MACHINE LEARNING

In Q1 to Q5, only one option is correct, Choose the correct option:

1. In which of the following you can say that the model is overfitting?
   A) High R-squared value for train-set and High R-squared value for test-set.
   B) Low R-squared value for train-set and High R-squared value for test-set.
   C) High R-squared value for train-set and Low R-squared value for test-set.
   D) None of the above

ANS : High R-squared value for train-set and High R-squared value for test-set.

2. Which among the following is a disadvantage of decision trees?
   A) Decision trees are prone to outliers.
   B) Decision trees are highly prone to overfitting.
   C) Decision trees are not easy to interpret
   D) None of the above.

ANS B) Decision trees are highly prone to overfitting.

3. Which of the following is an ensemble technique?
   A) SVM
   B) Logistic Regression
   C) Random Forest
   D) Decision tree

ANS : C) RANDOM FOREST

4. Suppose you are building a classification model for detection of a fatal disease where detection of the disease is most important. In this case which of the following metrics you would focus on?
   A) Accuracy
   B) Sensitivity
   C) Precision
   D) None of the above.

ANS : c) PRECISION

5. The value of AUC (Area under Curve) value for ROC curve of model A is 0.70 and of model B is 0.85. Which of these two models is doing better job in classification?
   A) Model A
   B) Model B
   C) both are performing equal
   D) Data Insufficient

Ans : d)Data Insufficient

In Q6 to Q9, more than one options are correct, Choose all the correct options:

6. Which of the following are the regularization technique in Linear Regression??
   A) Ridge
   B) R-squared
   C) MSE
   D) Lasso

   Ans: a)Ridge d)Lasso c) MSE

MACHINE LEARNING

7. Which of the following is not an example of boosting technique?
   A) Adaboost B) Decision Tree
   C) Random Forest D) Xgboost.

ANS : RANDOM FOREST AND DECISION TREE

8. Which of the techniques are used for regularization of Decision Trees?
   A) Pruning B) L2 regularization
   C) Restricting the max depth of the tree D) All of the above

ANS : all of the above

9. Which of the following statements is true regarding the Adaboost technique?
   A) We initialize the probabilities of the distribution as 1/n, where n is the number of data-points
   B) A tree in the ensemble focuses more on the data points on which the previous tree was not performing well
   C) It is example of bagging technique
   D) None of the above

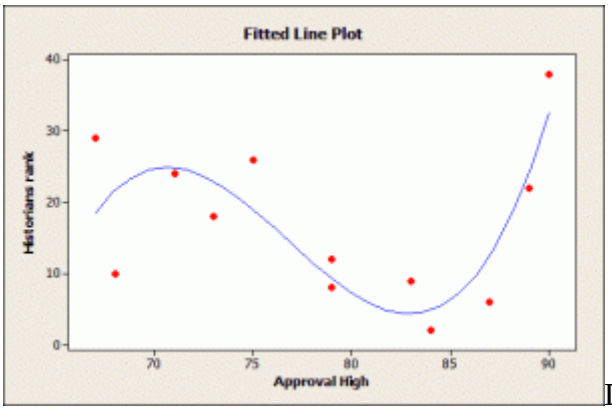
Ans : none of the above

Q10 to Q15 are subjective answer type questions, Answer them briefly.

10. Explain how does the adjusted R-squared penalize the presence of unnecessary predictors in the model?

R-squared tends to reward you for including too many independent variables in a regression model, and it doesn't provide any incentive to stop adding more. Adjusted R-squared and predicted R-squared use different approaches to help you fight that impulse to add too many. The protection that adjusted R-squared and predicted R-squared provide is critical because too many terms in a model can produce results that you can't trust. These statistics help you include the correct number of independent variables in your regression model.

MACHINE LEARNING



Does this graph display an actual relationship or is it an overfit model? This blog post shows you how to make this determination.

Multiple linear regression can seduce you! Yep, you read it here first. It's an incredibly tempting statistical analysis that practically begs you to include additional independent variables in your model. Every time you add a variable, the R-squared increases, which tempts you to add more. Some of the independent variables *will* be statistically significant. Perhaps there is an actual relationship? Or is it just a chance correlation?

You just pop the variables into the model as they occur to you or just because the data are readily available. Higher-order polynomials curve your regression line any which way you want. But are you fitting real relationships or just playing connect the dots? Meanwhile, the R-squared increases, mischievously convincing you to include yet more variables!

In my post about interpreting R-squared, I show how evaluating how well a linear regression model fits the data is not as intuitive as you may think. Now, I'll explore reasons why you need to use adjusted R-squared and predicted R-squared to help you specify a good regression model!

Some Problems with R-squared

Previously, I demonstrated that you cannot use R-squared to conclude whether your model is biased. To check for this bias, you need to check your residual plots. Unfortunately, there are yet more problems with R-squared that we need to address.

Problem 1: R-squared increases every time you add an independent variable to the model. The R-squared *never* decreases, not even when it's just a chance correlation between variables. A regression model that contains more independent variables than another model can look like it provides a better fit merely because it contains more variables.

Let's say you are comparing a model with five independent variables to a model with one variable and the five variable model has a higher R-squared. Is the model with five variables actually a better model, or does it just have more variables? To determine this, just compare the adjusted R-squared values!

The adjusted R-squared adjusts for the number of terms in the model. Importantly, its value increases only when the new term improves the model fit more than expected by chance alone.

MACHINE LEARNING

The adjusted R-squared value actually decreases when the term doesn't improve the model fit by a sufficient amount.

The example below shows how the adjusted R-squared increases up to a point and then decreases. On the other hand, R-squared blithely increases with each and every additional independent variable.

Vars	R-Sq	R-Sq(adj)
1	72.1	71.0
2	85.9	84.8
3	87.4	85.9
4	89.1	82.3
5	89.9	80.7

In this example, the researchers might want to include only three independent variables in their regression model. My R-squared blog post shows how an under-specified model (too few terms) can produce biased estimates. However, an overspecified model (too many terms) can reduce the model's precision. In other words, both the coefficient estimates and predicted values can have larger margins of error around them. That's why you don't want to include too many terms in the regression model!

What Is the Predicted R-squared?

Use predicted R-squared to determine how well a regression model makes predictions. This statistic helps you identify cases where the model provides a good fit for the existing data but isn't as good at making predictions. However, even if you aren't using your model to make predictions, predicted R-squared still offers valuable insights about your model.

Statistical software calculates predicted R-squared using the following procedure:

- It removes a data point from the dataset.
- Calculates the regression equation.
- Evaluates how well the model predicts the missing observation.
- And repeats this for all data points in the dataset.

Predicted R-squared helps you determine whether you are overfitting a regression model. Again, an overfit model includes an excessive number of terms, and it begins to fit the random noise in your sample.

By its very definition, it is not possible to predict random noise. Consequently, if your model fits a lot of random noise, the predicted R-squared value must fall. A predicted R-squared that is distinctly smaller than R-squared is a warning sign that you are overfitting the model. Try reducing the number of terms.

MACHINE LEARNING

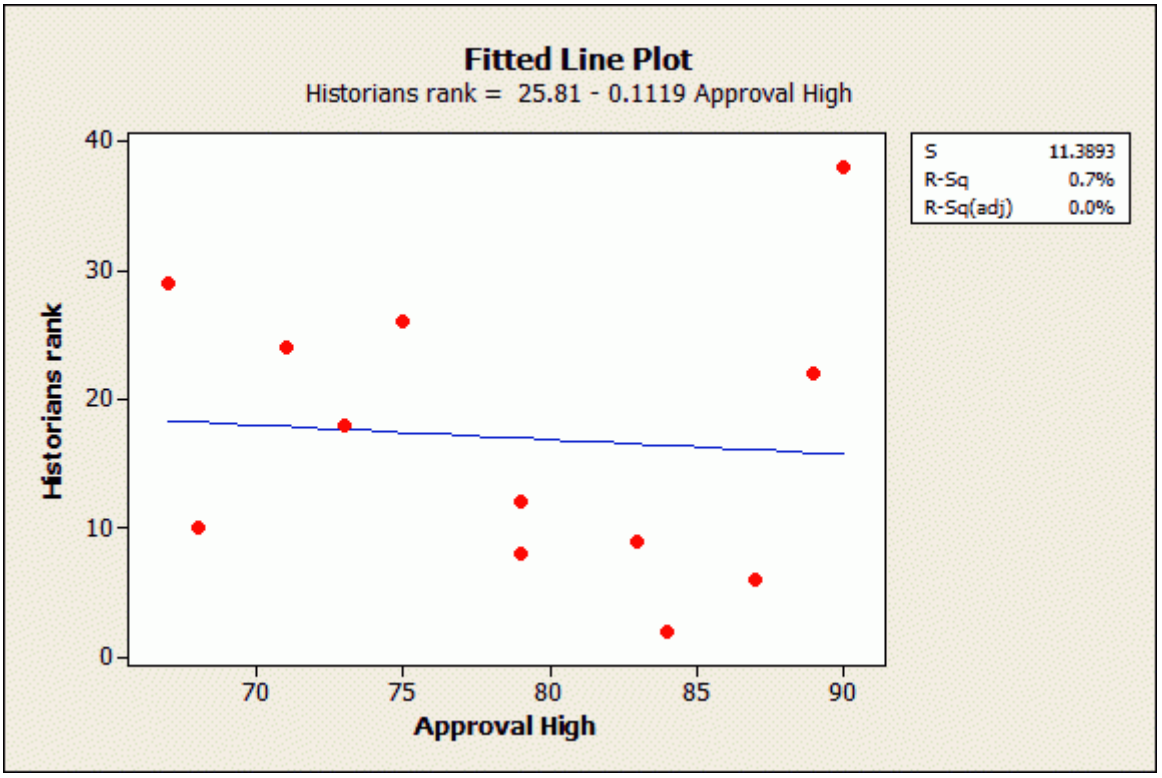
If I had to name my favorite flavor of R-squared, it would be predicted R-squared!

Related post: Overfitting Regression Models: Problems, Detection, and Avoidance

Example of an Overfit Model and Predicted R-squared

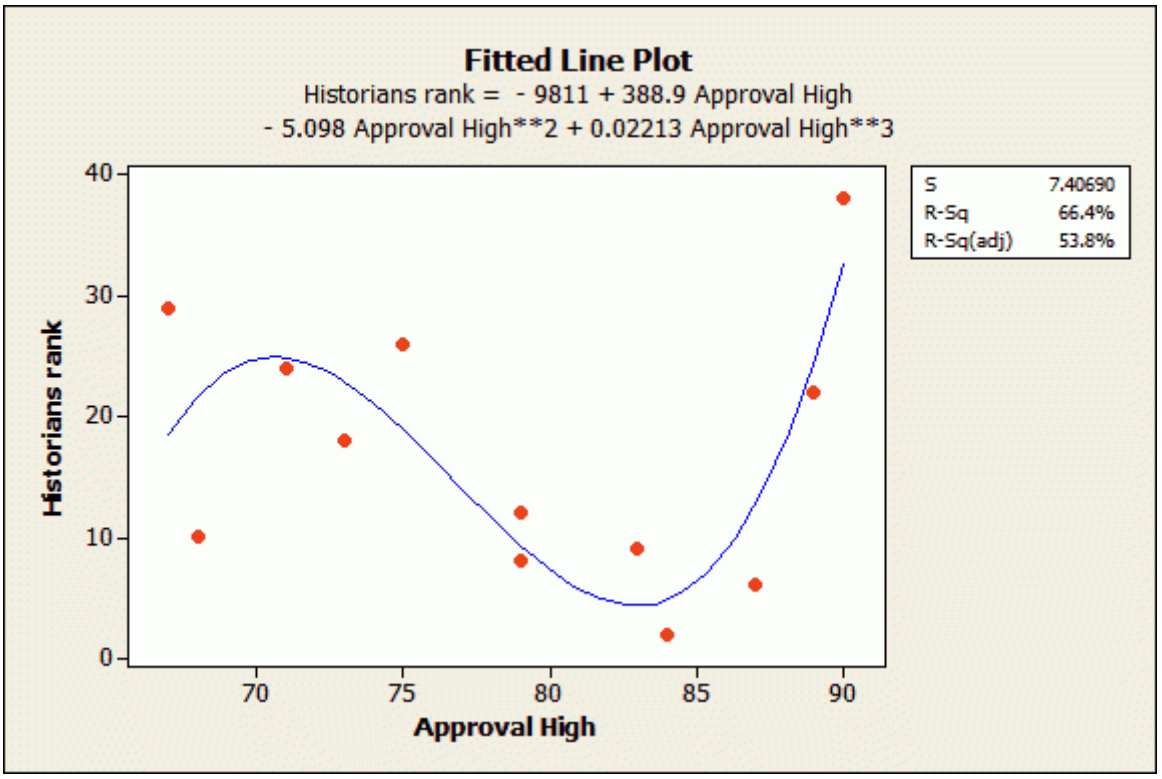
You can try this example using this CSV data file: PresidentRanking.

These data come from an analysis I performed that assessed the relationship between the highest approval rating that a U.S. President achieved and their rank by historians. I found no correlation between these variables, as shown in the fitted line plot. It's nearly a perfect example of no relationship because it is a flat line with an R-squared of 0.7%!



Now, imagine that we are chasing a high R-squared and we fit the model using a cubic term that provides an S-shape.

MACHINE LEARNING



Regression Analysis: Historians rank versus Approval High

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Regression	3	867.10	289.034	5.27	0.027
Approval High	1	438.35	438.347	7.99	0.022
Approval High*Approval High	1	460.23	460.225	8.39	0.020
Approval High*Approval High*Approval High	1	481.55	481.552	8.78	0.018
Error	8	438.90	54.862		
Lack-of-Fit	7	430.90	61.557	7.69	0.271
Pure Error	1	8.00	8.000		
Total	11	1306.00			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
7.40690	66.39%	53.79%	0.00%

Amazing! R-squared and adjusted R-squared look great! The coefficients are statistically significant because their p-values are all less than 0.05. I didn't show the residual plots, but they look good as well.

Hold on a moment! We're just twisting the regression line to force it to connect the dots rather than finding an actual relationship. We overfit the model, and the predicted R-squared of 0% gives this away.

If the predicted R-squared is small compared to R-squared, you might be over-fitting the model even if the independent variables are statistically significant.

MACHINE LEARNING

To read about the analysis above where I had to be extremely careful to avoid an overfit model, read Understanding Historians' Rankings of U.S. Presidents using Regression Models.

A Caution about the Problems of Chasing a High R-squared

All study areas involve a certain amount of variability that you can't explain. If you chase a high R-squared by including an excessive number of variables, you force the model to explain the unexplainable. This is not good. While this approach *can* obtain higher R-squared values, it comes at the cost of misleading regression coefficients, p-values, R-squared, and imprecise predictions.

Adjusted R-squared and predicted R-square help you resist the urge to add too many independent variables to your model.

- Adjusted R-square compares models with different numbers of variables.
- Predicted R-square can guard against models that are too complicated.

Remember, the great power that comes with multiple regression analysis requires your restraint to use it wisely!

11. Differentiate between Ridge and Lasso Regression.

Ridge Regression:

Ridge regression is a technique used to analyze multi-linear regression (multicollinear), also known as L2 regularization. It is Applied when predicted values are greater than the observed values.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

Above equation represents the formula for Ridge Regression! where,

Lambda (λ) in the equation is tuning parameter which is selected using cross-validation technique which makes the fit small by making squares small (β2) by adding shrinkage factor.

The shrinkage factor is lambda times the sum of squares of regression coefficients (The last element in the above equation).

MACHINE LEARNING

Lasso Regression:

Lasso stands for – Least Absolute Shrinkage and Selection Operator. It is a technique where data points are shrunk towards a central point, like the mean. Lasso is also known as L1 regularization.

It is applied when the model is overfitted or facing computational challenges.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

The above equation represents the formula for Lasso Regression! where, Lambda (λ) is a tuning parameter selected using the before Cross-validation technique.

Unlike Ridge Regression, Lasso uses |β| to penalize the high coefficients.

The shrinkage factor is lambda times the sum of Regression coefficients (The last factor in the above equation).

Without further delay, let us look into an example of how we can build Ridge and Lasso Models!
Let us Consider the dataset from an Ad Agency! that advertised their Ad through different forums such as TV, Radio, and Newspapers and recorded their sales against it!

Application

Import the required libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import *
from sklearn.metrics import r2_score
data = pd.read_csv('/content/Advertising Agency.csv')
data.head()
```

MACHINE LEARNING

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

Defining the Dependent and Independent Variable

```
X = data[['TV', 'Newspaper', 'Radio']] #independent variable


y = data[['Sales']] #dependent variable
```

Now, Let us Split the data into training and testing set

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.5, ran
dom_state=1)
```

Fitting into a Linear regression model

```
regression = LinearRegression()

regression.fit(x_train, y_train)
```

```
LinearRegression()
```

Linear Regression in Machine Learning

The article discusses about Linear Regression in Machine Learning to model a linear relationship between features in your data.

Checking the r2 for training and testing set

```
x_train_predict = regression.predict(x_train)
```

MACHINE LEARNING

```
r2_score(y_train, x_train_predict)
```
```
0.9026611359788856
```

```
x_test_predict = regression.predict(x_test)
```
```
r2_score(y_test, x_test_predict)
```
```
0.89993161028032
```

Let us try Ridge and Lasso regression technique and see whether the r2 score will improve or not!

Building Ridge regression model

```
ridge_regression = Ridge()
```
```
ridge_regression.fit(x_train,y_train)
```
```
Ridge()
```

```
x_test_predict_ridge = ridge_regression.predict(x_test)
```
```
r2_score(y_test, x_test_predict_ridge)
```
```
0.8999324149002296
```

Building Lasso regression model

```
lasso_regression = Lasso()
```
```
lasso_regression.fit(x_train,y_train)
```
```
Lasso()
```

```
x_test_predict_lasso = lasso_regression.predict(x_test)
```
```
r2_score(y_test, x_test_predict_lasso)
```
```
0.9009134444661037
```

Let us create a data frame to compare the r2 score and see which among the regressions (Ridge or Lasso) helped in model's performance

```
data1 = [['Linear',0.8788068760241095], ['Ridge',0.8788091619025382], ['Lasso',0.8806676233582594]]
```
```
summary = pd.DataFrame(data1, columns=['Model', 'R2 Score'])
```

MACHINE LEARNING

```
summary
```

	Model	R2 Score
0	Linear	0.902661
1	Ridge	0.899932
2	Lasso	0.900913

It is evident that by applying Lasso Regression model accuracy has improved!

12. What is VIF? What is the suitable value of a VIF for a feature to be included in a regression modelling?

Most research papers consider a VIF (Variance Inflation Factor) > 10 as an indicator of multicollinearity, but some choose a more conservative threshold of 5 or even 2.5.

So what threshold should YOU choose?

When choosing a VIF threshold, you should take into account that multicollinearity is a lesser problem when dealing with a large sample size compared to a smaller one. [Source]

MACHINE LEARNING

How to interpret a given VIF value?

Consider the following linear regression model:

$Y = \beta_0 + \beta_1 \times X_1 + \beta_2 \times X_2 + \beta_3 \times X_3 + \varepsilon$
For each of the independent variables X_1, X_2 and X_3 we can calculate the variance inflation factor (VIF) in order to determine if we have a multicollinearity problem. Here's the formula for calculating the VIF for X_1:

$$VIF_1 = \frac{1}{1 - R^2}$$

R^2 in this formula is the coefficient of determination from the linear regression model which has:

- X_1 as dependent variable
- X_2 and X_3 as independent variables

In other words, R^2 comes from the following linear regression model:
$X_1 = \beta_0 + \beta_1 \times X_2 + \beta_2 \times X_3 + \varepsilon$
And because R^2 is a number between 0 and 1:

- When R^2 is close to 1 (i.e. X_2 and X_3 are highly predictive of X_1): the VIF will be very large
- When R^2 is close to 0 (i.e. X_2 and X_3 are not related to X_1): the VIF will be close to 1

Therefore the range of VIF is between 1 and infinity.

Now, let's discuss how to interpret the following cases where:

1. VIF = 1
2. VIF = 2.5
3. VIF = $+\infty$

Example 1: VIF = 1

A VIF of 1 for a given independent variable (say for X_1 from the model above) indicates the total absence of collinearity between this variable and other predictors in the model (X_2 and X_3).

MACHINE LEARNING

Example 2: VIF = 2.5

If for example the variable X_3 in our model has a VIF of 2.5, this value can be interpreted in 2 ways:

1. The variance of β_3 (the regression coefficient of X_3) is 2.5 times greater than it would have been if X_3 had been entirely non-related to other variables in our model
2. The variance of β_3 is 150% greater than it would be if there were no collinearity effect at all between X_3 and other variables in our model

Why 150%?

This percentage is calculated by subtracting 1 (the value of VIF if there were no collinearity) from the actual value of VIF:

2.5 – 1 = 1.5

In percent:

1.5 * 100 / 100 = 150%.

Example 3: VIF = *Infinity*

An infinite value of VIF for a given independent variable indicates that it can be perfectly predicted by other variables in the model.

Looking at the equation above, this happens when R^2 approaches 1.

MACHINE LEARNING

13. Why do we need to scale the data before feeding it to the train the model?

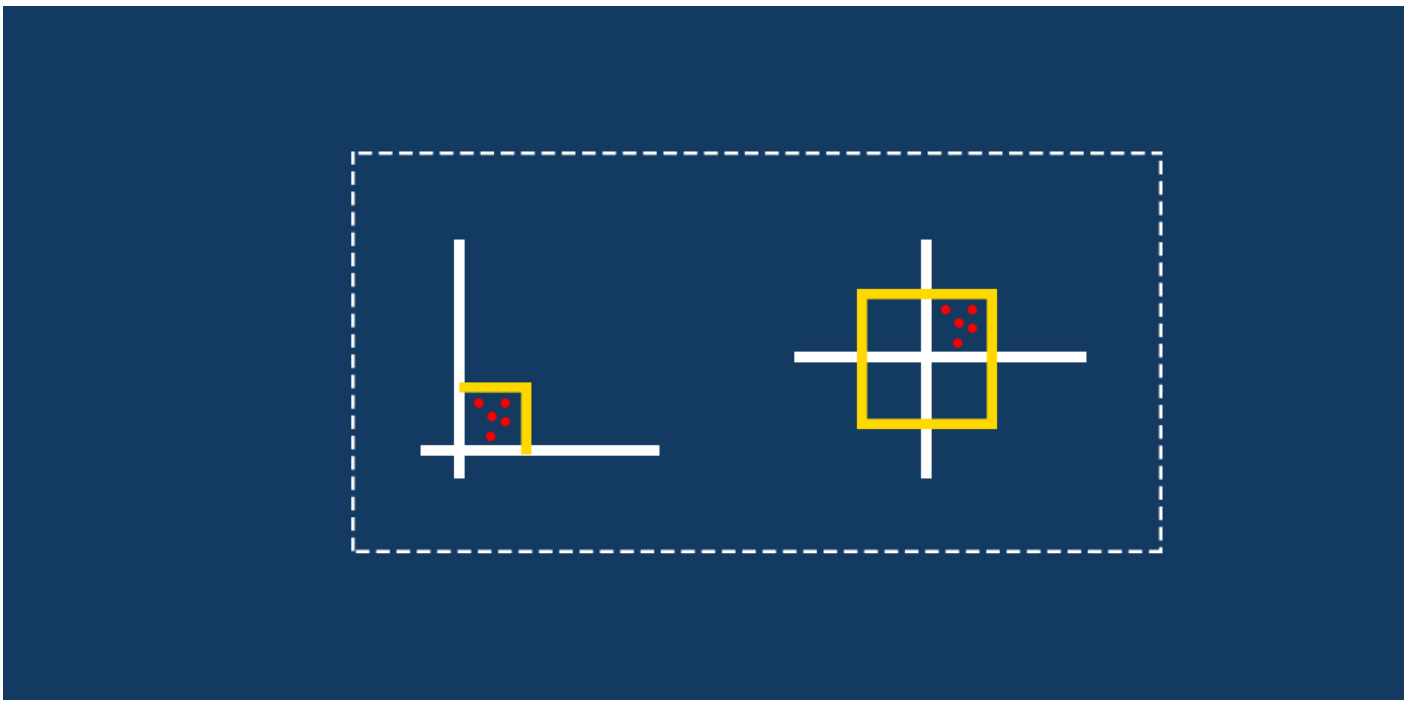
To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model

Introduction to Feature Scaling

I was recently working with a dataset from an ML Course that had multiple features spanning varying degrees of magnitude, range, and units. This is a significant obstacle as a few machine learning algorithms are highly sensitive to these features.

I'm sure most of you must have faced this issue in your projects or your learning journey. For example, one feature is entirely in kilograms while the other is in grams, another one is liters, and so on. How can we use these features when they vary so vastly in terms of what they're presenting?

This is where I turned to the concept of feature scaling. It's a crucial part of the data preprocessing stage but I've seen a lot of beginners overlook it (to the detriment of their machine learning model).



MACHINE LEARNING

Here's the curious thing about feature scaling – it improves (significantly) the performance of some machine learning algorithms and does not work at all for others. What could be the reason behind this quirk?

Also, what's the difference between normalization and standardization? These are two of the most commonly used feature scaling techniques in machine learning but a level of ambiguity exists in their understanding. When should you use which technique?

I will answer these questions and more in this article on feature scaling. We will also implement feature scaling in Python to give you a practice understanding of how it works for different machine learning algorithms.

Why Should we Use Feature Scaling?

The first question we need to address – why do we need to scale the variables in our dataset? Some machine learning algorithms are sensitive to feature scaling while others are virtually invariant to it. Let me explain that in more detail.

Gradient Descent Based Algorithms

Machine learning algorithms like linear regression, logistic regression, neural network, etc. that use gradient descent as an optimization technique require data to be scaled. Take a look at the formula for gradient descent below:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and

MACHINE LEARNING

that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

Having features on a similar scale can help the gradient descent converge more quickly towards the minima.

Distance-Based Algorithms

Distance algorithms like KNN, K-means, and SVM are most affected by the range of features. This is because behind the scenes **they are using distances between data points to determine their similarity.**

For example, let's say we have data containing high school CGPA scores of students (ranging from 0 to 5) and their future incomes (in thousands Rupees):

	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40
3	4	4.5	50
4	5	4.2	52

Since both the features have different scales, there is a chance that higher weightage is given to features with higher magnitude. This will impact the performance of the machine learning algorithm and obviously, we do not want our algorithm to be biassed towards one feature.

Therefore, we scale our data before employing a distance based algorithm so that all the features contribute equally to the result.

MACHINE LEARNING

	Student	CGPA	Salary '000
0	1	-1.184341	1.520013
1	2	-1.184341	-1.100699
2	3	0.416120	-1.100699
3	4	1.216350	0.209657
4	5	0.736212	0.471728

The effect of scaling is conspicuous when we compare the Euclidean distance between data points for students A and B, and between B and C, before and after scaling as shown below:

- Distance AB before scaling =>$\sqrt{(40-60)^2+(3-3)^2}=20$
- Distance BC before scaling =>$\sqrt{(40-40)^2+(4-3)^2}=1$
- Distance AB after scaling =>$\sqrt{(1.1+1.5)^2+(1.18-1.18)^2}=2.6$
- Distance BC after scaling =>$\sqrt{(1.1-1.1)^2+(0.41+1.18)^2}=1.59$

Scaling has brought both the features into the picture and the distances are now more comparable than they were before we applied scaling.

Tree-Based Algorithms

Tree-based algorithms, on the other hand, are fairly insensitive to the scale of the features. Think about it, a decision tree is only splitting a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. This split on a feature is not influenced by other features.

So, there is virtually no effect of the remaining features on the split. This is what makes them invariant to the scale of the features!

MACHINE LEARNING

14. What are the different metrics which are used to check the goodness of fit in linear regression?

There are three error metrics that are commonly used for evaluating and reporting the performance of a regression model; they are: **Mean Squared Error (MSE). Root Mean Squared Error (RMSE). Mean Absolute Error (MAE)**

Root Mean Squared Error (RMSE)and Mean Absolute Error (MAE) are metrics used to evaluate a Regression Model. These metrics tell us how accurate our predictions are and, what is the amount of deviation from the actual values.

Technically, RMSE is the **R**oot of the **M**ean of the **S**quare of **E**rrors and MAE is the **M**ean of **A**bsolute value of **E**rrors. Here, errors are the differences between the predicted values (values predicted by our regression model) and the actual values of a variable. They are calculated as follows :

$$RMSE = \sqrt{\frac{\sum (y_i - y_p)^2}{n}}$$

$$MAE = \frac{|(y_i - y_p)|}{n}$$

y_i = actual value
y_p = predicted value
n = number of observations/rows

On close inspection, you will see that both are average of errors.

Let's understand this with an example. Say, I want to predict the salary of a data scientist based on the number of years of experience. So, salary is my target variable (Y) and experience is the independent variable(X). I have some

MACHINE LEARNING

random data on X and Y and we will use Linear Regression to predict salary. Let's use pandas and scikit-learn for data loading and creating linear model.

```
import pandas as pd
from sklearn.linear_model import LinearRegressionsal_data={"Exp":[2,2.2,
2.8, 4, 7, 8, 11, 12, 21, 25],
          "Salary": [7, 8, 11, 15, 22, 29, 37 ,45.7, 49, 52]}#Load data
into a pandas Dataframe
df=pd.DataFrame(sal_data)
df.head(3)
```

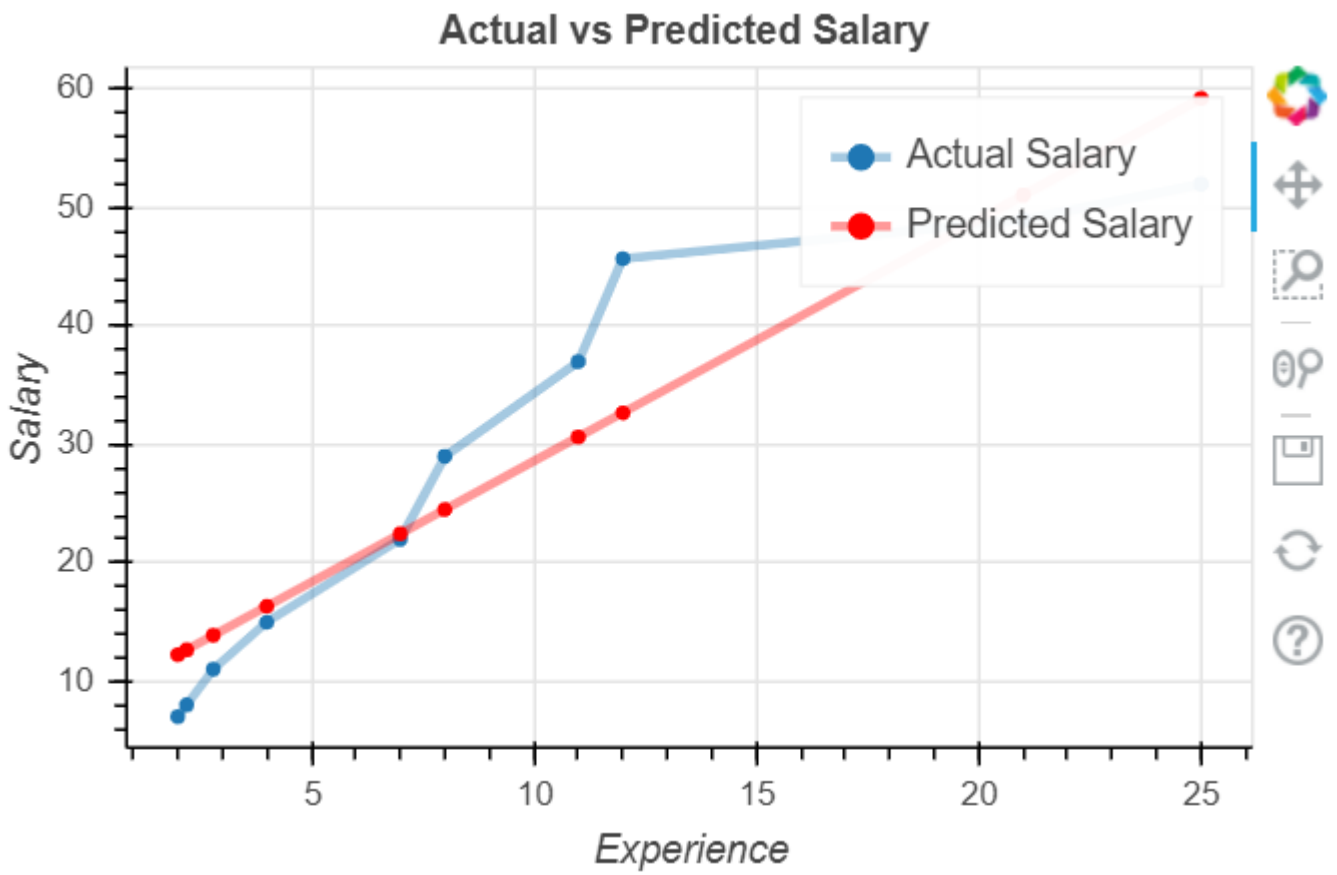
	Exp	Salary
0	2.0	7.0
1	2.2	8.0
2	2.8	11.0

```
#Selecting X and y variablesX=df[['Experience']]
y=df.Salary#Creating a Simple Linear Regression Model to predict
salarieslm=LinearRegression()
lm.fit(X,y)#Prediction of salaries by the model
yp=lm.predict(X)
print(yp)[12.23965934 12.64846842 13.87489568 16.32775018 22.45988645
24.50393187 30.63606813 32.68011355 51.07652234 59.25270403]
```

Now, we have 'yp' — our array of salary prediction and we will evaluate our model by plotting predicted(yp) and actual salary(y). I am using bohek for my visualizations.

```
from bokeh.plotting import figure, show, output_filep=figure(title="Actual
vs Predicted Salary", width=450, height=300)
p.title.align = 'center'
p.circle(df.Exp, df.Salary)
p.line(df.Exp, df.Salary, legend_label='Actual Salary', line_width=3,
line_alpha=0.4)
p.circle(df.Exp, yp, color="red")
p.line(df.Exp,yp, color="red",legend_label='Predicted Salary',
line_width=3, line_alpha=0.4)p.xaxis.axis_label = 'Experience'
p.yaxis.axis_label = 'Salary'show(p)
```

MACHINE LEARNING



From the graph above, we see that there is a gap between predicted and actual data points. Statistically, this gap/difference is called residuals and commonly called error, and is used in RMSE and MAE. Scikit-learn provides metrics library to calculate these values. However, we will compute RMSE and MAE by using the above mathematical expressions. Both methods will give you the same result.

```
import numpy as np
print(f'Residuals: {y-yp}')
np.sqrt(np.mean(np.square(y-yp)))  #RMSEnp.mean(abs(y-yp))
#MAE#RMSE/MAE computation using sklearn library
from sklearn.metrics import mean_squared_error,
mean_absolute_errornp.sqrt(mean_squared_error(y, yp))
mean_absolute_error(y, yp)6.48
5.68
```

This is our baseline model. MAE is around 5.7 — which seems to be higher. Now our goal is to improve this model by reducing this error.

Let's run a polynomial transformation on "experience" (X) with the same model and see if our errors reduce.

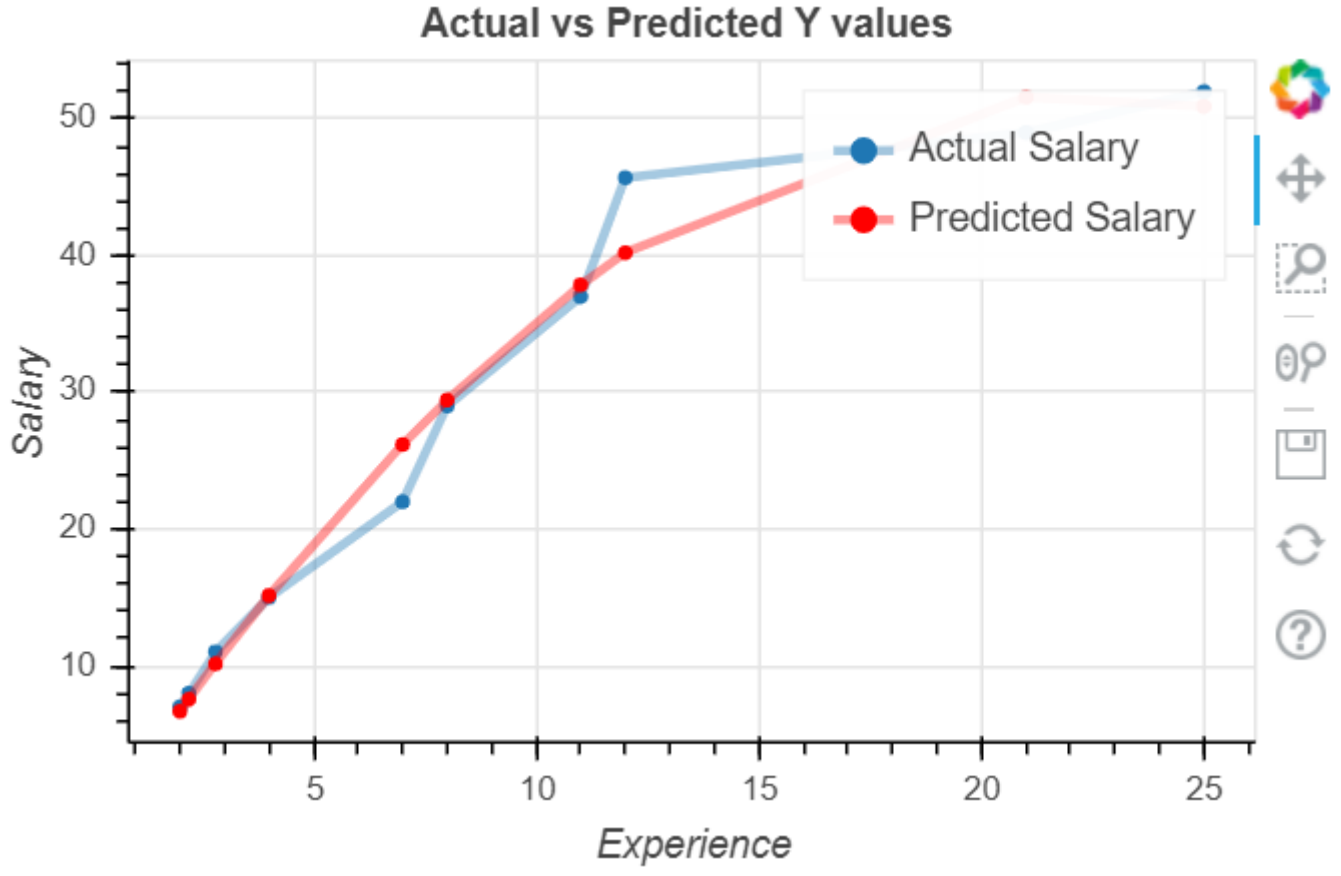
```
from sklearn.preprocessing import PolynomialFeatures
pf=PolynomialFeatures()      #Linear Equation of degree 2
X_poly=pf.fit_transform(X)
lm.fit(X_poly, y)
yp=lm.predict(X_poly)
```

I have used Scikit-learn PolynomialFeatures to create a matrix of 1, X, and X2 and passed that as input to my model.

Calculating our error metrics and ....

```
#RMSE and MAE
np.sqrt(np.mean(np.square(y-yp)))
np.mean(abs(y-yp))2.3974
1.6386
```

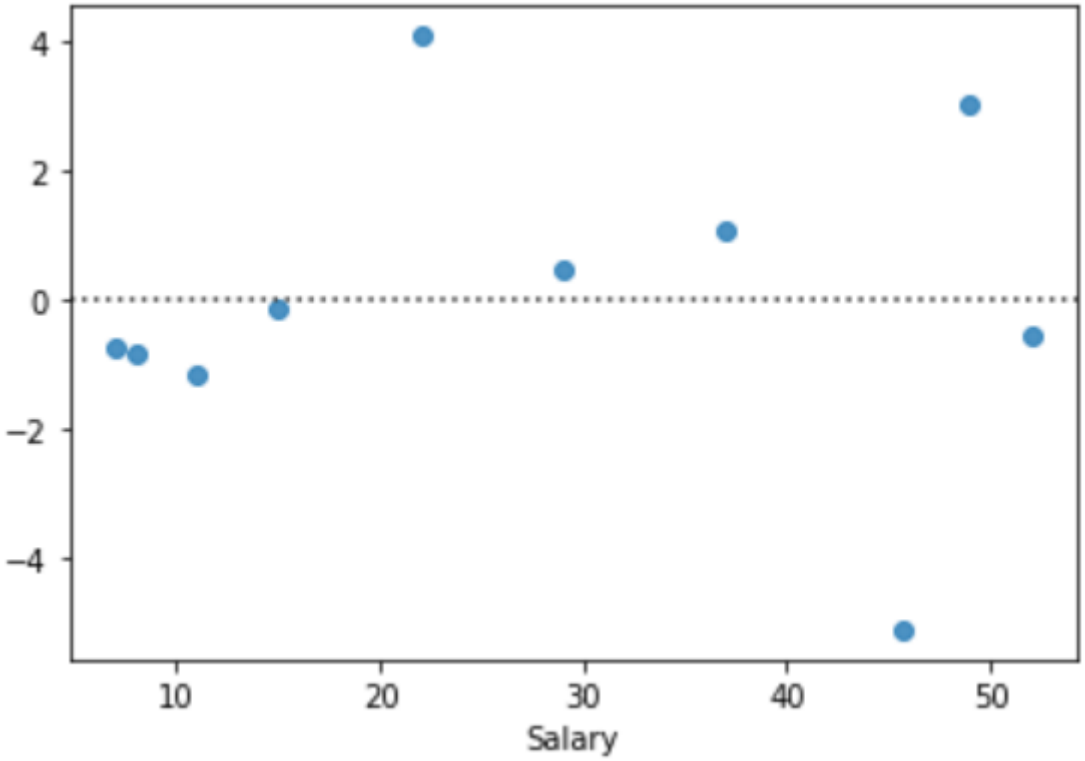
Voilaaa... they are much lower this time. It fits better than our baseline model! Let's plot y and yp (like how we did before) to check the overlap.



MACHINE LEARNING

The gap between the 2 lines has reduced. Let's observe the distribution of residuals or errors(y-yp) using seaborn's residual plot function

```
print(y-yp)   #residuals
[ 0.333921 0.447306  0.84028668 -0.136044 -4.190238 -0.434767
 -0.847751  5.488121 -2.584481  1.083648]import seaborn as sns
sns.residplot(y, yp)
plt.show()
```



We see that residuals tend to concentrate around the x-axis, which makes sense because they are negligible.

There is a third metric — R-Squared score, usually used for regression models. This measures the amount of variation that can be explained by our model i.e. percentage of correct predictions returned by our model. It is also called the coefficient of determination and calculated by the formula:

MACHINE LEARNING

$$1 - \frac{\sum (y_i - y_p)^2}{\sum (y_i - \overline{y_i})^2}$$

y_i = actual value

y_p = predicted value

$\overline{y_i}$ = mean of all actual values

Let's compute R2 mathematically using the formula and using sklearn library and compare the values. Both methods should give you the same result.

```
#Calculating R-Squared manually
a=sum(np.square(y-yp))           # a -> sum of square of residuals
b=sum(np.square(y-np.mean(y)))   # b -> total sum of sqauresr2_value = 1-
(a/b)
0.979#calculating r2 using sklearn
from sklearn.metrics import r2_score
print(r2_score(y, yp))
0.979
```

Thus, overall we can interpret that 98% of the model predictions are correct and the variation in the errors is around 2 units. For an ideal model, RMSE/MAE=0 and R2 score = 1, and all the residual points lie on the X-axis. Achieving such a value for any business solution is almost impossible!

Some of the techniques we can use to improve our model accuracy include:

- Transform/scale features
- Treat outliers (if many)
- Add new features/feature engineering
- Use different algorithms

MACHINE LEARNING

- Model hyperparameter tuning

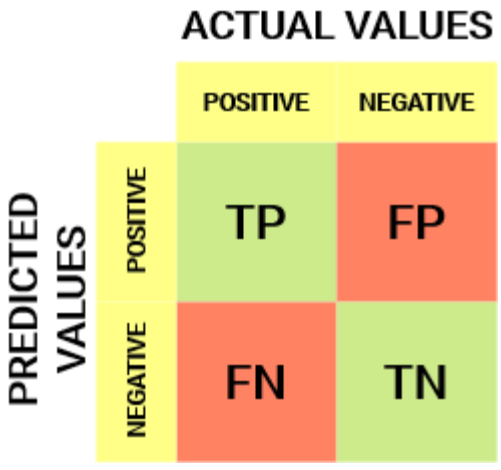
15. From the following confusion matrix calculate sensitivity, specificity, precision, recall and accuracy.

Actual/Predicted	True	False
True	1000	50
False	250	1200

The million-dollar question – what, after all, is a confusion matrix?

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:



MACHINE LEARNING

Let's decipher the matrix:

- The target variable has two values: **Positive** or **Negative**
- The **columns** represent the **actual values** of the target variable
- The **rows** represent the **predicted values** of the target variable

But wait – what's TP, FP, FN and TN here? That's the crucial part of a confusion matrix.

Let's understand each term below.

Understanding True Positive, True Negative, False Positive and False Negative in a Confusion Matrix

True Positive (TP)

- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value
- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

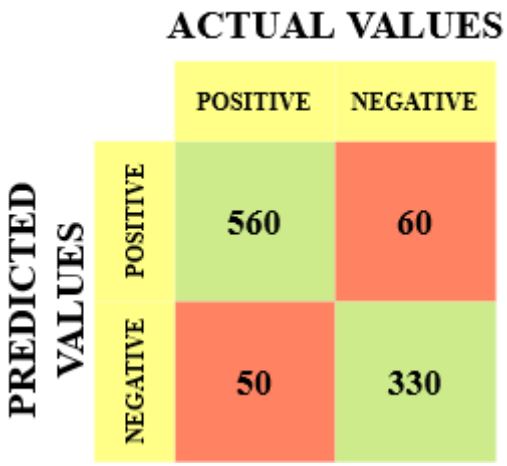
- The predicted value was falsely predicted
- The actual value was negative but the model predicted a positive value
- Also known as the **Type 1 error**

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted
- The actual value was positive but the model predicted a negative value
- Also known as the **Type 2 error**

MACHINE LEARNING

Let me give you an example to better understand this. Suppose we had a classification dataset with 1000 data points. We fit a classifier on it and get the below confusion matrix:



The different values of the Confusion matrix would be as follows:

- True Positive (TP) = 560; meaning 560 positive class data points were correctly classified by the model
- True Negative (TN) = 330; meaning 330 negative class data points were correctly classified by the model
- False Positive (FP) = 60; meaning 60 negative class data points were incorrectly classified as belonging to the positive class by the model
- False Negative (FN) = 50; meaning 50 positive class data points were incorrectly classified as belonging to the negative class by the model

This turned out to be a pretty decent classifier for our dataset considering the relatively larger number of true positive and true negative values.

Remember the Type 1 and Type 2 errors. Interviewers love to ask the difference between these two! You can prepare for all this better from our Machine Learning Course Online

Why Do We Need a Confusion Matrix?

Before we answer this question, let's think about a hypothetical classification problem.

MACHINE LEARNING

Let's say you want to predict how many people are infected with a contagious virus in times before they show the symptoms, and isolate them from the healthy population (ringing any bells, yet?). The two values for our target variable would be: Sick and Not Sick.

Now, you must be wondering – why do we need a confusion matrix when we have our all-weather friend – Accuracy? Well, let's see where accuracy falters.

Our dataset is an example of an **imbalanced dataset**. There are 947 data points for the negative class and 3 data points for the positive class. This is how we'll calculate the accuracy:

ID	Actual Sick?	Predicted Sick?	Outcome
1	1	1	TP
2	0	0	TN
3	0	0	TN
4	1	1	TP
5	0	0	TN
6	0	0	TN
7	1	0	FP
8	0	1	FN
9	0	0	TN
10	1	0	FP
:	:	:	:
1000	0	0	FN

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Let's see how our model performed:

The total outcome values are:

TP = 30, TN = 930, FP = 30, FN = 10

So, the accuracy for our model turns out to be:

MACHINE LEARNING

$$Accuracy = \frac{30 + 930}{30 + 30 + 930 + 10} = 0.96$$

96%! Not bad!

But it is giving the wrong idea about the result. Think about it.

Our model is saying "I can predict sick people 96% of the time". However, it is doing the opposite. It is predicting the people who will not get sick with 96% accuracy while the sick are spreading the virus!

Do you think this is a correct metric for our model given the seriousness of the issue? Shouldn't we be measuring how many positive cases we can predict correctly to arrest the spread of the contagious virus? Or maybe, out of the correctly predicted cases, how many are positive cases to check the reliability of our model?

This is where we come across the dual concept of Precision and Recall.

Precision vs. Recall

Precision tells us how many of the correctly predicted cases actually turned out to be positive.

Here's how to calculate Precision:

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

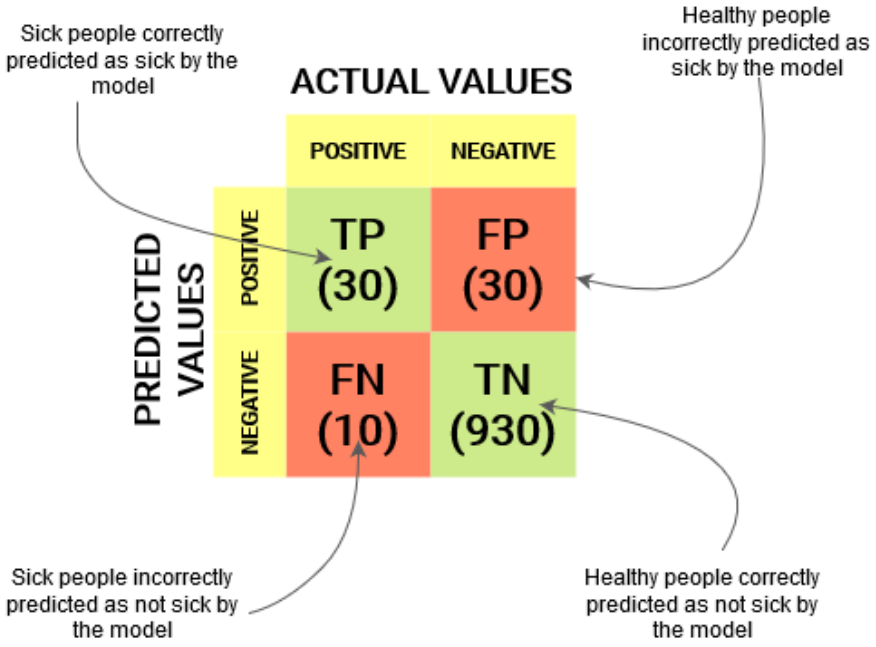
This would determine whether our model is reliable or not.

Recall tells us how many of the actual positive cases we were able to predict correctly with our model.

MACHINE LEARNING

And here's how we can calculate Recall:

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



We can easily calculate Precision and Recall for our model by plugging in the values into the above questions:

$$Precision = \frac{30}{30 + 30} = 0.5$$

$$Recall = \frac{30}{30 + 10} = 0.75$$

50% percent of the correctly predicted cases turned out to be positive cases. Whereas 75% of the positives were successfully predicted by our model. Awesome!

Precision is a useful metric in cases where False Positive is a higher concern than False Negatives.

MACHINE LEARNING

Precision is important in music or video recommendation systems, e-commerce websites, etc. Wrong results could lead to customer churn and be harmful to the business.

Recall is a useful metric in cases where False Negative trumps False Positive.

Recall is important in medical cases where it doesn't matter whether we raise a false alarm but the actual positive cases should not go undetected!

In our example, Recall would be a better metric because we don't want to accidentally discharge an infected person and let them mix with the healthy population thereby spreading the contagious virus. Now you can understand why accuracy was a bad metric for our model.

But there will be cases where there is no clear distinction between whether Precision is more important or Recall. What should we do in those cases? We combine them!

F1-Score

In practice, when we try to increase the precision of our model, the recall goes down, and vice-versa. The F1-score captures both the trends in a single value:

$$F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

But there is a catch here. The interpretability of the F1-score is poor. This means that we don't know what our classifier is maximizing – precision or recall? So, we use it in

MACHINE LEARNING

combination with other evaluation metrics which gives us a complete picture of the result.

Confusion Matrix using scikit-learn in Python

You know the theory – now let's put it into practice. Let's code a confusion matrix with the Scikit-learn (sklearn) library in Python.

Python Code:

Sklearn has two great functions: **confusion_matrix()** and **classification_report()**.

- Sklearn **confusion_matrix()** returns the values of the Confusion matrix. The output is, however, slightly different from what we have studied so far. It takes the rows as Actual values and the columns as Predicted values. The rest of the concept remains the same.
- Sklearn **classification_report()** outputs precision, recall and f1-score for each target class. In addition to this, it also has some extra values: **micro avg**, **macro avg**, and **weighted avg**

Mirco average is the precision/recall/f1-score calculated for all the classes.

$$\text{Micro avg Precision} = \frac{TP1 + TP2}{TP1 + TP2 + FP1 + FP2}$$

Macro average is the average of precision/recall/f1-score.

$$\text{Macro avg Precision} = \frac{P1 + P2}{2}$$

Weighted average is just the weighted average of precision/recall/f1-score.

MACHINE LEARNING

Confusion Matrix for Multi-Class Classification

How would a confusion matrix work for a multi-class classification problem? Well, don't scratch your head! We will have a look at that here.

Let's draw a confusion matrix for a multiclass problem where we have to predict whether a person loves Facebook, Instagram or Snapchat. The confusion matrix would be a 3 x 3 matrix like this: