Assignment6

July 10, 2024

1 Assignment is below at the end

- https://scikit-learn.org/stable/modules/tree.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html

[1]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (20, 6)
plt.rcParams['font.size'] = 14
import pandas as pd
```

[2]:
```
df = pd.read_csv('adult.data', index_col=False)
```

[3]:
```
golden = pd.read_csv('adult.test', index_col=False)
```

[4]:
```
golden.head()
```

[4]:
```
   age  workclass  fnlwgt     education  education-num      marital-status  \
0   25    Private  226802          11th              7       Never-married
1   38    Private   89814       HS-grad              9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm             12  Married-civ-spouse
3   44    Private  160323  Some-college             10  Married-civ-spouse
4   18          ?  103497  Some-college             10       Never-married

          occupation relationship   race     sex  capital-gain  \
0  Machine-op-inspct    Own-child  Black    Male             0
1    Farming-fishing      Husband  White    Male             0
2    Protective-serv      Husband  White    Male             0
3  Machine-op-inspct      Husband  Black    Male          7688
4                  ?    Own-child  White  Female             0

   capital-loss  hours-per-week  native-country  salary
0             0              40   United-States  <=50K.
1             0              50   United-States  <=50K.
2             0              40   United-States   >50K.
3             0              40   United-States   >50K.
```

```
4             0              30   United-States   <=50K.
```

[5]:
```
df.head()
```

[5]:
```
   age          workclass  fnlwgt   education  education-num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
3   53            Private  234721        11th              7
4   28            Private  338409   Bachelors             13

        marital-status          occupation    relationship    race      sex  \
0        Never-married        Adm-clerical   Not-in-family   White     Male
1   Married-civ-spouse     Exec-managerial         Husband   White     Male
2             Divorced   Handlers-cleaners   Not-in-family   White     Male
3   Married-civ-spouse   Handlers-cleaners         Husband   Black     Male
4   Married-civ-spouse      Prof-specialty            Wife   Black   Female

   capital-gain  capital-loss  hours-per-week  native-country  salary
0          2174             0              40   United-States   <=50K
1             0             0              13   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40            Cuba   <=50K
```

[6]:
```
df.columns
```

[6]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
       'salary'],
      dtype='object')
```

[7]:
```
from sklearn import preprocessing
```

[8]:
```
# Columns we want to transform
transform_columns = ['sex']

#Columns we can't use because non-numerical
non_num_columns = ['workclass', 'education', 'marital-status',
                   'occupation', 'relationship', 'race', 'sex',
                   'native-country']
```

1.1 First let's try using pandas.get_dummies() to transform columns

[9]:
```python
dummies = pd.get_dummies(df[transform_columns])
dummies
```

[9]:
```
        sex_ Female  sex_ Male
0             False       True
1             False       True
2             False       True
3             False       True
4              True      False
…                 …          …
32556          True      False
32557         False       True
32558          True      False
32559         False       True
32560          True      False

[32561 rows x 2 columns]
```

[10]:
```python
dummies.shape
```

[10]: (32561, 2)

1.2 sklearn has a similar process for OneHot Encoding features

[11]:
```python
onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist",␣
 ↪sparse=False)
onehot.fit(df[transform_columns])
```

```
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages/sklearn/preprocessing/_encoders.py:972: FutureWarning: `sparse` was
renamed to `sparse_output` in version 1.2 and will be removed in 1.4.
`sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

[11]:
```
OneHotEncoder(handle_unknown='infrequent_if_exist', sparse=False,
              sparse_output=False)
```

[12]:
```python
onehot.categories_
```

[12]: [array([' Female', ' Male'], dtype=object)]

[13]:
```python
sex = onehot.transform(df[transform_columns])
sex
```

[13]:
```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       …,
       [1., 0.],
       [0., 1.],
       [1., 0.]])
```

[14]:
```
sex.shape
```

[14]:
```
(32561, 2)
```

1.3 In addition to OneHot encoding there is Ordinal Encoding

[15]:
```
enc = preprocessing.OrdinalEncoder()
enc.fit(df[["salary"]])
salary = enc.transform(df[["salary"]])
salary
```

[15]:
```
array([[0.],
       [0.],
       [0.],
       …,
       [0.],
       [0.],
       [1.]])
```

[16]:
```
enc.categories_[0]
```

[16]:
```
array([' <=50K', ' >50K'], dtype=object)
```

[61]:
```
x = df.copy()

# transformed = pd.get_dummies(df[transform_columns])


#onehot = preprocessing.OneHotEncoder(handle_unknown="infrequent_if_exist",␣
↪sparse=False).fit(df[transform_columns])

enc = preprocessing.OrdinalEncoder()
enc.fit(df[["salary"]])

x = df.copy()
x = pd.concat([x.drop(non_num_columns, axis=1),
               pd.get_dummies(df[transform_columns])], axis=1)
```

```python
#transformed = onehot.transform(df[transform_columns])
#new_cols = list(onehot.categories_[0].flatten())
#df_trans = pd.DataFrame(transformed, columns=new_cols)


#x = pd.concat(
#    [
#        x.drop(non_num_columns, axis=1),
#        df_trans
#    ],
#    axis=1,)


x["salary"] = enc.transform(df[["salary"]])
```

[62]:

```python
x.head()
```

[62]:

```
   age  fnlwgt  education-num  capital-gain  capital-loss  hours-per-week  \
0   39   77516             13          2174             0              40
1   50   83311             13             0             0              13
2   38  215646              9             0             0              40
3   53  234721              7             0             0              40
4   28  338409             13             0             0              40

   salary  sex_ Female  sex_ Male
0     0.0        False       True
1     0.0        False       True
2     0.0        False       True
3     0.0        False       True
4     0.0         True      False
```

[63]:

```python
xt = golden.copy()

transformed = onehot.transform(xt[transform_columns])
new_cols = list(onehot.categories_[0].flatten())
#df_trans = pd.DataFrame(transformed, columns=new_cols)

#x = pd.concat(
#    [
#        xt.drop(non_num_columns, axis=1),
#        df_trans
#    ],
#    axis=1,)
# xt["salary"] = enc.fit_transform(golden[["salary"]])
xt = pd.concat([xt.drop(non_num_columns, axis=1),
                pd.get_dummies(xt[transform_columns])], axis=1)
xt["salary"] = enc.fit_transform(xt[["salary"]])
```

[64]:
```
xt.salary.value_counts()
```

[64]:
```
salary
0.0    12435
1.0     3846
Name: count, dtype: int64
```

[65]:
```
enc.categories_
```

[65]:
```
[array([' <=50K.', ' >50K.'], dtype=object)]
```

[66]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Choose the model of your preference: DecisionTree or RandomForest

[67]:
```
model = RandomForestClassifier(criterion='entropy')
```

[68]:
```
model = DecisionTreeClassifier(criterion='entropy', max_depth=None)
```

[69]:
```
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

[69]:
```
DecisionTreeClassifier(criterion='entropy')
```

[70]:
```
model.tree_.node_count
```

[70]:
```
8317
```

[71]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.
↪feature_importances_))
```

[71]:
```
[('age', 0.32366222896975533),
 ('education-num', 0.15861240773791468),
 ('capital-gain', 0.2284592742844192),
 ('capital-loss', 0.07843149812577388),
 ('hours-per-week', 0.15529640153538432),
 ('sex_ Female', 0.021232345235427594),
 ('sex_ Male', 0.03430584411132491)]
```

[72]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.
↪feature_importances_))
```

[72]:
```
[('age', 0.32366222896975533),
 ('education-num', 0.15861240773791468),
 ('capital-gain', 0.2284592742844192),
 ('capital-loss', 0.07843149812577388),
```

```
 ('hours-per-week', 0.15529640153538432),
 ('sex_ Female', 0.021232345235427594),
 ('sex_ Male', 0.03430584411132491)]
```

[73]:
```
x.drop(['fnlwgt','salary'], axis=1).head()
```

[73]:
```
   age  education-num  capital-gain  capital-loss  hours-per-week  \
0   39             13          2174             0              40
1   50             13             0             0              13
2   38              9             0             0              40
3   53              7             0             0              40
4   28             13             0             0              40

   sex_ Female  sex_ Male
0        False       True
1        False       True
2        False       True
3        False       True
4         True      False
```

[74]:
```
set(x.columns) - set(xt.columns)
```

[74]:
```
set()
```

[75]:
```
list(x.drop('salary', axis=1).columns)
```

[75]:
```
['age',
 'fnlwgt',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'sex_ Female',
 'sex_ Male']
```

[76]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

[77]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

[78]:
```
accuracy_score(xt.salary, predictions)
```

[78]:
```
0.821018364965297
```

[79]:
```python
accuracy_score(xt.salary, predictions)
```

[79]:
```
0.821018364965297
```

[80]:
```python
confusion_matrix(xt.salary, predictions)
```

[80]:
```
array([[11462,   973],
       [ 1941,  1905]])
```

[81]:
```python
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.92      0.89     12435
         1.0       0.66      0.50      0.57      3846

    accuracy                           0.82     16281
   macro avg       0.76      0.71      0.73     16281
weighted avg       0.81      0.82      0.81     16281
```

[82]:
```python
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.86      0.92      0.89     12435
         1.0       0.66      0.50      0.57      3846

    accuracy                           0.82     16281
   macro avg       0.76      0.71      0.73     16281
weighted avg       0.81      0.82      0.81     16281
```

[83]:
```python
accuracy_score(x.salary, predictionsx)
```

[83]:
```
0.8955806025613464
```

[84]:
```python
confusion_matrix(x.salary, predictionsx)
```

[84]:
```
array([[24097,   623],
       [ 2777,  5064]])
```

[85]:
```python
print(classification_report(x.salary, predictionsx))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.97      0.93     24720
         1.0       0.89      0.65      0.75      7841
```

```
    accuracy                           0.90     32561
   macro avg       0.89      0.81      0.84     32561
weighted avg       0.90      0.90      0.89     32561
```

[86]:
```
print(classification_report(x.salary, predictionsx))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.97      0.93     24720
         1.0       0.89      0.65      0.75      7841

    accuracy                           0.90     32561
   macro avg       0.89      0.81      0.84     32561
weighted avg       0.90      0.90      0.89     32561
```

2 For the following use the above adult dataset.

3 1. Show the RandomForest outperforms the DecisionTree for a fixed max_depth by training using the train set and calculate precision, recall, f1, confusion matrix on golden-test set. Start with only numerical features/columns. (age, education-num, capital-gain, capital-loss, hours-per-week)

[95]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪confusion_matrix
import pandas as pd

# Assuming train_set and golden_test_set are DataFrames containing the required␣
 ↪columns

# Selecting only the numerical features
numerical_features = ['age', 'education-num', 'capital-gain', 'capital-loss',␣
 ↪'hours-per-week']

# Train set
X_train = x[numerical_features]
y_train = x['salary']  # Replace 'target_column' with the actual target column␣
 ↪name

# Golden-test set
```

```
X_test = xt[numerical_features]
y_test = xt['salary']  # Replace 'target_column' with the actual target column␣
↪name

# Train a DecisionTree model
dt_model = DecisionTreeClassifier(max_depth=5)  # Using a fixed max_depth of 5
dt_model.fit(X_train, y_train)

# Train a RandomForest model
rf_model = RandomForestClassifier(max_depth=5, n_estimators=100)  # Using a␣
↪fixed max_depth of 5 and 100 trees
rf_model.fit(X_train, y_train)

# Making predictions
dt_predictions = dt_model.predict(X_test)
rf_predictions = rf_model.predict(X_test)

# Calculate precision, recall, and F1 score
dt_precision = precision_score(y_test, dt_predictions)
dt_recall = recall_score(y_test, dt_predictions)
dt_f1 = f1_score(y_test, dt_predictions)

rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_f1 = f1_score(y_test, rf_predictions)

# Calculate confusion matrix
dt_conf_matrix = confusion_matrix(y_test, dt_predictions)
rf_conf_matrix = confusion_matrix(y_test, rf_predictions)
```

4 2. Use a RandomForest or DecisionTree and the `adult` dataset, systematically add new columns, one by one, that are non-numerical but converted using the feature-extraction techniques we learned. Using the golden-test set show [`precision`, `recall`, `f1`, `confusion matrix`] for each additional feature added.

[103]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪confusion_matrix
```

[113]:
```
non_numerical_columns = ['workclass', 'education', 'marital-status',␣
↪'occupation', 'relationship', 'race', 'sex', 'native-country']
```

```python
xt = pd.get_dummies(xt, columns=['salary'])

for column in non_numerical_columns:
    rf_model = RandomForestClassifier(max_depth=5, n_estimators=100)
    rf_model.fit(X_train, y_train)
    rf_predictions = rf_model.predict(X_test)
    rf_precision = precision_score(y_test, rf_predictions)
    rf_recall = recall_score(y_test, rf_predictions)
    rf_f1 = f1_score(y_test, rf_predictions)
    rf_conf_matrix = confusion_matrix(y_test, rf_predictions)
    print(f"Performance metrics for {column}: Precision={rf_precision}, Recall={rf_recall}, F1={rf_f1}, Confusion Matrix={rf_conf_matrix}")
```

```
Performance metrics for workclass: Precision=0.7644529383659818,
Recall=0.4160166406656266, F1=0.5388112476847955, Confusion Matrix=[[11942
493]
 [ 2246  1600]]
Performance metrics for education: Precision=0.763445978105664,
Recall=0.4170566822672907, F1=0.5394316462081722, Confusion Matrix=[[11938
497]
 [ 2242  1604]]
Performance metrics for marital-status: Precision=0.7629453681710214,
Recall=0.4175767030681227, F1=0.5397412199630315, Confusion Matrix=[[11936
499]
 [ 2240  1606]]
Performance metrics for occupation: Precision=0.7628326996197718,
Recall=0.4173166926677067, F1=0.5394957983193277, Confusion Matrix=[[11936
499]
 [ 2241  1605]]
Performance metrics for relationship: Precision=0.7735294117647059,
Recall=0.41029641185647425, F1=0.5361875637104995, Confusion Matrix=[[11973
462]
 [ 2268  1578]]
Performance metrics for race: Precision=0.7653256704980843,
Recall=0.4154966198647946, F1=0.5385911695315133, Confusion Matrix=[[11945
490]
 [ 2248  1598]]
Performance metrics for sex: Precision=0.7722195240407965,
Recall=0.41341653666146644, F1=0.5385266723116003, Confusion Matrix=[[11966
469]
 [ 2256  1590]]
Performance metrics for native-country: Precision=0.7601142313184198,
Recall=0.41523660946437857, F1=0.5370775180763411, Confusion Matrix=[[11931
504]
 [ 2249  1597]]
```