Homework 3

March 7, 2024

Aaron Vo

Step 1: Read in Titanic.csv and observe a few samples, some features are categorical, and others are numerical. If some features are missing, fill them in using the average of the same feature of other samples. Take a random 80% samples for training and the rest 20% for test.

[1]:
```
# Import the packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
```

[2]:
```
#Read the Titanic CSV
df = pd.read_csv("Titanic.csv")
df.head()
```

[2]:
```
   Unnamed: 0 pclass  survived                             name     sex  \
0           1    1st         1     Allen, Miss. Elisabeth Walton  female
1           2    1st         1   Allison, Master. Hudson Trevor    male
2           3    1st         0     Allison, Miss. Helen Loraine  female
3           4    1st         0  Allison, Mr. Hudson Joshua Crei    male
4           5    1st         0  Allison, Mrs. Hudson J C (Bessi  female

       age  sibsp  parch  ticket        fare    cabin     embarked boat  \
0  29.0000      0      0   24160  211.337494       B5  Southampton    2
1   0.9167      1      2  113781  151.550003  C22 C26  Southampton   11
2   2.0000      1      2  113781  151.550003  C22 C26  Southampton  NaN
3  30.0000      1      2  113781  151.550003  C22 C26  Southampton  NaN
4  25.0000      1      2  113781  151.550003  C22 C26  Southampton  NaN

    body                        home.dest
0    NaN                     St Louis, MO
1    NaN  Montreal, PQ / Chesterville, ON
2    NaN  Montreal, PQ / Chesterville, ON
3  135.0  Montreal, PQ / Chesterville, ON
4    NaN  Montreal, PQ / Chesterville, ON
```

[3]:
```
def getNaNCount(df):
    nan_count = df.isna().sum()
    print(nan_count)
```

[4]:
```
age_mean = df['age'].mean()
df['age'].fillna(age_mean, inplace=True)
getNaNCount(df)
```

```
Unnamed: 0       0
pclass           0
survived         0
name             0
sex              0
age              0
sibsp            0
parch            0
ticket           0
fare             1
cabin         1014
embarked         2
boat           823
body          1188
home.dest      564
dtype: int64
```

Step 2: Fit a decision tree model using independent variables `pclass + sex + age + sibsp` and dependent variable `survived`. Plot the full tree. Make sure `survived` is a qualitative variable taking 1 (yes) or 0 (no) in your code. You may see a tree similar to this one (the actual structure and size of your tree can be different):

[5]:
```
def getValueCounts(df, column):
    print(df[column].value_counts())
```

[6]:
```
getValueCounts(df, 'sex')
```

```
male      843
female    466
Name: sex, dtype: int64
```

[7]:
```
df['sex'] = df['sex'].replace({'female': 0, 'male': 1})
getValueCounts(df, 'sex')
```

```
1    843
0    466
Name: sex, dtype: int64
```

[8]:
```
getValueCounts(df, 'pclass')
```

```
3rd    709
1st    323
```

```
2nd    277
Name: pclass, dtype: int64
```

[9]:
```
df['pclass'] = df['pclass'].replace({'1st':1, '2nd':2, '3rd': 3})
getValueCounts(df, 'pclass')
```

```
3    709
1    323
2    277
Name: pclass, dtype: int64
```

[10]:
```
data = df[['pclass', 'sex', 'age', 'sibsp', 'survived']]
data.head()
```

[10]:

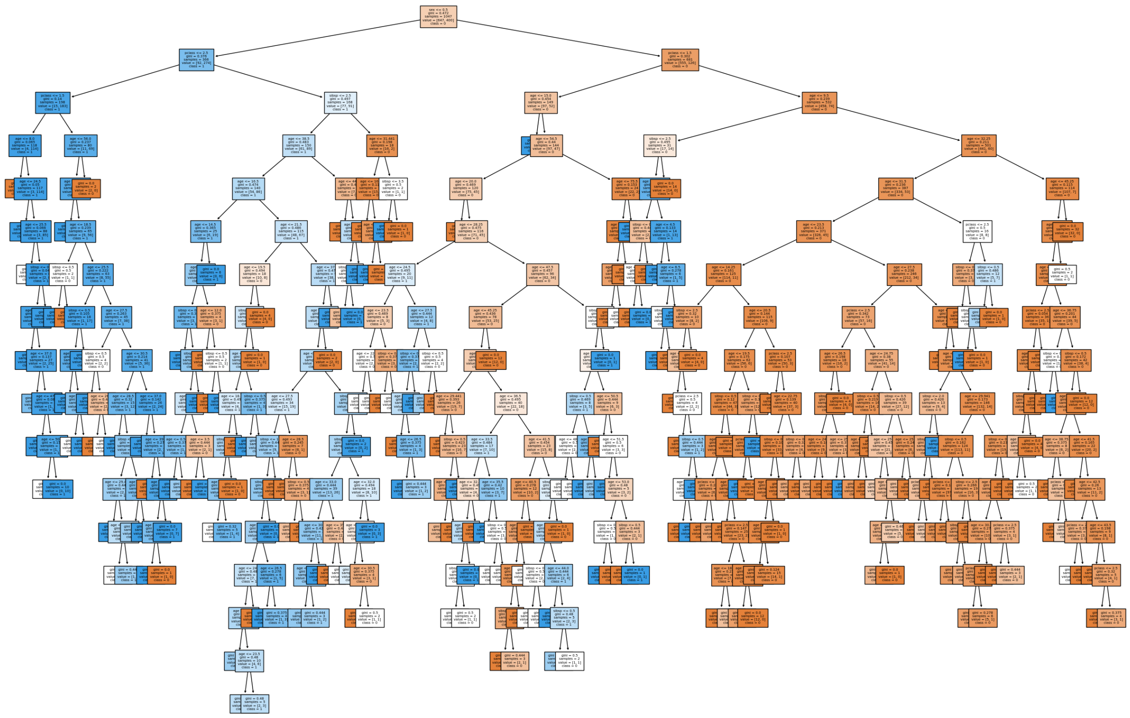
	pclass	sex	age	sibsp	survived
0	1	0	29.0000	0	1
1	1	1	0.9167	1	1
2	1	0	2.0000	1	0
3	1	1	30.0000	1	0
4	1	0	25.0000	1	0

[11]:
```
X_train, X_test, y_train, y_test = train_test_split(data[['pclass', 'sex',␣
↪'age', 'sibsp']],
                                                    data['survived'],
                                                    test_size = 0.2,
                                                    random_state = 5)
```

[12]:
```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

[12]:
```
DecisionTreeClassifier()
```

[13]:
```
# Plot the decision tree
plt.figure(figsize = (30, 20))
plot_tree(clf, filled = True, feature_names = X_train.columns, class_names =␣
↪list(map(str, clf.classes_)), fontsize = 5)
plt.show()
```



Step 3: Use the GridSearchCV() function to find the best parameter max_leaf_nodes to prune the tree. Plot the pruned tree which shall be smaller than the tree you obtained in Step 2.

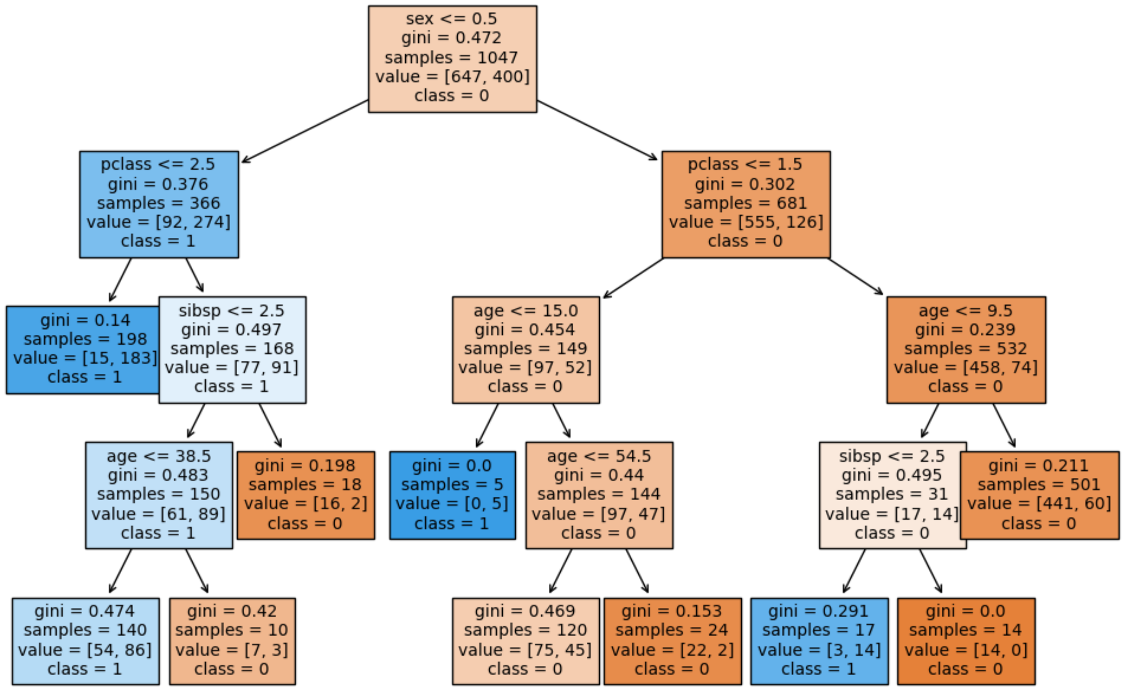
[14]:
```python
param_grid = {'max_leaf_nodes': [5, 10, 15, 20, 25]}
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

best_classifier = grid_search.best_estimator_

plt.figure(figsize = (12, 8))
plot_tree(best_classifier, filled = True, feature_names = X_train.columns, class_names=list(map(str, best_classifier.classes_)), fontsize=10)
plt.show()
```

```
Best Parameters: {'max_leaf_nodes': 10}
```



Step 4: For the pruned tree, report its accuracy on the test set for the following:

```
percent survivors correctly predicted (on test set)
percent fatalities correctly predicted (on test set)
```

[15]:
```python
def printConfusionMatrix(y_test,y_pred):
    conf_matrix = confusion_matrix(y_test, y_pred)

    percent_survivors_correct = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1])
    percent_fatalities_correct = conf_matrix[1, 1] / (conf_matrix[1, 0] + conf_matrix[1, 1])

    print(f"Percent Survivors Correctly Predicted: {percent_survivors_correct:.2%}")
    print(f"Percent Fatalities Correctly Predicted: {percent_fatalities_correct:.2%}")
```

[16]:
```python
y_pred = best_classifier.predict(X_test)

printConfusionMatrix(y_test, y_pred)
```

```
Percent Survivors Correctly Predicted: 82.10%
Percent Fatalities Correctly Predicted: 70.00%
```

[17]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       162
           1       0.71      0.70      0.70       100

    accuracy                           0.77       262
   macro avg       0.76      0.76      0.76       262
weighted avg       0.77      0.77      0.77       262
```

Step 5: Use the RandomForestClassifier() function to train a random forest using the value of max_leaf_nodes you found in Step 3. You can set n_estimators as 50. Report the accuracy of random forest on the test set for the following:

```
percent survivors correctly predicted (on test set)
percent fatalities correctly predicted (on test set)
```

Check whether there is improvement as compared to a single tree obtained in Step 4.

[18]:
```
rf_clf = RandomForestClassifier(max_leaf_nodes = 10)
rf_clf.fit(X_train, y_train)
rd_predict = rf_clf.predict(X_test)
printConfusionMatrix(y_test, rd_predict)
```

```
Percent Survivors Correctly Predicted: 80.25%
Percent Fatalities Correctly Predicted: 68.00%
```

[19]:
```
print(classification_report(y_test, rd_predict))
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80       162
           1       0.68      0.68      0.68       100

    accuracy                           0.76       262
   macro avg       0.74      0.74      0.74       262
weighted avg       0.76      0.76      0.76       262
```

Based on the results from the `RandomForestClassifer`, we see that there is minimal difference in the accuracy, as we see that the it is only 2% more accurate the the `DecisionTreeClassifier`.

Regarding the percentage on the survivors and fatalities correctly predicted, we see that the `RandomForestClassifier` was predicted less correctly than the `DecisionTreeClassifier`