that are correctly predicted to be positive.

$$\text{TPR} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

False Positive Rate:

False Positive Rate is the proportion of observations that are incorrectly predicted to be positive.

$$\text{FPR} = \frac{False\ Positive}{True\ Negative + False\ Positive}$$

For different threshold values we will get different TPR and FPR. So, in order to visualise which threshold is best suited for the classifier we plot the ROC curve. The following figure shows what a typical ROC curve look like.:

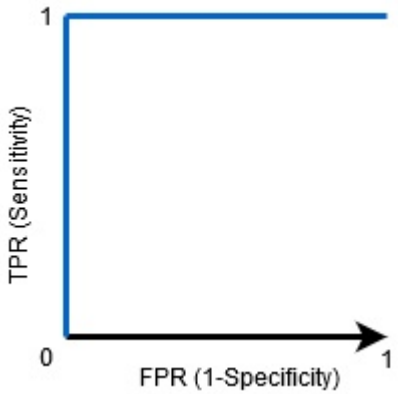
Receiver operating characteristic (ROC) curve can be used to understand the overall performance (worth) of a logistic regression model (and, in general, of classification models) and used for model selection. The term has its origin in electrical engineering when electrical signals were used for predicting enemy objects (such as submarines and aircraft) during World War II. Given a random pairof positive and negative class records, ROC gives the proportions of such pairs that will be correctlyclassified.

ROC curve is a plot between **sensitivity** (true positive rate) on the vertical axis and **1 – specificity**(false positive rate) on the horizontal axis.

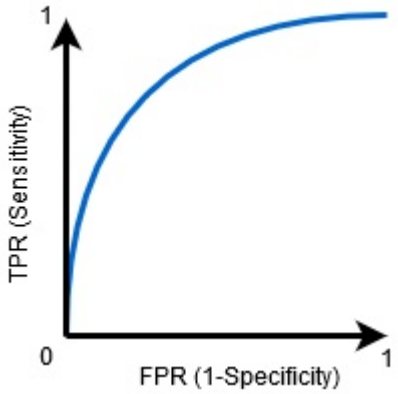
AUC (Area Under the Curve) is believed to be one of the best ways to summarize performance in a single number. AUC indicates that the probability of a randomly selected positive example will be scored higher by the classifier than a randomly selected negative example. If you have multiple models with nearly the same accuracy, you can pick the one that gives higher AUC.
Area Under the Curve or AUC ROC curve is nothing but the area under the curve calculated in the ROC space. One of the easy ways to calculate the AUC score is using the trapezoidal rule, which is adding up all trapezoids under the curve.

The **Area Under the Curve (AUC)** is the measure of the ability of a binary classifier to distinguish between classes and is used as a summary of the ROC curve.

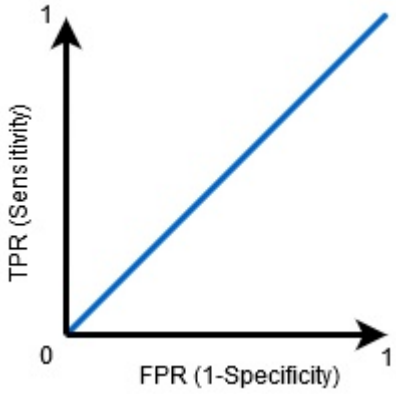
The higher the AUC, the better the model's performance at distinguishing between the positive and negative classes.



When AUC = 1, the classifier can correctly distinguish between all the Positive and the Negative class points. If, however, the AUC had been 0, then the classifier would predict all Negatives as Positives and all Positives as Negatives.



When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative ones. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.



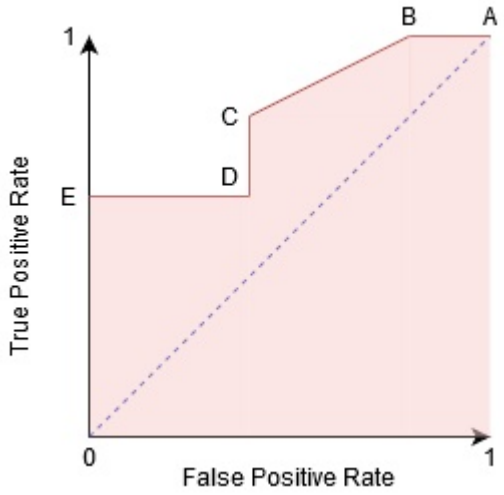
When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning that the classifier either predicts a random class or a constant class for all the data points.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

How Does the AUC-ROC Curve Work?

In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance False positives and False negatives.

Let’s dig a bit deeper and understand what our ROC curve would look like for different threshold values and how the specificity and sensitivity would vary.



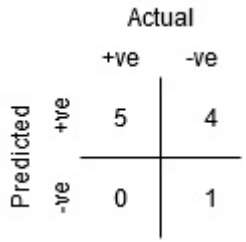
We can try and understand this graph by generating a confusion matrix for each point corresponding to a threshold and talk about the performance of our classifier:

		Actual	
		+ve	-ve
Predicted	+ve	5	5
	-ve	0	0

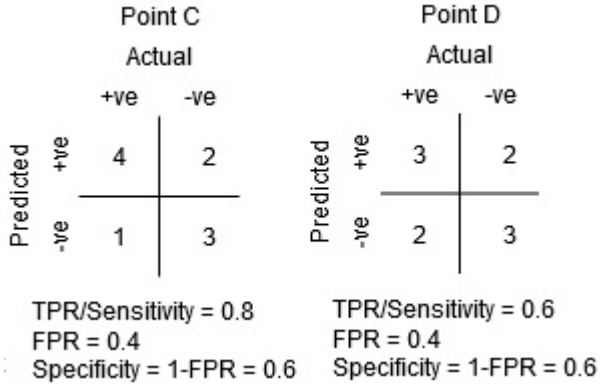
Point A is where the **Sensitivity** is the highest and **Specificity** the lowest. This means all the Positive class points are classified correctly, and all the Negative class points are classified incorrectly.

In fact, any point on the blue line corresponds to a situation where the True Positive Rate is equal to False Positive Rate.

All points above this line correspond to the situation where the proportion of correctly classified points belonging to the Positive class is greater than the proportion of incorrectly classified points belonging to the Negative class.



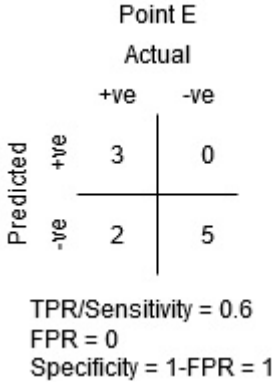
Although **Point B** has the **same Sensitivity as Point A**, it has a **higher Specificity**. Meaning the number of incorrectly Negative class points is lower than the previous threshold. This indicates that this threshold is better than the previous one.



Between **points C and D**, the Sensitivity at **point C is higher than point D for the same Specificity**. This means, for the same number of incorrectly classified Negative class points, the classifier predicted a higher number of Positive class points. Therefore, the threshold at point C is better than point D.

Now, depending on how many incorrectly classified points we want to tolerate for our classifier, we would choose between point B or C to predict whether you can defeat your opponent in PUBG or not.

“False hopes are more dangerous than fears.”–J.R.R. Tolkein



Point E is where the **Specificity becomes highest**. Meaning the model classifies no False Positives. The model can correctly classify all the Negative class points! We would choose this point if our problem was to give perfect song recommendations to our users.

Going by this logic, can you guess where the point corresponding to a perfect classifier would lie on the graph?

Yes! It would be on the top-left corner of the ROC graph corresponding to the coordinate (0, 1) in the cartesian plane. Here, both the Sensitivity and Specificity would be the highest, and the classifier would correctly classify all the Positive and Negative class points.

Code_1:

```
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1],
[1, 1, 1, 0]).ravel()
# true negatives, false positives, false negatives, true positives
(tn, fp, fn, tp)
```

Result--> (0, 2, 1, 1)

Code_2:

```
import pandas as pd
importnumpy as np
importmatplotlib.pyplot as plt
importscipy.stats as stats
importstatsmodels.api as sm

# Load data
df = pd.read_csv('F:/Practical__PYTHON/datasets/code__data__6steps/Chapter_3_Code/Code\Data/Grade_Set_1_Classification.csv')
print (df)

fromsklearn.linear_model import LogisticRegression
```

```
# manually add intercept
df['intercept'] = 1
independent_variables = ['Hours_Studied', 'intercept']
x = df[independent_variables]       # independent variable
y = df['Result']                    # dependent variable
# instantiate a logistic regression model, and fit with X and y
model = LogisticRegression(solver='lbfgs')
model = model.fit(x, y)

'"The “lbfgs” is an optimization algorithm that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm, which belongs to quasi-Newton methods. The “lbfgs” solver is recommended for use for small data-sets but for larger datasets its performance suffers.
The “lbfgs” solver is used by default for its robustness. For large datasets the “saga” solver is usually faster. For large dataset, you may also consider using SGDClassifier with ‘log’ loss, which might be even faster but requires more tuning.'"

# check the accuracy on the training set
model.score(x, y)

# predict_proba will return array containing probability of y = 0 and y = 1
print ('Predicted probability:', model.predict_proba(x)[:,1])

# predict will give convert the probability(y=1) values > .5 to 1 else 0
print ('Predicted Class:',model.predict(x))
```

```
# plotting fitted line
plt.scatter(df.Hours_Studied, y,  color='black')
plt.yticks([0.0, 0.5, 1.0])
plt.plot(df.Hours_Studied, model.predict_proba(x)[:,1], color='blue', linewidth=3)
plt.title('Hours Studied vs Result - Fitted Line')
plt.ylabel('Result')
plt.xlabel('Hours_Studied')
plt.show()
fromsklearn import metrics
# generate evaluation metrics
print ("Accuracy :", metrics.accuracy_score(y, model.predict(x)))
print ("AUC :", metrics.roc_auc_score(y, model.predict_proba(x)[:,1]))
print ("Confusion matrix :",metrics.confusion_matrix(y, model.predict(x)))
print ("classification report :", metrics.classification_report(y, model.predict(x)))
# Determine the false positive and true positive rates
fpr, tpr, _ = metrics.roc_curve(y, model.predict_proba(x)[:,1])
 # Calculate the AUC
roc_auc = metrics.auc(fpr, tpr)
print ('ROC AUC: %0.2f' % roc_auc)
 # Plot of a ROC curve for a specific class
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
# instantiate a logistic regression model with default c value, and fit with X and y
model = LogisticRegression(solver='lbfgs')
model = model.fit(x, y)
# check the accuracy on the training set
print ("C = 1 (default), Accuracy :", metrics.accuracy_score(y, model.predict(x)))
# instantiate a logistic regression model with c = 10, and fit with X and y
model1 = LogisticRegression(solver='lbfgs',C=10)
model1 = model1.fit(x, y)
# check the accuracy on the training set
print ("C = 10, Accuracy :", metrics.accuracy_score(y, model1.predict(x)))
# instantiate a logistic regression model with c = 100, and fit with X and y
model2 = LogisticRegression(solver='lbfgs',C=100)
model2 = model2.fit(x, y)
# check the accuracy on the training set
```

```
print ("C = 100, Accuracy :", metrics.accuracy_score(y, model2.predict(x)))

# instantiate a logistic regression model with c = 1000, and fit with X and y
model3 = LogisticRegression(solver='lbfgs',C=1000)
model3 = model3.fit(x, y)

# check the accuracy on the training set
print ("C = 1000, Accuracy :", metrics.accuracy_score(y, model3.predict(x)))

# plotting fitted line
plt.scatter(df.Hours_Studied, y,  color='black', label='Result')
plt.yticks([0.0, 0.5, 1.0])
plt.plot(df.Hours_Studied, model.predict_proba(x)[:,1], color='gray', linewidth=2, label='C=1.0')
plt.plot(df.Hours_Studied, model1.predict_proba(x)[:,1], color='blue', linewidth=2,label='C=10')
plt.plot(df.Hours_Studied, model2.predict_proba(x)[:,1], color='green', linewidth=2,label='C=100')
plt.plot(df.Hours_Studied, model3.predict_proba(x)[:,1], color='red', linewidth=2,label='C=1000')
plt.legend(loc='lower right') # legend location
plt.title('Hours Studied vs Result')
plt.ylabel('Result')
plt.xlabel('Hours_Studied')
plt.show()
```

References:

1. https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262
2. https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124
3. https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
4. https://intellipaat.com/blog/roc-curve-in-machine-learning/
5. https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
6. https://www.numpyninja.com/post/recall-specificity-precison-f1-scores-and-accuracy
7. https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/