Week4_Joythi_sanam

February 22, 2024

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
import numpy as np
from scikitplot.estimators import plot_feature_importances
```

[2]:
```
df = pd.read_csv("cleaned_churn_data.csv")

df.tail(15)
```

[2]:
```
      tenure  PhoneService  MonthlyCharges  TotalCharges  Churn  \
7028      68             1           64.10       4326.25      0
7029       6             0           44.40        263.05      0
7030       2             1           20.05         39.25      0
7031      55             1           60.00       3316.10      0
7032       1             1           75.75         75.75      1
7033      38             1           69.50       2625.25      0
7034      67             1          102.95       6886.25      1
7035      19             1           78.70       1495.10      0
7036      12             0           60.65        743.30      0
7037      72             1           21.15       1419.40      0
7038      24             1           84.80       1990.50      0
7039      72             1          103.20       7362.90      0
7040      11             0           29.60        346.45      0
7041       4             1           74.40        306.60      1
7042      66             1          105.65       6844.50      0

      TotalCharges_to_MonthlyCharges_ratio  \
7028                              67.492200
7029                               5.924550
7030                               1.957606
7031                              55.268333
7032                               1.000000
```

```
7033                               37.773381
7034                               66.889267
7035                               18.997459
7036                               12.255565
7037                               67.111111
7038                               23.472877
7039                               71.345930
7040                               11.704392
7041                                4.120968
7042                               64.784666

      PaymentMethod_Bank transfer (automatic)  \
7028                                        1
7029                                        0
7030                                        0
7031                                        0
7032                                        0
7033                                        0
7034                                        0
7035                                        1
7036                                        0
7037                                        1
7038                                        0
7039                                        0
7040                                        0
7041                                        0
7042                                        1

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
7028                                      0                               1
7029                                      0                               0
7030                                      0                               1
7031                                      1                               1
7032                                      0                               0
7033                                      1                               1
7034                                      1                               1
7035                                      0                               1
7036                                      0                               0
7037                                      0                               1
7038                                      0                               1
7039                                      1                               1
7040                                      0                               0
7041                                      0                               1
7042                                      0                               1

      PaymentMethod_Mailed check  Contract_Month-to-month  Contract_One year  \
7028                           0                        1                  0
```

```
7029                              0                          0                    0
7030                              1                          0                    0
7031                              0                          1                    1
7032                              0                          0                    0
7033                              0                          0                    0
7034                              0                          0                    0
7035                              0                          0                    0
7036                              0                          1                    1
7037                              0                          1                    0
7038                              1                          1                    1
7039                              0                          1                    1
7040                              0                          0                    0
7041                              1                          0                    0
7042                              0                          1                    0

      Contract_Two year
7028                  1
7029                  0
7030                  0
7031                  0
7032                  0
7033                  0
7034                  0
7035                  0
7036                  0
7037                  1
7038                  0
7039                  0
7040                  0
7041                  0
7042                  1
```

0.1 Split data into features and targets

[3]:
```
X = df.drop('Churn', axis=1)
y = df['Churn']
```

0.2 Split into training and test sets

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.3 Check for infiniy values error in the dataset

[5]:
```
print("Infinity values in X_train:", np.any(np.isinf(X_train)))
print("NaN values in X_train:", np.any(np.isnan(X_train)))
```

```
Infinity values in X_train: False
NaN values in X_train: False
```

0.4 Decision tree

[6]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)

print("Decision Tree:")
print("Training accuracy:", dt_model.score(X_train, y_train))
print("Testing accuracy:", dt_model.score(X_test, y_test))
```

```
Decision Tree:
Training accuracy: 0.9943201987930422
Testing accuracy: 0.7210787792760823
```

The significant difference between the training accuracy (0.9943) and testing accuracy (0.7232) suggests that the decision tree model (dt_model) may be overfitting the training data.

0.5 Bayesian search for DT HPP tuning

[7]:
```
param_grid = {'max_depth': [2, 3, 5, 7, 10]}
dt_model = DecisionTreeClassifier()
grid_search = GridSearchCV(dt_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

[7]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [2, 3, 5, 7, 10]})
```

[8]:
```
best_max_depth = grid_search.best_params_['max_depth']

print(best_max_depth)
```

```
3
```

We found the best max_depth as 3

[9]:
```
dt_tuned = DecisionTreeClassifier(max_depth=best_max_depth)
dt_tuned.fit(X_train, y_train)

print("Tuned Decision Tree:")
print("Training accuracy:", dt_tuned.score(X_train, y_train))
print("Testing accuracy:", dt_tuned.score(X_test, y_test))
```

```
Tuned Decision Tree:
Training accuracy: 0.7864749733759319
Testing accuracy: 0.78708303761533
```

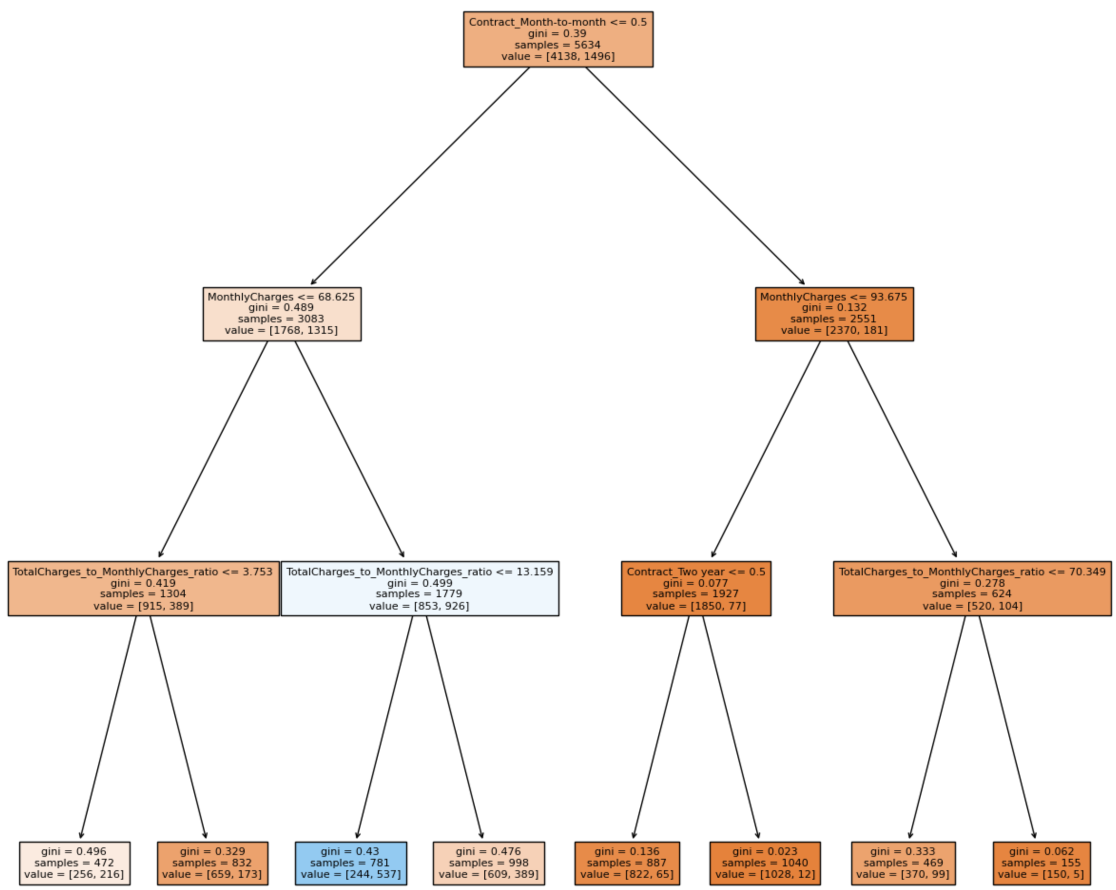
The smaller difference between the training and testing accuracies indicates that the model is now better at generalizing to new, unseen data. This suggests that the regularization imposed by setting the maximum depth to 3 has helped in reducing overfitting.

[10]:
```python
dt_tuned.get_depth()
```

[10]: 3

0.6 Visualize tuned decision tree

[11]:
```python
plt.figure(figsize=(15, 15))
_ = plot_tree(dt_tuned, fontsize=8, feature_names=X.columns, filled=True)
plt.show()
```



The color gradient represents,

Impurity (Gini impurity or entropy)

Nodes with higher impurity are represented by warmer colors such as orange, while nodes with lower impurity are represented by cooler colors (blue). This helps visualize how well the tree is able to split the data based on the features.

0.7 Random Forest

[12]:
```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

[12]:
```
RandomForestClassifier(random_state=42)
```

0.8 Display model metrics

[13]:
```
print("Random Forest:")
print("Training accuracy:", rf_model.score(X_train, y_train))
print("Testing accuracy:", rf_model.score(X_test, y_test))
```

```
Random Forest:
Training accuracy: 0.9941427050053249
Testing accuracy: 0.7842441447835344
```

Clearly there's overfitting due to the large differences between the training and testing accuracies

0.9 Hyperparameter tuning

[14]:
```
param_grid = {'max_depth': [2, 5, 10]}

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

[14]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [2, 5, 10]})
```

[15]:
```
best_params = grid_search.best_params_
print("Best parameters:", best_params)
```

```
Best parameters: {'max_depth': 5}
```

[16]:
```
rf_best = RandomForestClassifier(random_state=42, **best_params)
rf_best.fit(X_train, y_train)
```

[16]:
```
RandomForestClassifier(max_depth=5, random_state=42)
```

[17]:
```
print("Tuned Random Forest:")
print("Training accuracy:", rf_best.score(X_train, y_train))
print("Testing accuracy:", rf_best.score(X_test, y_test))
```

```
Tuned Random Forest:
Training accuracy: 0.7994320198793042
Testing accuracy: 0.794180269694819
```

Now the overfitting hazard has been dealt with

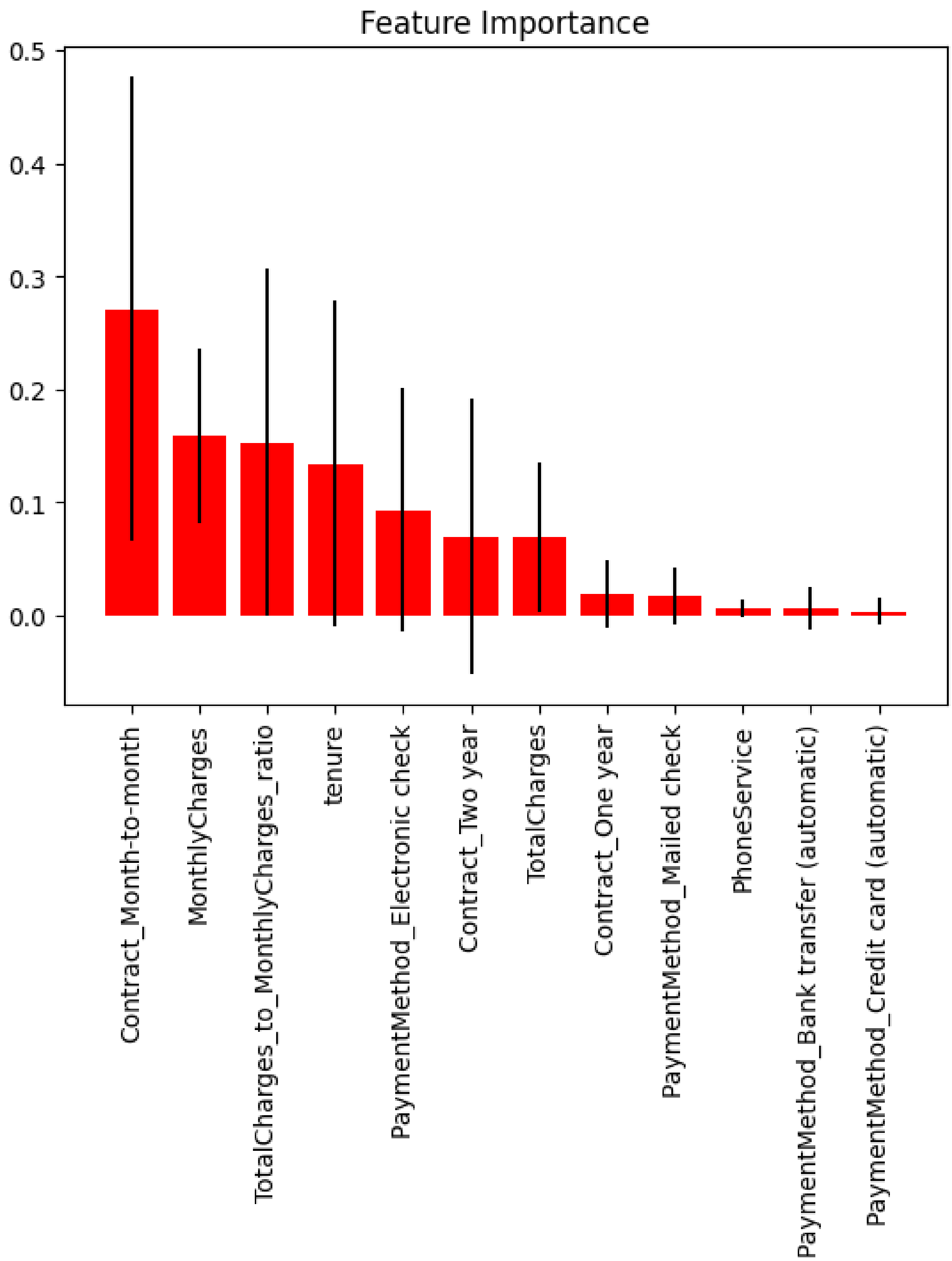
0.10 Feature selection

[18]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr()[['Churn']], annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title("Correlations between features and Churn")
plt.show()
```

0.11 Featured importances fro Tuned RF

[19]:
```
plt.figure(figsize=(10, 6))
plot_feature_importances(rf_best, feature_names=X.columns, x_tick_rotation=90)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



[20]:

```
less_important_features = ['PaymentMethod_Electronic check', 'TotalCharges',␣
↪'Contract_Two year', 'Contract_One year', 'PaymentMethod_Mailed check',␣
↪'PaymentMethod_Credit card (automatic)', 'PaymentMethod_Bank transfer␣
↪(automatic)', 'PhoneService']
```

```
X_train_filtered = X_train.drop(less_important_features, axis=1)
X_test_filtered = X_test.drop(less_important_features, axis=1)
```

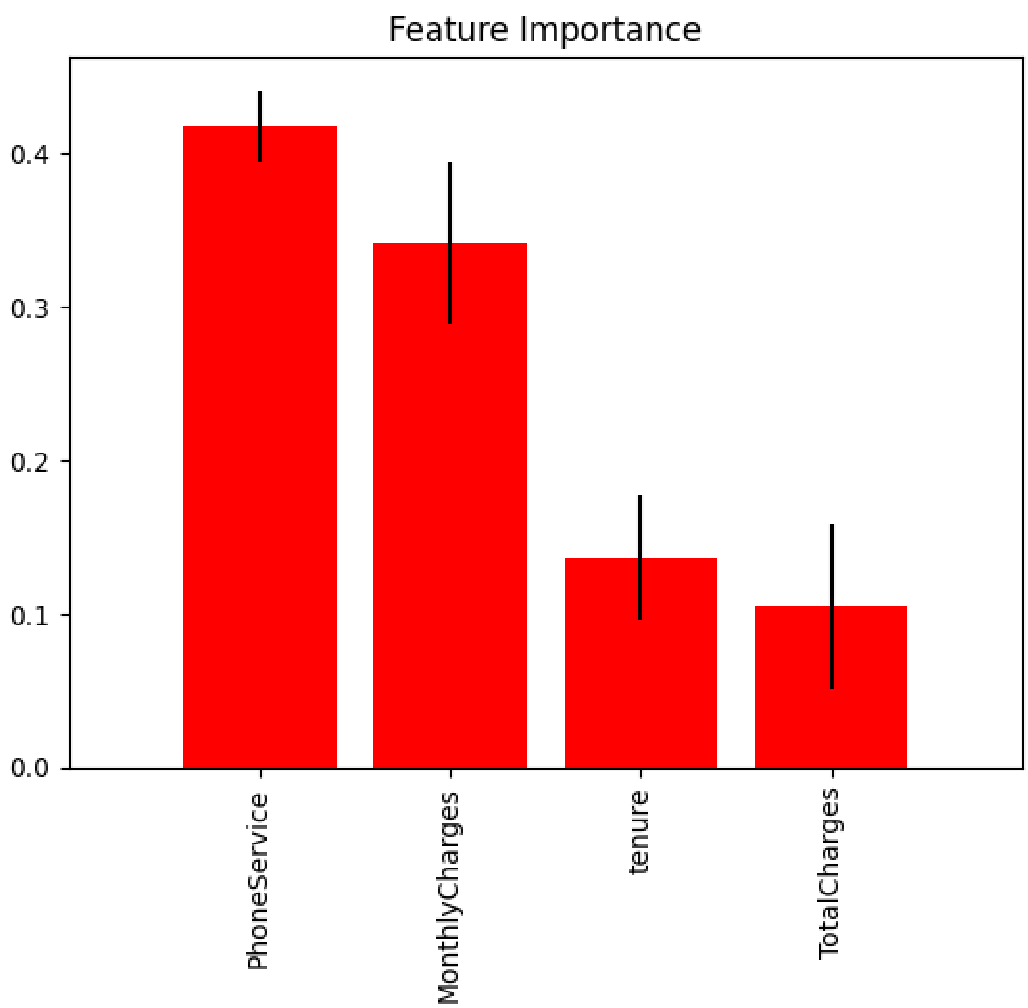
0.12 Fit and Visualize RFC

[22]:
```
rf_filtered = RandomForestClassifier(n_estimators=100, random_state=42)
rf_filtered.fit(X_train_filtered, y_train)
```

[22]:
```
RandomForestClassifier(random_state=42)
```

[23]:
```
plt.figure(figsize=(10, 6))
plot_feature_importances(rf_filtered, feature_names=X.columns,␣
 ↪x_tick_rotation=90)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



```
[27]: def evaluate_model(model, X, y_true):
    y_pred = model.predict(X)
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall:␣
 ↪{recall:.4f}, F1 Score: {f1:.4f}")

print("Random Forest with remaining selected features:")
evaluate_model(rf_filtered, X_test_filtered, y_test)
```

```
Random Forest with remaining selected features:
Accuracy: 0.7587, Precision: 0.5541, Recall: 0.4531, F1 Score: 0.4985
```

A modest reduction in accuracy after removal of the less important features

0.13 Summary

Data Loading and Splitting

Loaded the churn prediction dataset. Created feature (X) and target (y) variables. Split the data into training and testing sets.

Decision Tree

Built a baseline decision tree without hyperparameter tuning. Evaluated the decision tree on training and testing data. Utilized GridSearchCV for hyperparameter tuning, specifically adjusting the max_depth parameter. Created a tuned decision tree based on the best hyperparameters. Visualized the tuned decision tree.

Random Forest

Built a baseline random forest without hyperparameter tuning. Utilized BGridSearchCV for hyperparameter tuning, focusing on the max_depth parameter. Created a tuned random forest (rf_tuned) based on the best hyperparameters.

Feature Selection

Explored feature correlations using a heatmap. Plotted feature importances for the tuned random forest.

Further Feature Selection and Model Evaluation

Identified and dropped less important features. Trained a random forest model (rf_filtered) on the filtered feature set. Visualized feature importances after removing less important features. Evaluated the random forest model after feature selection using metrics such as accuracy, precision, recall, and F1 score.