1. Model Building
   1. Feature Selection
   2. Build Logistic Refression Model
   3. Find Optimal Cut off
   4. Build Random Forest Model

**7.4.2 Build the random forest model**

Started to build the Random Forest Model with standard params

Number of trees: 10 with each tree max depth of 4 with each node having max of 10 features. After initializing I had fitted the model with train data

%%time

rf = RandomForestClassifier(n\_estimators=10, max\_depth=4, max\_features=10, random\_state=100, oob\_score=True)  
rf.fit(X\_train\_sm, y\_resampled\_dummies\_df)

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**7.4.3 Get feature importance scores and select important features**

I used the random forest attribute

rf.feature\_importances\_

**Output:**

All the 50 features which was present, the score of importance is displayed in the second column.

A screenshot of a computer

AI-generated content may be incorrect.

Then I selected all the features with importance more than “0.004” which selected 26 features for the model to process.

# Select features with high importance scores  
important\_features = feature\_importance\_df[feature\_importance\_df['Importance'] > 0.004]

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**7.4.4 Train the model with selected features**

Trained the model with new 26 features.

Code and Output

A screenshot of a computer

AI-generated content may be incorrect.

**7.4.5 Generate predictions on the training data**

Run prediction on the training data

y\_train\_pred = rf.predict(X\_train\_important\_feature)

**7.4.6 Check accuracy of the model**

Using the sklearn metrics library calculated the accuracy of the train data.

# Check accuracy of the model  
from sklearn.metrics import confusion\_matrix, accuracy\_score  
  
train\_accuracy = accuracy\_score(y\_resampled\_dummies\_df, y\_train\_pred)  
print("Train Accuracy :", train\_accuracy)

Output:

Train Accuracy : 0.7875894988066826

**7.4.7 Create confusion matrix**

Created a confusion matrix usng the SKlearn package

train\_confusion = confusion\_matrix(y\_resampled\_dummies\_df, y\_train\_pred)  
print("Train Confusion Matrix:")  
print(train\_confusion)

Output

A black background with white text

AI-generated content may be incorrect.

**7.4.8 Create variables for true positive, true negative, false positive and false negative**

TN, FP, FN, TP = train\_confusion.ravel() # true negatives, false positives, false negatives, true positive

**7.4.9 Calculate sensitivity, specificity, precision, recall and F1-score of the model**

Calculated the sensitivity aka recall, specificity, precision and F1 Score using the standard formula

recall = TP / float(TP + FN)  
  
# Calculate the specificity  
print(TN / float(TN+FP))  
  
  
# Calculate Precision  
precision = TP / float(TP + FP)  
  
  
# Calculate F1 Score  
f1\_score = 2 \* (precision \* recall) / (precision + recall)

Output

**A screenshot of a computer

AI-generated content may be incorrect.**

**7.4.10 Check if the model is overfitting training data using cross validation**

Used the sklearn model selection cross validate function to calculate the overfitting, Since in the previous step we calculated the sensitivity aka recall, specificity, precision and F1 Score, used the same scoring technique to say the F1 Score on train and test. As expected the model **is experiencing the Overfitting** as we can see 6 % greater in Training F1 Score and Test F1 Score.

scoring = ['f1', 'accuracy', 'precision', 'recall']  
  
cv\_results = cross\_validate(  
 estimator=rf,  
 X=X\_train\_important\_feature,  
 y=y\_resampled\_dummies\_df,  
 cv=5,  
 scoring=scoring,  
 return\_train\_score=True,  
 n\_jobs=-1  
)  
train\_f1\_mean = np.mean(cv\_results['train\_f1'])  
validation\_f1\_mean = np.mean(cv\_results['test\_f1'])

Output:

Training F1 Score: 0.8078

Test F1 Score: 0.7432

----------------------------------------

**7.5 Hyperparameter Tuning**

**7.5.1 Use grid search to find the best hyperparameter values**

Used the sklearn KFold and GridSearchCV for finding the best hyperparameters,

def use\_grid\_search\_cv(X\_train\_grid\_cv, y\_train\_grid\_cv):  
 folds = KFold(n\_splits = 3, shuffle = True, random\_state = 100)  
 hyper\_params = {  
 'max\_depth': [3, 5],  
 'n\_estimators': [10, 100, 200],  
 'min\_samples\_split': [10, 20, 30],  
 'min\_samples\_leaf': [5, 10],  
}  
 model\_cv = GridSearchCV(estimator = rf,  
 param\_grid = hyper\_params,  
 scoring= 'f1',  
 cv = folds,  
 verbose = 1,  
 return\_train\_score=True)  
  
 model\_cv.fit(X\_train\_grid\_cv, y\_train\_grid\_cv)  
 return model\_cv.cv\_results\_

Output:

Copy pasting only the Mean score of test and train data as that was the metric used to derive the hyper params.

Fitting 3 folds for each of 36 candidates, totalling 108 fits

'mean\_train\_score': array([0.76724949, 0.78625979, 0.7937922 , 0.77018613, 0.7854358 ,

0.78838074, 0.75904531, 0.78083776, 0.7797087 , 0.74370653,

0.77425707, 0.78029202, 0.74370653, 0.77425707, 0.78029202,

0.74560664, 0.77372709, 0.77940961, 0.83256469, 0.86427579,

0.86601614, 0.84022438, 0.84851981, 0.84949217, 0.82792928,

0.83578221, 0.83910987, 0.81568107, 0.83676993, 0.83625567,

0.81568107, 0.83676993, 0.83625567, 0.80196311, 0.82946313,

0.82304696]),

'mean\_test\_score': array([0.70257431, 0.74367229, 0.74235033, 0.72251254, 0.7346105 ,

0.73948253, 0.68391885, 0.73249134, 0.73558367, 0.69725899,

0.73134116, 0.7332476 , 0.69725899, 0.73134116, 0.7332476 ,

0.67213262, 0.7280895 , 0.73769709, 0.73750827, 0.77656375,

0.77919834, 0.74638236, 0.76396303, 0.77375593, 0.7359688 ,

0.75168569, 0.76284918, 0.733822 , 0.76354368, 0.76568572,

0.733822 , 0.76354368, 0.76568572, 0.72990791, 0.75483957,

0.76710221]),

**Plotting Cross Validation Results:**

Plotted the cv results in graphical view to see the changes of the hyper parameters suggesting. As in the output it is evident that taining score is .82 while the test is .73 for max depth of 5, after that the model gets complex.

plt.figure(figsize=(16,6))  
  
plot\_param = 'param\_max\_depth'  
grouped\_results = pd.DataFrame(cv\_results).groupby(plot\_param)[['mean\_test\_score', 'mean\_train\_score']].mean().reset\_index()

Output:

A graph with a line

AI-generated content may be incorrect.

**7.5.2 Build a random forest model based on hyperparameter tuning results**

As per suggestion from the hyper tuning final result used the number of estimators as 100 with max depth 5 minimun samples split of 20 for each node, minimum samples for the leaf node is 5.

rf = RandomForestClassifier(n\_estimators=100, max\_depth=5, min\_samples\_split=20, min\_samples\_leaf=5, random\_state=100, oob\_score=True)

**7.5.3 Make predictions on training data**

# Make predictions on training data  
rf.fit(X\_train\_important\_feature, y\_resampled\_dummies\_df)  
y\_train\_pred = rf.predict(X\_train\_important\_feature)

**7.5.4 Check accuracy of Random Forest Model**

Using the sklearn metrics library calculated the accuracy of the train data.

Output:

Train Accuracy : 0.8281622911694511

**7.5.5 Create confusion matrix**

Created a confusion matrix usng the SKlearn package

Output:

Train Confusion Matrix:

[[312 107]

[ 37 382]]

**7.5.6 Create variables for true positive, true negative, false positive and false**

TN, FP, FN, TP = train\_confusion.ravel() # true negatives, false positives, false negatives, true positive

**7.5.7 Calculate sensitivity, specificity, precision, recall and F1-score of the model**

Calculated the sensitivity aka recall, specificity, precision and F1 Score using the standard formula

recall = TP / float(TP + FN)  
  
# Calculate the specificity  
print(TN / float(TN+FP))  
  
  
# Calculate Precision  
precision = TP / float(TP + FP)  
  
  
# Calculate F1 Score  
f1\_score = 2 \* (precision \* recall) / (precision + recall)

Output:

A screenshot of a computer

AI-generated content may be incorrect.

Rerun the Cross validation on train and test data to see the improvement in the F1 Score

scoring = ['f1', 'accuracy', 'precision', 'recall']  
  
cv\_results = cross\_validate(  
 estimator=rf,  
 X=X\_train\_important\_feature,  
 y=y\_resampled\_dummies\_df,  
 cv=5,  
 scoring=scoring,  
 return\_train\_score=True,  
 n\_jobs=-1  
)  
train\_f1\_mean = np.mean(cv\_results['train\_f1'])  
validation\_f1\_mean = np.mean(cv\_results['test\_f1'])

Output:

F1 Score after hyper tuning parameters

Training F1 Score: 0.8326

Validation F1 Score: 0.7529

----------------------------------------

**Finding the optimal Threshold for increasing the precision:**

During the fine tunning the hyper parameters we noticed that there was a low precision, however the tuning the hyper parameters, so we tried to see if we can increased the threshold for the model to classify as **“Fraud”** with increase in threshold. The output showed that default threshold was working fine as the threshold the model was struggling to identify the “Fraud” case as it lagged the necessary data.

thresholds = np.arange(0.5, 0.9, 0.05) # Check thresholds from 0.50 to 0.85  
best\_threshold = 0.5  
best\_f1 = 0

Output:

A screen shot of a computer

AI-generated content may be incorrect.

**8. Prediction and Model Evaluation**

**8.1 Make predictions over validation data using logistic regression model**

**8.1.1 Select relevant features for validation data and add constant**

Listed the Validation DF and selected only the columns by Recursive Feature Elimanaiton Cross Validation (RFECV)

# Select the relevant features for validation data  
# Select only the columns selected by RFECV  
col\_selected\_rfe = X\_dummies\_validation\_df[col]  
# Add constant to X\_validation  
X\_validation\_sm = sm.add\_constant(col\_selected\_rfe)  
# Check the data  
X\_validation\_sm

Output:

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**8.1.2 Make predictions over validation data**

# Make predictions on the validation data and store it in the variable 'y\_validation\_pred'  
y\_validation\_predict = logsk.predict(X\_validation\_sm)  
# Reshape it into an array  
y\_validation\_predict[:20]

Output:

array([1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0])

**8.1.3 Create DataFrame with actual values and predicted values for validation data**

# Create DataFrame with actual values and predicted values for validation data  
y\_validation\_pred\_final = pd.DataFrame({'fraud':y\_dummies\_validation\_df.values, 'fraud\_prob':y\_validation\_predict})  
# Create new column indicating predicted classifications based on a cutoff value of 0.5  
y\_validation\_pred\_final['fraud\_predicted'] = y\_validation\_pred\_final.fraud\_prob.map(lambda x: 1 if x > 0.6 else 0)  
y\_validation\_pred\_final.head()

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**8.1.4 Make final prediction based on cutoff value**

# Make final predictions on the validation data using the optimal cutoff  
y\_validation\_pred\_final['final\_predicted'] = y\_validation\_pred\_final.fraud\_prob.map( lambda x: 1 if x > 0.6 else 0)  
y\_validation\_pred\_final.head()

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**8.1.5 Check the accuracy of logistic regression model on validation data**

metrics.accuracy\_score(y\_validation\_pred\_final.fraud, y\_validation\_pred\_final.final\_predicted)

Output:

**0.5875**

**8.1.6 Create confusion matrix**

# Create the confusion matrix  
confusion\_validation = metrics.confusion\_matrix(y\_validation\_pred\_final.fraud, y\_validation\_pred\_final.final\_predicted)  
confusion\_validation

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**8.1.7 Create variables for true positive, true negative, false positive and** **false negative**

# Create variables for true positive, true negative, false positive and false negative  
TP = confusion\_validation[1,1] # true positive  
TN = confusion\_validation[0,0] # true negatives  
FP = confusion\_validation[0,1] # false positives  
FN = confusion\_validation[1,0] # false negatives

**8.1.8 Calculate sensitivity, specificity, precision, recall and f1 score of the model**

# Calculate the sensitivity  
sensitivity = TP / float(TP+FN)  
print("sensitivity = ",sensitivity)  
# Calculate the specificity  
specificity = TN / float(TN+FP)  
print("specificity = ",specificity)  
# Calculate Precision  
Precision = TP / float(TP+FP)  
print("Precision = ",Precision)  
# Calculate Recall  
Recall = TP / float(TP+FN)  
print("Recall = ",Recall)  
# Calculate F1 Score

Output:

A black background with white numbers and symbols

AI-generated content may be incorrect.

**8.2 Make predictions over validation data using random forest mode**

**8.2.1 Select the important features and make predictions over validation data**

Used the important feature found in Hyper tunning on the validation / test data and predicted it

# Select the relevant features for validation data  
print(X\_validation\_important\_feature.columns)  
# Make predictions on the validation data  
y\_validation\_pred = rf.predict(X\_validation\_important\_feature)

Output:

**A screen shot of a computer

AI-generated content may be incorrect.**

**8.2.2 Check accuracy of random forest model**

# Check accuracy  
validation\_accuracy = accuracy\_score(y\_dummies\_validation\_df, y\_validation\_pred)  
print("Validation Accuracy :", validation\_accuracy)

Output:

Validation Accuracy : 0.6208333333333333

**8.2.3 Create confusion matrix**

# Create the confusion matrix  
validation\_confusion = confusion\_matrix(y\_dummies\_validation\_df, y\_validation\_pred)  
print("Validation Confusion Matrix:")  
print(validation\_confusion)

Output:

**A black screen with white text

AI-generated content may be incorrect.**

**8.2.4 Create variables for true positive, true negative, false positive and false negative**

# Create variables for true positive, true negative, false positive and false negative  
TP = confusion\_validation[1,1] # true positive  
TN = confusion\_validation[0,0] # true negatives  
FP = confusion\_validation[0,1] # false positives  
FN = confusion\_validation[1,0] # false negatives

**8.2.5 Calculate sensitivity, specificity, precision, recall and f1 score of the model**

# Calculate the sensitivity  
sensitivity = TP / float(TP+FN)  
print("sensitivity = ",sensitivity)  
# Calculate the specificity  
specificity = TN / float(TN+FP)  
print("specificity = ",specificity)  
# Calculate Precision  
Precision = TP / float(TP+FP)  
print("Precision = ",Precision)  
# Calculate Recall  
Recall = TP / float(TP+FN)  
print("Recall = ",Recall)  
# Calculate F1 Score

Output:

**A screenshot of a computer

AI-generated content may be incorrect.**