FeatureEngineering

March 9, 2024

1 Milestone 3: Feature and model Engineering

In this file we have the code in which we first do feature engineering, and then test our model on the new engineered train data, and make predictions about the validation set. We are cheking to see if our feature engineering is increasing the performance metrices.

[53]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import pandas as pd
import seaborn as sns
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from sklearn.metrics import classification_report
```

[54]:
```
df = pd.read_csv("Assign1Data.csv")
df.head()
```

[54]:
```
   Gender Senior Citizen Partner  Tenure Months Phone Service Multiple Lines  \
0    Male             No      No              2           Yes             No
1  Female             No      No              2           Yes             No
2  Female             No      No              8           Yes            Yes
3  Female             No     Yes             28           Yes            Yes
4    Male             No      No             49           Yes            Yes

  Internet Service Online Security Online Backup Device Protection  \
0              DSL             Yes           Yes                No
1      Fiber optic              No            No                No
2      Fiber optic              No            No               Yes
3      Fiber optic              No            No               Yes
4      Fiber optic              No           Yes               Yes

  Tech Support Streaming TV Streaming Movies        Contract  \
0           No           No                No  Month-to-month
```

```
1           No           No               No  Month-to-month
2           No          Yes              Yes  Month-to-month
3          Yes          Yes              Yes  Month-to-month
4           No          Yes              Yes  Month-to-month

  Paperless Billing              Payment Method  Monthly Charges  \
0               Yes                Mailed check            53.85
1               Yes            Electronic check            70.70
2               Yes            Electronic check            99.65
3               Yes            Electronic check           104.80
4               Yes  Bank transfer (automatic)           103.70

   Total Charges  Churn Value
0         108.15            1
1         151.65            1
2         820.50            1
3        3046.05            1
4        5036.30            1
```

1.1 we will apply the following Feature Engineerings:

1. Dropping Gender and Internet service columns as they dont have high correlation with the churn Value. This can be seen in the data visualization.
2. Merging the Streaming TV and Streaming Movies column as "Streaming" as they are both about streaming and have some what same correlation.
3. Merging the Phone Service and multiple lines column as "Lines" as they are both about phone services and have some what same correlation.

[55]:
```
columns_to_drop = ['Gender', 'Internet Service']

columns_to_drop = [col.strip() for col in columns_to_drop]

df.drop(columns_to_drop, axis=1, inplace=True)

df.head()
```

[55]:
```
  Senior Citizen Partner  Tenure Months Phone Service Multiple Lines  \
0             No      No              2           Yes             No
1             No      No              2           Yes             No
2             No      No              8           Yes            Yes
3             No     Yes             28           Yes            Yes
4             No      No             49           Yes            Yes

  Online Security Online Backup Device Protection Tech Support Streaming TV  \
0             Yes           Yes                No           No           No
1              No            No                No           No           No
```

```
2              No            No               Yes           No          Yes
3              No            No               Yes          Yes          Yes
4              No           Yes               Yes           No          Yes

  Streaming Movies        Contract Paperless Billing  \
0               No  Month-to-month               Yes
1               No  Month-to-month               Yes
2              Yes  Month-to-month               Yes
3              Yes  Month-to-month               Yes
4              Yes  Month-to-month               Yes

              Payment Method  Monthly Charges  Total Charges  Churn Value
0               Mailed check            53.85         108.15            1
1           Electronic check            70.70         151.65            1
2           Electronic check            99.65         820.50            1
3           Electronic check           104.80        3046.05            1
4  Bank transfer (automatic)           103.70        5036.30            1
```

[56]:
```
df = df.fillna(df.mean())

X = df.drop('Churn Value', axis=1)
y = df['Churn Value']
```

```
/var/folders/4z/qh15q30n25nckhc10gfcfdwh0000gn/T/ipykernel_42759/1494793626.py:1
: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError.  Select only valid columns before calling the reduction.
  df = df.fillna(df.mean())
```

[57]:
```
# creating a new column 'Streaming' instead of these 2 columns: 'Streaming TV',␣
 ↪'Streaming Movies'
X['Streaming'] = X['Streaming TV'] + X['Streaming Movies']

# Replace values with new labels based on whether the customer has no, either␣
 ↪one, or both Streaming Services
X['Streaming'] = X['Streaming'].replace({'NoNo': 'No Streaming',
                                         'YesNo': 'Only TV',
                                         'NoYes': 'Only Movies',
                                         'YesYes': 'Both',
                                         'No internet serviceNo internet␣
 ↪service': 'No Streaming'})

X = X.drop(['Streaming TV', 'Streaming Movies'], axis=1)

X.head()
```

[57]:
	Senior Citizen	Partner	Tenure Months	Phone Service	Multiple Lines
0	No	No	2	Yes	No
1	No	No	2	Yes	No
2	No	No	8	Yes	Yes
3	No	Yes	28	Yes	Yes
4	No	No	49	Yes	Yes

	Online Security	Online Backup	Device Protection	Tech Support
0	Yes	Yes	No	No
1	No	No	No	No
2	No	No	Yes	No
3	No	No	Yes	Yes
4	No	Yes	Yes	No

	Contract	Paperless Billing	Payment Method
0	Month-to-month	Yes	Mailed check
1	Month-to-month	Yes	Electronic check
2	Month-to-month	Yes	Electronic check
3	Month-to-month	Yes	Electronic check
4	Month-to-month	Yes	Bank transfer (automatic)

	Monthly Charges	Total Charges	Streaming
0	53.85	108.15	No Streaming
1	70.70	151.65	No Streaming
2	99.65	820.50	Both
3	104.80	3046.05	Both
4	103.70	5036.30	Both

[58]:
```
# creating a new column 'Lines' instead of these 2 columns: 'Phone Service',␣
 ↪'Multiple Lines'
X['Lines'] = X['Phone Service'] + X['Multiple Lines']

# Replace values with new labels based on whether the customer has no lines,␣
 ↪single line, or multiple line.
X['Lines'] = X['Lines'].replace({'NoNo phone service': 'No Lines',
                                 'YesNo': 'Single Line',
                                 'YesYes': 'Multiple Lines'})

X = X.drop(['Phone Service', 'Multiple Lines'], axis=1)

X.head()
```

[58]:
	Senior Citizen	Partner	Tenure Months	Online Security	Online Backup
0	No	No	2	Yes	Yes
1	No	No	2	No	No
2	No	No	8	No	No
3	No	Yes	28	No	No

```
4             No      No             49              No           Yes

  Device Protection Tech Support        Contract Paperless Billing  \
0                No           No  Month-to-month               Yes
1                No           No  Month-to-month               Yes
2               Yes           No  Month-to-month               Yes
3               Yes          Yes  Month-to-month               Yes
4               Yes           No  Month-to-month               Yes

              Payment Method  Monthly Charges  Total Charges     Streaming  \
0               Mailed check            53.85         108.15  No Streaming
1           Electronic check            70.70         151.65  No Streaming
2           Electronic check            99.65         820.50          Both
3           Electronic check           104.80        3046.05          Both
4  Bank transfer (automatic)           103.70        5036.30          Both

            Lines
0     Single Line
1     Single Line
2  Multiple Lines
3  Multiple Lines
4  Multiple Lines
```

Next we also use one hot encoding and feature scaling, as one hot encoding is a better alternative then manually converting each categories into integers which we were previously doing and doing some scaling and ensuring that the numeric columns are all scaled.

Below are the cells for One Hot Encoding and Min Max Scaling

[59]:
```
# One hot Encoding the categorical Values
from sklearn.preprocessing import OneHotEncoder

columns_to_encode = ['Senior Citizen', 'Partner', 'Online Security',
                     'Online Backup', 'Device Protection', 'Tech Support',␣
 ↪'Contract',
                     'Paperless Billing', 'Payment Method', 'Streaming',␣
 ↪'Lines']

ohe = OneHotEncoder()

encoded_columns = ohe.fit_transform(X[columns_to_encode])

encoded_df = pd.DataFrame(encoded_columns.toarray(), columns=ohe.
 ↪get_feature_names_out(columns_to_encode))

X.drop(columns_to_encode, axis=1, inplace=True)

X = pd.concat([X, encoded_df], axis=1)
```

[60]:
```
X.head()
```

[60]:

	Tenure Months	Monthly Charges	Total Charges	Senior Citizen_No	\
0	2	53.85	108.15	1.0	
1	2	70.70	151.65	1.0	
2	8	99.65	820.50	1.0	
3	28	104.80	3046.05	1.0	
4	49	103.70	5036.30	1.0	

	Senior Citizen_Yes	Partner_No	Partner_Yes	Online Security_No	\
0	0.0	1.0	0.0	0.0	
1	0.0	1.0	0.0	1.0	
2	0.0	1.0	0.0	1.0	
3	0.0	0.0	1.0	1.0	
4	0.0	1.0	0.0	1.0	

	Online Security_No internet service	Online Security_Yes	…	\
0	0.0	1.0	…	
1	0.0	0.0	…	
2	0.0	0.0	…	
3	0.0	0.0	…	
4	0.0	0.0	…	

	Payment Method_Credit card (automatic)	Payment Method_Electronic check	\
0	0.0	0.0	
1	0.0	1.0	
2	0.0	1.0	
3	0.0	1.0	
4	0.0	0.0	

	Payment Method_Mailed check	Streaming_Both	Streaming_No Streaming	\
0	1.0	0.0	1.0	
1	0.0	0.0	1.0	
2	0.0	1.0	0.0	
3	0.0	1.0	0.0	
4	0.0	1.0	0.0	

	Streaming_Only Movies	Streaming_Only TV	Lines_Multiple Lines	\
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	1.0	
3	0.0	0.0	1.0	
4	0.0	0.0	1.0	

	Lines_No Lines	Lines_Single Line
0	0.0	1.0
1	0.0	1.0

```
2              0.0                0.0
3              0.0                0.0
4              0.0                0.0

[5 rows x 35 columns]
```

[61]:
```
# Scaling the columns with numeric data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X[['Monthly Charges', 'Total Charges', 'Tenure Months']] = scaler.
 ↪fit_transform(X[['Monthly Charges', 'Total Charges', 'Tenure Months']])

X.head()
```

[61]:
```
   Tenure Months  Monthly Charges  Total Charges  Senior Citizen_No  \
0       0.027778         0.354229       0.010310                1.0
1       0.027778         0.521891       0.015330                1.0
2       0.111111         0.809950       0.092511                1.0
3       0.388889         0.861194       0.349325                1.0
4       0.680556         0.850249       0.578987                1.0

   Senior Citizen_Yes  Partner_No  Partner_Yes  Online Security_No  \
0                 0.0         1.0          0.0                 0.0
1                 0.0         1.0          0.0                 1.0
2                 0.0         1.0          0.0                 1.0
3                 0.0         0.0          1.0                 1.0
4                 0.0         1.0          0.0                 1.0

   Online Security_No internet service  Online Security_Yes  …  \
0                                  0.0                 1.0  …
1                                  0.0                 0.0  …
2                                  0.0                 0.0  …
3                                  0.0                 0.0  …
4                                  0.0                 0.0  …

   Payment Method_Credit card (automatic)  Payment Method_Electronic check  \
0                                     0.0                              0.0
1                                     0.0                              1.0
2                                     0.0                              1.0
3                                     0.0                              1.0
4                                     0.0                              0.0

   Payment Method_Mailed check  Streaming_Both  Streaming_No Streaming  \
0                          1.0             0.0                     1.0
1                          0.0             0.0                     1.0
```

```
2                          0.0             1.0                       0.0
3                          0.0             1.0                       0.0
4                          0.0             1.0                       0.0

   Streaming_Only Movies  Streaming_Only TV  Lines_Multiple Lines  \
0                    0.0                0.0                   0.0
1                    0.0                0.0                   0.0
2                    0.0                0.0                   1.0
3                    0.0                0.0                   1.0
4                    0.0                0.0                   1.0

   Lines_No Lines  Lines_Single Line
0             0.0                1.0
1             0.0                1.0
2             0.0                0.0
3             0.0                0.0
4             0.0                0.0

[5 rows x 35 columns]
```

Here we are dividing the dataset into train, test and validation set. This should be noted that the data set which is being split right now is with the new columns which were manually made, and with the numeric values being scaled. Also we are using SMOTE to ensure that the train data is balanced to ensure our models doesnt get biased.

[62]:
```
X_train, X_test_final, y_train, y_test_final = train_test_split(X, y,␣
 ↪test_size=0.2, random_state=42)

# printing for varification
print("data shapes:")
print("X_train shape:", X_train.shape)
print("X_test_final shape:", X_test_final.shape)
print("y_train shape:", y_train.shape)
print("y_test_final shape:", y_test_final.shape)

smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

X_train_val, X_test_validation, y_train_val, y_test_validation =␣
 ↪train_test_split(X_train_balanced, y_train_balanced, test_size=0.2,␣
 ↪random_state=42)
print("X_train_val shape:", X_train_val.shape)
print("X_test_validation shape:", X_test_validation.shape)
print("y_train_val shape:", y_train_val.shape)
print("y_test_validation shape:", y_test_validation.shape)
```

```
data shapes:
X_train shape: (5634, 35)
```

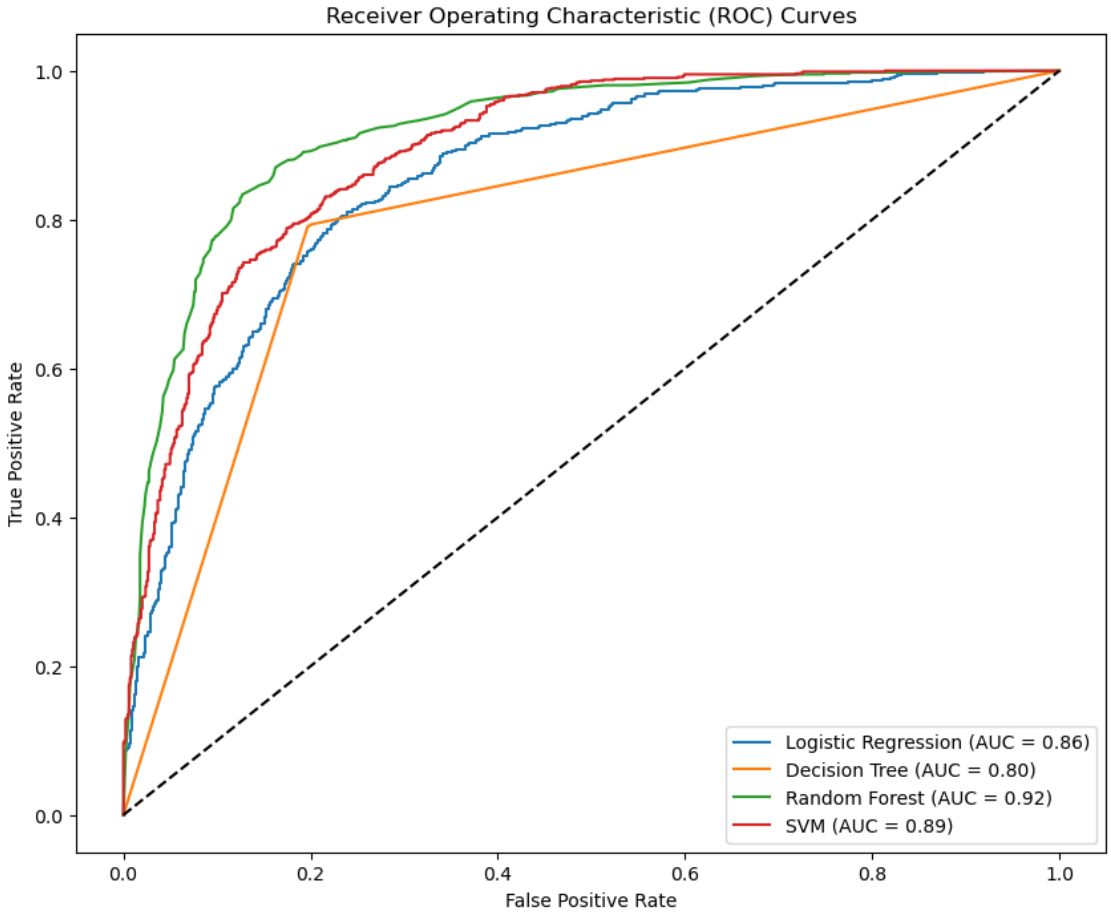
```
X_test_final shape: (1409, 35)
y_train shape: (5634,)
y_test_final shape: (1409,)
X_train_val shape: (6664, 35)
X_test_validation shape: (1666, 35)
y_train_val shape: (6664,)
y_test_validation shape: (1666,)
```

[63]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score
from sklearn.metrics import classification_report

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_val, y_train_val)

dec_tree = DecisionTreeClassifier(random_state=42)
dec_tree.fit(X_train_val, y_train_val)

y_pred_log_reg = log_reg.predict(X_test_validation)
y_pred_dec_tree = dec_tree.predict(X_test_validation)

random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train_val, y_train_val)

svm_clf = SVC(probability=True, random_state=42)
svm_clf.fit(X_train_val, y_train_val)

y_pred_rf = random_forest.predict(X_test_validation)
y_pred_svm = svm_clf.predict(X_test_validation)

print("Logistic Regression")
print(classification_report(y_test_validation, y_pred_log_reg))

print ("")

print("Decision Tree")
print(classification_report(y_test_validation, y_pred_dec_tree))

print("")

print("Random Forest:")
print(classification_report(y_test_validation, y_pred_rf))

print("")

print("SVM:")
print(classification_report(y_test_validation, y_pred_svm))
```

```
Logistic Regression
              precision    recall  f1-score   support

           0       0.81      0.74      0.78       843
           1       0.76      0.82      0.79       823

    accuracy                           0.78      1666
   macro avg       0.78      0.78      0.78      1666
weighted avg       0.78      0.78      0.78      1666


Decision Tree
              precision    recall  f1-score   support

           0       0.80      0.80      0.80       843
           1       0.80      0.79      0.79       823

    accuracy                           0.80      1666
   macro avg       0.80      0.80      0.80      1666
weighted avg       0.80      0.80      0.80      1666


Random Forest:
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       843
           1       0.84      0.86      0.85       823

    accuracy                           0.85      1666
   macro avg       0.85      0.85      0.85      1666
weighted avg       0.85      0.85      0.85      1666


SVM:
              precision    recall  f1-score   support

           0       0.83      0.77      0.80       843
           1       0.78      0.84      0.81       823

    accuracy                           0.80      1666
   macro avg       0.81      0.81      0.80      1666
weighted avg       0.81      0.80      0.80      1666
```

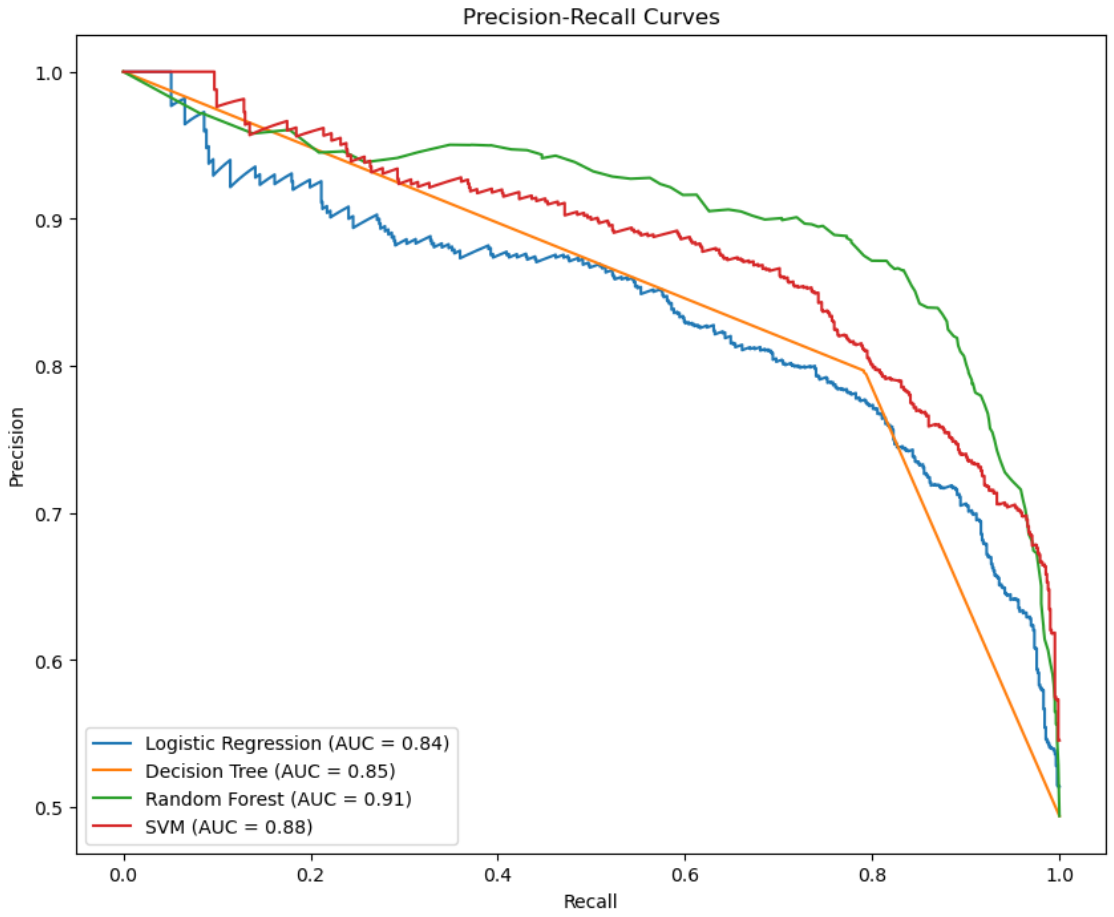
Now we are finally plotting the Auroc curve and the precision recall curve to visually demonstrate how each models performed on the new data which was manually engineered.

[64]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Predicted probabilities for the test set
y_scores_lr = log_reg.predict_proba(X_test_validation)[:, 1]
y_scores_dt = dec_tree.predict_proba(X_test_validation)[:, 1]
y_scores_rf = random_forest.predict_proba(X_test_validation)[:, 1]
y_scores_svm = svm_clf.predict_proba(X_test_validation)[:, 1]

# ROC curve data
fpr_lr, tpr_lr, _ = roc_curve(y_test_validation, y_scores_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

fpr_dt, tpr_dt, _ = roc_curve(y_test_validation, y_scores_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)

fpr_rf, tpr_rf, _ = roc_curve(y_test_validation, y_scores_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

fpr_svm, tpr_svm, _ = roc_curve(y_test_validation, y_scores_svm)
roc_auc_svm = auc(fpr_svm, tpr_svm)

# Plot ROC curve
plt.figure(figsize=(10, 8))  # Adjust size as needed
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {roc_auc_lr:.2f})')
plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {roc_auc_dt:.2f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {roc_auc_svm:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for no-skill classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend(loc="lower right")
plt.show()
```



[65]:
```
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt

# Calculating precision and recall for all models
precision_lr, recall_lr, _ = precision_recall_curve(y_test_validation,␣
 ↪y_scores_lr)
precision_dt, recall_dt, _ = precision_recall_curve(y_test_validation,␣
 ↪y_scores_dt)
precision_rf, recall_rf, _ = precision_recall_curve(y_test_validation,␣
 ↪y_scores_rf)
precision_svm, recall_svm, _ = precision_recall_curve(y_test_validation,␣
 ↪y_scores_svm)

# Calculating AUC for Precision-Recall curve (PR AUC)
pr_auc_lr = auc(recall_lr, precision_lr)
pr_auc_dt = auc(recall_dt, precision_dt)
pr_auc_rf = auc(recall_rf, precision_rf)
pr_auc_svm = auc(recall_svm, precision_svm)
```

```
# Plotting Precision-Recall curves
plt.figure(figsize=(10, 8))
plt.plot(recall_lr, precision_lr, label=f'Logistic Regression (AUC = {pr_auc_lr:
  ↪.2f})')
plt.plot(recall_dt, precision_dt, label=f'Decision Tree (AUC = {pr_auc_dt:.
  ↪2f})')
plt.plot(recall_rf, precision_rf, label=f'Random Forest (AUC = {pr_auc_rf:.
  ↪2f})')
plt.plot(recall_svm, precision_svm, label=f'SVM (AUC = {pr_auc_svm:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curves')
plt.legend(loc="lower left")
plt.show()
```



Now we will compare resource utilization for training each of these models on the new data set

[66]:
```
import pandas as pd
import time
import memory_profiler
from sklearn import linear_model, tree, ensemble, svm, naive_bayes

# Functions for each model
def lr_go(train_ftrs, train_tgt):
    linreg = linear_model.LinearRegression()
    return linreg.fit(train_ftrs, train_tgt)

def rf_go(train_ftrs, train_tgt):
    rforest = ensemble.RandomForestClassifier(random_state=42)
    return rforest.fit(train_ftrs, train_tgt)

def dt_go(train_ftrs, train_tgt):
    dtree = tree.DecisionTreeClassifier(random_state=42)
    return dtree.fit(train_ftrs, train_tgt)

def svm_go(train_ftrs, train_tgt):
    svc = svm.SVC(probability=True, random_state=42)
    return svc.fit(train_ftrs, train_tgt)

def nb_go(train_ftrs, train_tgt):
    nb = naive_bayes.GaussianNB()
    return nb.fit(train_ftrs, train_tgt)

# Function to measure time and memory
def measure_model(model_func, train_ftrs, train_tgt):
    start_time = time.time()
    mem_usage_start = memory_profiler.memory_usage(max_usage=True)
    model = model_func(train_ftrs, train_tgt)
    mem_usage_end = memory_profiler.memory_usage(max_usage=True)
    end_time = time.time()

    print(f"{model_func.__name__}:")
    print(f" Time: {end_time - start_time:.4f} seconds")
    print(f" Memory: {mem_usage_end - mem_usage_start:.4f} MiB\n")

# Assuming `train_ftrs` and `train_tgt` are your features and target variables␣
 ↪for training
# You would load your dataset and set these variables accordingly

# Measure each model
for model_func in [lr_go, rf_go, dt_go, svm_go, nb_go]:
    measure_model(model_func, X_train_val, y_train_val)
```

```
lr_go:
 Time: 0.2124 seconds
```

```
 Memory: 0.2773 MiB

rf_go:
 Time: 0.8873 seconds
 Memory: 17.5547 MiB

dt_go:
 Time: 0.2534 seconds
 Memory: 0.7734 MiB

svm_go:
 Time: 8.7463 seconds
 Memory: 1.7070 MiB

nb_go:
 Time: 0.2127 seconds
 Memory: 0.8086 MiB
```

Random Forest came to be the best one so now lets do hyperparameter tuning with Random Forest

[67]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [200, 250, 300],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [2, 10],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 6],
    'bootstrap': [True, False],
    'criterion': ['entropy', 'gini']

}

rf = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3,␣
 ↪n_jobs=-1, verbose=2)

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

print(best_params)
```

```
print(best_estimator)
```

```
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
{'bootstrap': True, 'criterion': 'entropy', 'max_depth': 10, 'max_features':
'auto', 'min_samples_leaf': 6, 'min_samples_split': 2, 'n_estimators': 200}
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_leaf=6,
                       n_estimators=200, random_state=42)
```

[68]:
```
from sklearn.ensemble import RandomForestClassifier

best_params = {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 10,␣
 ↪'max_features': 'auto', 'min_samples_leaf': 6, 'min_samples_split': 2,␣
 ↪'n_estimators': 200}


random_forest_best = RandomForestClassifier(**best_params, random_state=42)

random_forest_best.fit(X_train_val, y_train_val)

y_pred_rd = random_forest_best.predict(X_test_validation)

print((classification_report(y_test_validation, y_pred_rd)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.78      0.81       843
           1       0.79      0.86      0.82       823

    accuracy                           0.82      1666
   macro avg       0.82      0.82      0.82      1666
weighted avg       0.82      0.82      0.82      1666
```

[69]:
```
y_test_pred_bestrd = random_forest_best.predict(X_test_final)

print((classification_report(y_test_final, y_test_pred_bestrd)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.76      0.82      1009
           1       0.56      0.77      0.65       400

    accuracy                           0.76      1409
   macro avg       0.72      0.76      0.73      1409
weighted avg       0.80      0.76      0.77      1409
```

Above is the performance matrix of tuned random forest on the unseen data, the test set. Not very good, but not BAD!