Statistics Foundation

Model Tuning

Digital Vidya

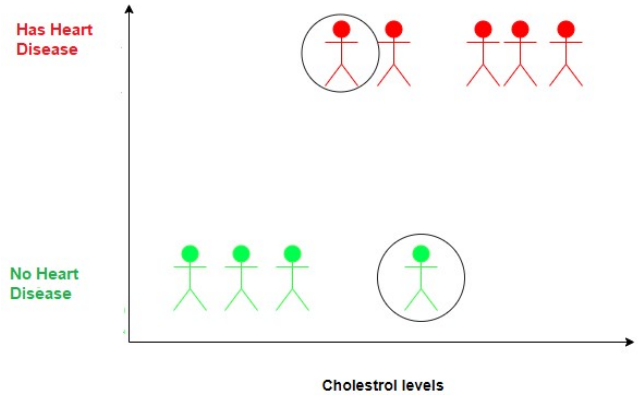
Model Tuning

- **ROC Curve and AUC Value**
  - ROC and AUC curves are important evaluation metrics for calculating the performance of any classification model
  - The concept of ROC and AUC builds upon the knowledge of Confusion Matrix, Specificity and Sensitivity
  - It is important to keep in mind that the concept of ROC and AUC can apply to more than just Logistic Regression

Model Tuning

ROC Curve and AUC Value

- Consider a hypothetical example containing a group of people
- The y-axis has two categories
  - Has Heart Disease represented by red people
  - Does not have Heart Disease represented by green circles
- Along the x-axis, we have cholesterol levels and the classifier tries to classify people into two categories depending upon their cholesterol levels



Few observations:

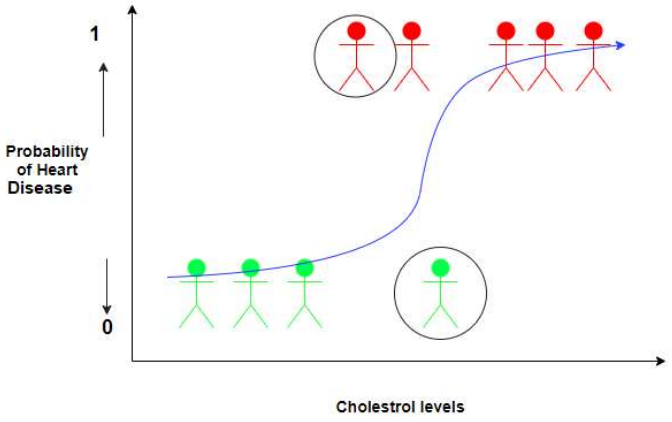
- Circled Green person has a high level of cholesterol but does not have heart disease
- This may be due to the reason that now the person is observing a better lifestyle and exercising regularly
- Circled Red person has low cholesterol levels still had a heart attack
- This may be due to the reason that he has other heart-related issues

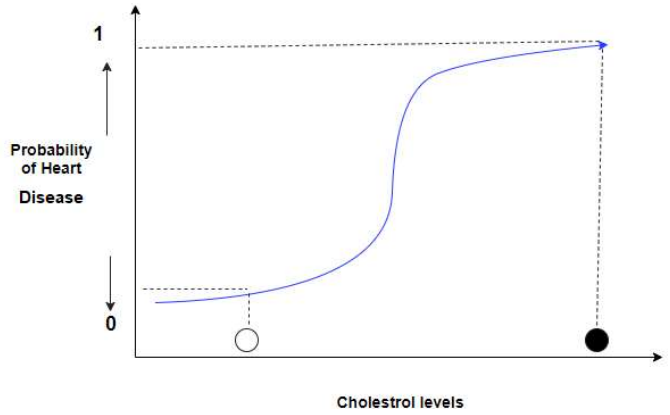
Digital Vidya

Model Tuning

Fitting a Logistic Regression

- Now if we fit a Logistic Regression curve to the data, the Y-axis will be converted to the Probability of a person having a heart disease based on the Cholesterol levels





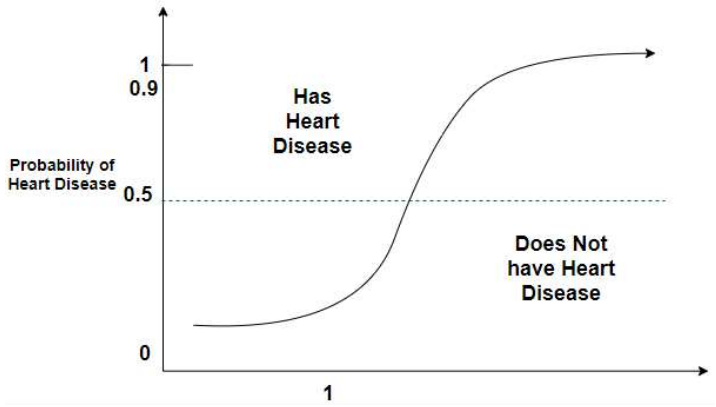
- The white dot represents a person having a lower heart disease probability than the person represented by the black dot

Digital Vidya

Model Tuning

Fitting a Logistic Regression

- To classify the people in the two categories, we need a way to turn probabilities into classifications
- One way is to set a threshold at 0.5
- Classify the people who have a probability of heart disease > 0.5 as “having a heart disease”
- Classify the people who have a probability of heart disease < 0.5 as “ not having a heart disease”

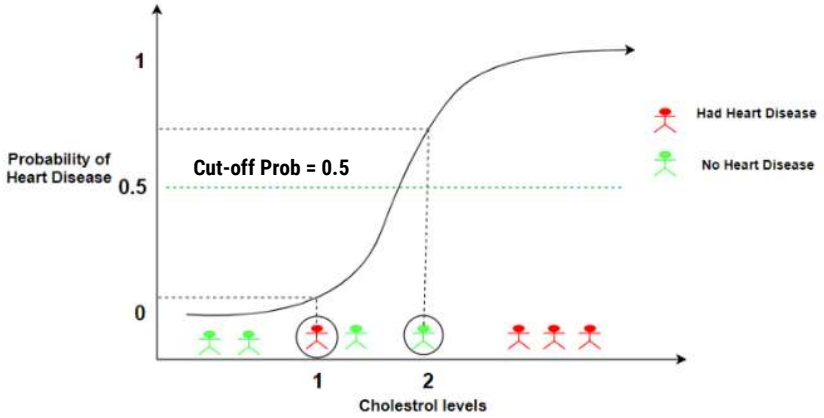


Digital Vidya

Model Tuning

Evaluating Logistic Regression

- Our Logistic Regression model correctly classifies all people except the persons 1 and 2
- We know Person 1 has heart disease but our model classifies it as otherwise
- We also know person 2 doesn't have heart disease but again our model classifies it incorrectly



	Actual	
Predicted	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	**True Positives**	**False Positives**
Doesnot have Heart Disease	**False Negatives**	**True Negatives**

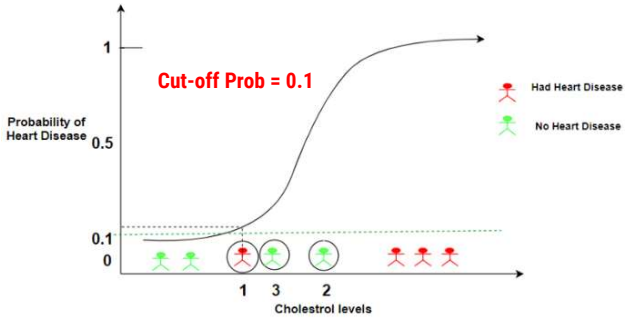
	Actual	
Predicted	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	3	1
Doesnot have Heart Disease	1	3

Digital Vidya

Model Tuning

Identifying the Correct Thresholds

- Setting the Threshold to 0.1
- This would correctly identify all people who have heart disease
- The person labeled 1 is also correctly classified to be a heart patient
- However, it would also increase the number of False Positives since now person 2 and 3 will be wrongly classified as having heart disease
- Therefore a lower threshold:
  - Increases the number of False Positives
  - Decreases the number of False Negatives



- In this case, it becomes important to identify people having a heart disease correctly so that the corrective measures can be taken else heart disease can lead to serious complications
- This means lowering the threshold is a good idea even if it results in more False Positive cases

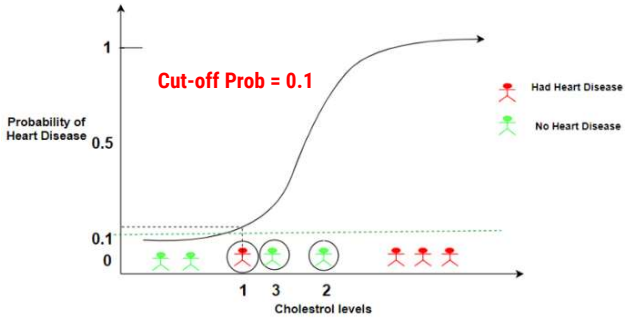
	Confusion Matrix	Actual: Has Heart Disease	Actual: Doesnot have Heart Disease
Predicted	Has Heart Disease	4	2
Predicted	Doesnot have Heart Disease	0	2

Digital Vidya

Model Tuning

Identifying the Correct Thresholds

- Setting the Threshold to 0.9
- This would now correctly identify all people who do not have heart disease
- The person labelled 1, would be incorrectly classified having no heart disease
- Therefore a lower threshold:
  - Decreases the number of False Positives
  - Increases the number of False Negatives



- The threshold could be set to any value between 0 and 1
- So how do we determine which threshold is the best?
- Do we need to experiment with all the threshold values?
- Every threshold results in a different confusion matrix and a number of thresholds will result in a large number of confusion matrices which is not the best way to work

Actual

Confusion Matrix	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	3	0
Doesnot have Heart Disease	1	4

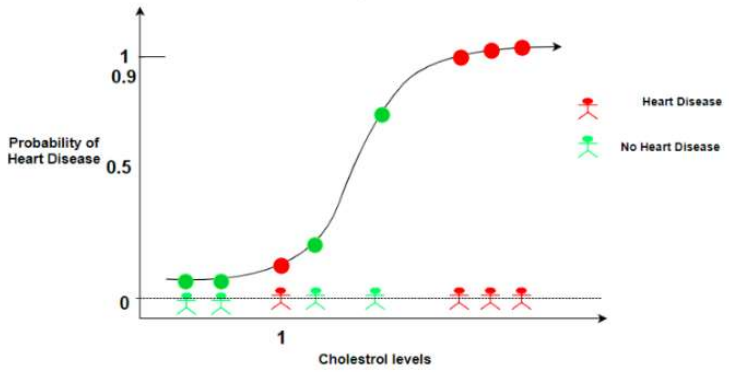
Digital Vidya

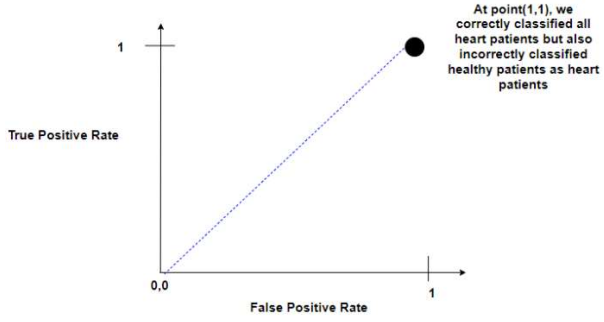
Model Tuning

ROC Graphs (Step 1)

- ROC(Receiver Operator Characteristic Curve) can help in deciding the best threshold value
- It is generated by plotting the True Positive Rate (y-axis) against the False Positive Rate (x-axis)

- Threshold classifying all people as having heart disease.



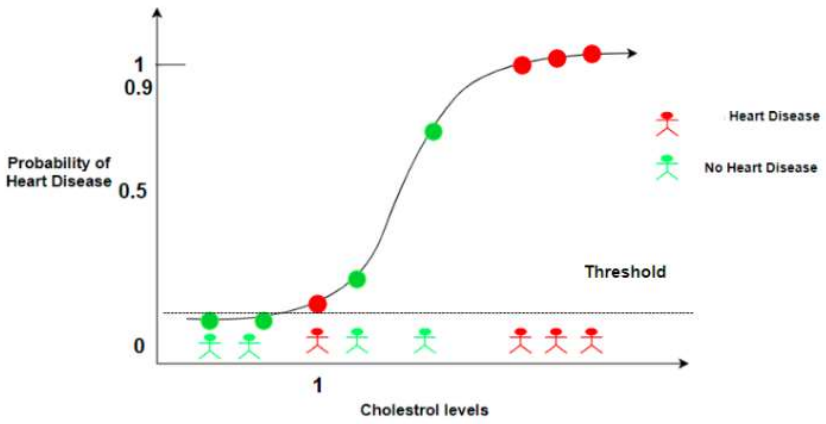


Digital Vidya

Model Tuning

ROC Graphs (Step 2)

- Increasing the Threshold slightly so that only the two people with the least cholesterol value are below the threshold
- This means this threshold is better than the previous one



Actual

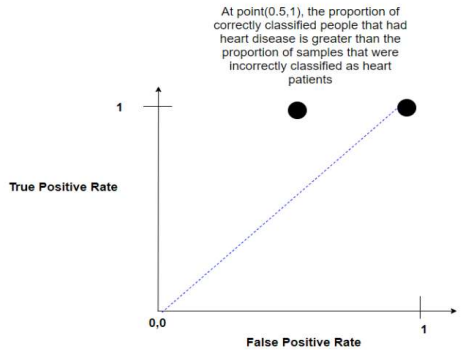
Confusion Matrix	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	4	2
Doesnot have Heart Disease	0	2

Predicted

True Positive Rate = Sensitivity = $\frac{4}{4+0} = 1$

False Positive Rate = 1 - Specificity = $\frac{2}{2+2} = 0.5$

Let's plot this point (0.5,1) on the ROC graph.



Digital Vidya

Model Tuning

ROC Graphs (Step 3)

- Now if we go on increasing the threshold values
- And reach a point where we get the following confusion matrix:

Confusion Matrix	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	3	0
Doesnot have Heart Disease	1	4

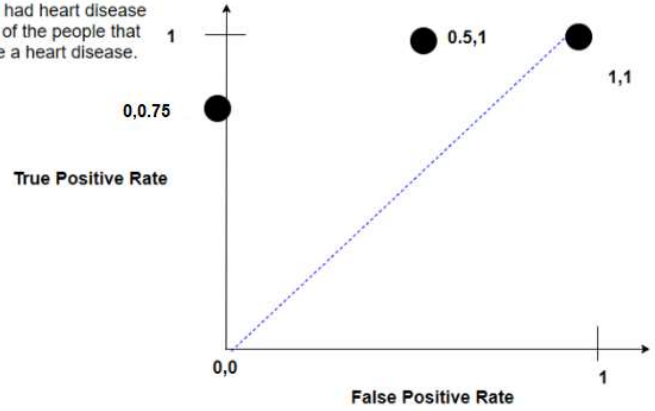
Predicted

True Positive Rate = Sensitivity = $\frac{3}{3+1} = 0.75$

False Positive Rate = 1 - Specificity = $\frac{0}{0+4} = 0$

Let's plot this point (0,0.75) on the ROC graph.

At point(**0,0.75**), the threshold correctly classified 75% of the people that had heart disease and 100% of the people that didn't have a heart disease.



By far this is the best threshold that we have got since it predicted no false positives

Digital Vidya

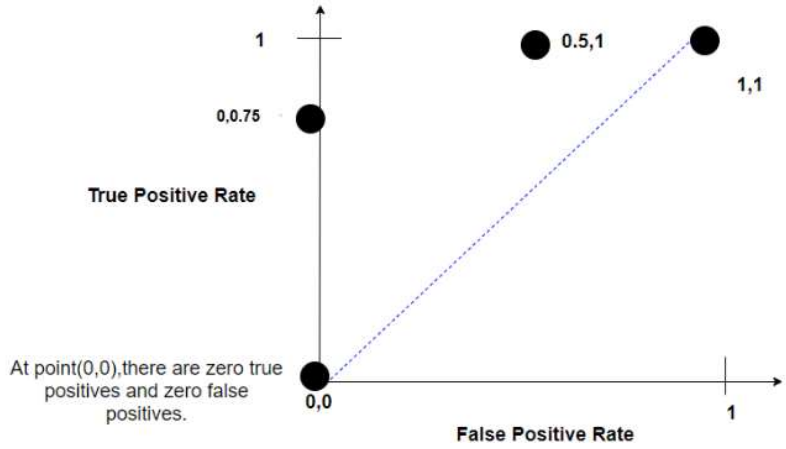
Model Tuning

ROC Graphs (Step 4)

- Lastly, we choose a threshold where we classify all people as not having a heart disease i.e Threshold of 1

We can then connect the dots which gives us a ROC graph

The graph, in this case, would be at (0,0):

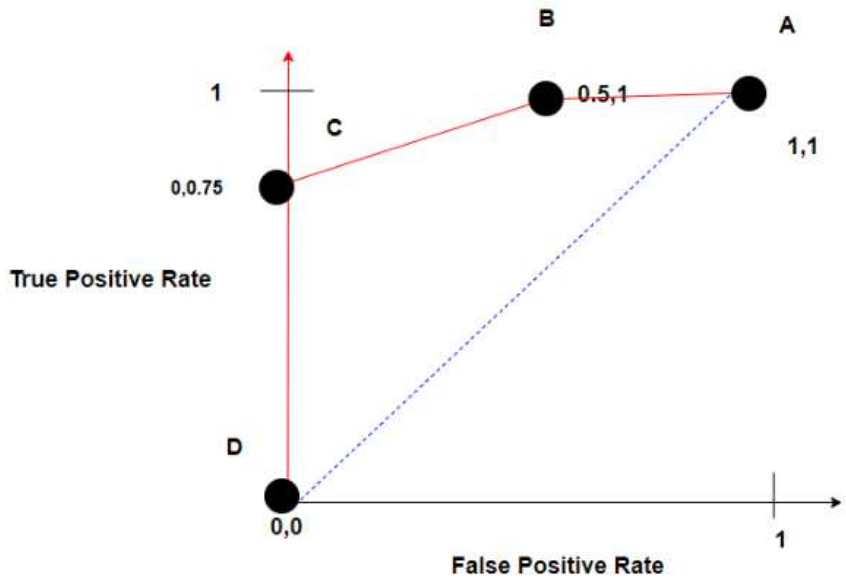


Digital Vidya

Model Tuning

ROC Graphs (Step 5)

The ROC graph summarises the confusion matrices produced for each threshold without having to actually calculate them



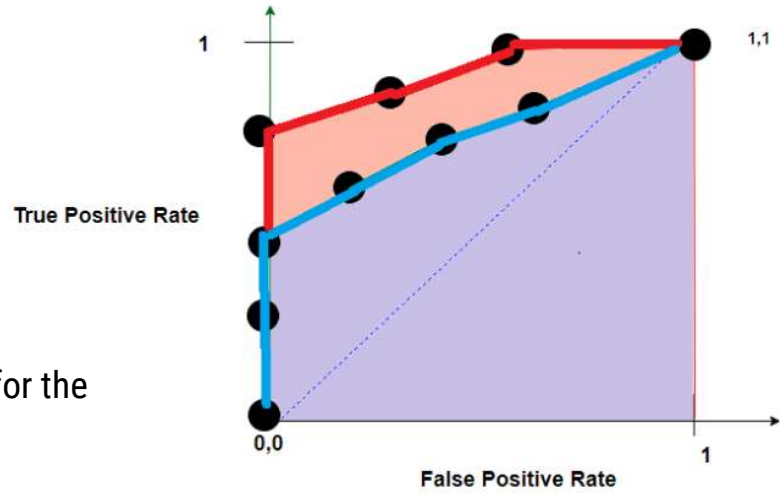
Just by glancing over the graph, we can conclude that threshold C is better than threshold B and depending on
how many False Positives that we are willing to accept, we can choose the optimal threshold

Digital Vidya

Model Tuning

AUC

- AUC stands for Area under the curve
- AUC gives the rate of successful classification by the logistic model
- AUC makes it easy to compare the ROC curve of one model to another



- The **AUC** for the red **ROC** curve is greater than the **AUC** for the blue **RO**C curve
- This means that the Red curve is better
- If the Red ROC curve was generated by say, a Random Forest and the Blue ROC by Logistic Regression
- We can conclude that the Random classifier did a better job in classifying the patients

Digital Vidya

Model Tuning

- **Imbalanced Classes**
  - Imbalanced classes are a common problem in machine learning classification where there are a disproportionate ratio of observations in each class
  - Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection
  - For example, we might have a 2-class (binary) classification problem with 100 instances (rows)
  - A total of 80 instances are labelled with Class-1 and the remaining 20 instances are labelled with Class-2
  - This is an imbalanced dataset and the ratio of Class-1 to Class-2 instances is 80:20 or more concisely 4:1
  - Another example is customer churn datasets, where the vast majority of customers stay with the service (the “No-Churn” class) and a small minority cancel their subscription (the “Churn” class)

Model Tuning

- **Accuracy Paradox**
  - It is the case where the accuracy measures tell the story that we have excellent accuracy (such as 90%), but the accuracy is only reflecting the underlying class distribution
  - It is very common, because classification accuracy is often the first measure we use when evaluating models on our classification problems
  - The reason we get 90% accuracy on an imbalanced data (with 90% of the instances in Class-1) is because our models look at the data and cleverly decide that the best thing to do is to always predict "Class-1" and achieve high accuracy
  - This is best seen when using a simple rule based algorithm. If we print out the rule in the final model we will see that it is very likely predicting one class regardless of the data it is asked to predict

Model Tuning

- **Resampling - Oversample minority class**
  - Oversampling can be defined as adding more copies of the minority class
  - Oversampling can be a good choice when we don't have a ton of data to work with
  - We need to split into test and train sets BEFORE trying oversampling techniques
  - Oversampling before splitting the data can allow the exact same observations to be present in both the test and train sets
  - This can allow our model to simply memorize specific data points and cause overfitting and poor generalization to the test data

```
from sklearn.utils import resample
# Separate input features and target
y = df.Class
X = df.drop('Class', axis=1)
# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=27)
# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)
# separate minority and majority classes
not_fraud = X[X.Class==0]
fraud = X[X.Class==1]
# upsample minority
fraud_upsampled = resample(fraud,      replace=True, # sample with replacement
                           n_samples=len(not_fraud), # match number in majority class
                           random_state=27) # reproducible results
# combine majority and upsampled minority
upsampled = pd.concat([not_fraud, fraud_upsampled])
# check new class counts
upsampled.Class.value_counts()
  1   213245
  0   213245
```

Model Tuning

- **Resampling - Oversample minority class**
  - After resampling we have an equal ratio of data points for each class
  - Let's try logistic regression again with the balanced training data

This improved recall score , but F1 is still way too less

```
# trying logistic regression again with the balanced dataset
y_train = upsampled.Class
X_train = upsampled.drop('Class', axis=1)

upsampled = LogisticRegression(solver='liblinear').fit(X_train, y_train)

upsampled_pred = upsampled.predict(X_test)

# Checking accuracy
accuracy_score(y_test, upsampled_pred)
   0.9807

# f1 score
f1_score(y_test, upsampled_pred)
   0.1437

recall_score(y_test, upsampled_pred)
   0.8712
```

Digital Vidya

Model Tuning

- **Resampling techniques – Undersample majority class**
  - Undersampling can be defined as removing some observations of the majority class
  - Undersampling can be a good choice when we have plenty of data
  - But a drawback is that we are removing information that may be valuable
  - This could lead to underfitting and poor generalization to the test set

```
# still using our separated classes fraud and not_fraud from above

# downsample majority
not_fraud_downsampled = resample(not_fraud,
                                 replace = False, # sample without replacement
                                 n_samples = len(fraud), # match minority n
                                 random_state = 27) # reproducible results

# combine minority and downsampled majority
downsampled = pd.concat([not_fraud_downsampled, fraud])

# checking counts
downsampled.Class.value_counts()
  1    360
  0    360
```

Digital Vidya

Model Tuning

- **Resampling - Undersample majority class**
  - After resampling we have an equal ratio of data points for each class
  - In this case a much smaller quantity of data to train the model on
  - Let's try logistic regression again with the balanced training data

Undersampling underperformed oversampling in this case

```
# trying logistic regression again with the undersampled dataset

y_train = downsampled.Class
X_train = downsampled.drop('Class', axis=1)

undersampled = LogisticRegression(solver='liblinear').fit(X_train, y_train)

undersampled_pred = undersampled.predict(X_test)

# Checking accuracy
accuracy_score(y_test, undersampled_pred)
   0.9758

# f1 score
f1_score(y_test, undersampled_pred)
   0.1171

recall_score(y_test, undersampled_pred)
   0.8636
```

Digital Vidya

Model Tuning

- **Generate synthetic samples**

  - A technique similar to upsampling is to create synthetic samples
  - Here we will use imblearn's SMOTE or Synthetic Minority Oversampling Technique
  - SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model
  - Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data

```
from imblearn.over_sampling import SMOTE

# Separate input features and target
y = df.Class
X = df.drop('Class', axis=1)

# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=27)

sm = SMOTE(random_state=27, ratio=1.0)
X_train, y_train = sm.fit_sample(X_train, y_train)
```

Model Tuning

- **Generate synthetic samples**
  - A technique similar to upsampling is to create synthetic samples
  - Here we will use imblearn's SMOTE or Synthetic Minority Oversampling Technique
  - SMOTE uses a nearest neighbors algorithm to generate new and synthetic data we can use for training our model
  - Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data

```
smote = LogisticRegression(solver='liblinear').fit(X_train, y_train)

smote_pred = smote.predict(X_test)

# Checking accuracy
accuracy_score(y_test, smote_pred)
  0.9858

# f1 score
f1_score(y_test, smote_pred)
  0.1846

recall_score(y_test, smote_pred)
  0.8636
```

Our F1 score is increased and recall is similar to the upsampled model above and for our data here outperforms undersampling

Digital Vidya

Thank You

Digital Vidya