Trees Classification

March 11, 2020

- 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

[2]:
```
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
```

[3]:
```
df = pd.read_csv('../data/adult.data', index_col=False)
```

[4]:
```
golden = pd.read_csv('../data/adult.test', index_col=False)
```

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

[5]:
```
   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.
```

[6]:
```python
df.head()
```

[6]:
```
   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
```

[7]:
```python
df.columns
```

[7]:
```
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')
```

[8]:
```python
from sklearn import preprocessing
```

[9]:
```python
enc = preprocessing.OrdinalEncoder()
```

[10]:
```python
transform_columns = ['sex']
non_num_columns = ['workclass', 'education', 'marital-status',
                   'occupation', 'relationship', 'race', 'sex',
                   'native-country']
```

[11]:
```python
pd.get_dummies(df[transform_columns]).head()
```

[11]:
```
   sex_ Female  sex_ Male
0            0          1
1            0          1
2            0          1
3            0          1
```

```
4             1          0
```

[12]:
```
x = df.copy()

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

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

[13]:
```
x.head()
```

[13]:
```
   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            0          1
1     0.0            0          1
2     0.0            0          1
3     0.0            0          1
4     0.0            1          0
```

[14]:
```
xt = golden.copy()

xt = pd.concat([xt.drop(non_num_columns, axis=1),
               pd.get_dummies(golden[transform_columns])], axis=1,)

xt["salary"] = enc.fit_transform(golden[["salary"]])
```

[15]:
```
xt.salary.value_counts()
```

[15]:
```
0.0    12435
1.0     3846
Name: salary, dtype: int64
```

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

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

[17]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

Choose the model of your preference: DecisionTree or RandomForest

[18]:
```
model = RandomForestClassifier(criterion='entropy')
```

[19]:
```
model = DecisionTreeClassifier(criterion='entropy', max_depth=None)
```

[20]:
```
model.fit(x.drop(['fnlwgt','salary'], axis=1), x.salary)
```

[20]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[21]:
```
model.tree_.node_count
```

[21]:
```
8321
```

[22]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.
 ↪feature_importances_))
```

[22]:
```
[('age', 0.323449785000724),
 ('education-num', 0.1616195849881024),
 ('capital-gain', 0.22752700247562493),
 ('capital-loss', 0.07781064775704498),
 ('hours-per-week', 0.1538187445175173),
 ('sex_ Female', 0.0541573148047826),
 ('sex_ Male', 0.001616920456203861)]
```

[23]:
```
list(zip(x.drop(['fnlwgt','salary'], axis=1).columns, model.
 ↪feature_importances_))
```

[23]:

```
[('age', 0.323449785000724),
 ('education-num', 0.1616195849881024),
 ('capital-gain', 0.22752700247562493),
 ('capital-loss', 0.07781064775704498),
 ('hours-per-week', 0.1538187445175173),
 ('sex_ Female', 0.0541573148047826),
 ('sex_ Male', 0.001616920456203861)]
```


[24]:
```
x.drop(['fnlwgt','salary'], axis=1).head()
```

[24]:

	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            0          1
1            0          1
2            0          1
3            0          1
4            1          0
```

[25]:
```
set(x.columns) - set(xt.columns)
```

[25]:
```
set()
```

[26]:
```
list(x.drop('salary', axis=1).columns)
```

[26]:
```
['age',
 'fnlwgt',
 'education-num',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'sex_ Female',
 'sex_ Male']
```

[27]:
```
predictions = model.predict(xt.drop(['fnlwgt','salary'], axis=1))
predictionsx = model.predict(x.drop(['fnlwgt','salary'], axis=1))
```

[28]:
```
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix, auc, roc_curve
)
```

[29]:
```
accuracy_score(xt.salary, predictions)
```

[29]:
```
0.8202198882132548
```

[30]:
```
accuracy_score(xt.salary, predictions)
```

[30]:
```
0.8202198882132548
```

[31]:
```
confusion_matrix(xt.salary, predictions)
```

[31]:
```
array([[11457,   978],
       [ 1949,  1897]])
```

[32]:
```
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.92      0.89     12435
         1.0       0.66      0.49      0.56      3846

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

[33]:
```python
print(classification_report(xt.salary, predictions))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.92      0.89     12435
         1.0       0.66      0.49      0.56      3846

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

[34]:
```python
accuracy_score(x.salary, predictionsx)
```

[34]:
```
0.8955806025613464
```

[35]:
```python
confusion_matrix(x.salary, predictionsx)
```

[35]:
```
array([[24097,   623],
       [ 2777,  5064]])
```

[36]:
```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
```

[37]:
```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
```

1 For the following use the above `adult` dataset. Start with only numerical features/columns.

2 1. Show the RandomForest outperforms the DecisionTree for a fixed `max_depth` by training using the train set and `precision`, `recall`, `f1` on golden-test set.

3 2. For RandomForest or DecisionTree and using the `adult` dataset, systematically add new columns, one by one, that are non-numerical but converted using the feature-extraction techniques we learned. Show [`precision`, `recall`, `f1`] for each additional feature added.

4 3. Optional: Using gridSearch find the most optimal parameters for your model

Warning: this can be computationally intensive and may take some time. - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html - https://scikit-learn.org/stable/modules/grid_search.html

[166]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import (accuracy_score,
                             classification_report,
                             confusion_matrix,
                             auc,
                             roc_curve)
```

[167]:
```
adult_data = pd.read_csv('../data/adult.data', index_col=False)
golden = pd.read_csv('../data/adult.test', index_col=False)
```

[168]:
```
# select columns that are of type int64.
golden_int_cols = golden.select_dtypes(include='int64').copy()
golden_int_cols['salary'] = golden.loc[:,'salary'].copy()
golden_int_cols.head()
```

```
[168]:    age  fnlwgt  education-num  capital-gain  capital-loss  hours-per-week  \
       0   25  226802              7             0             0              40
       1   38   89814              9             0             0              50
       2   28  336951             12             0             0              40
       3   44  160323             10          7688             0              40
       4   18  103497             10             0             0              30

          salary
       0  <=50K.
       1  <=50K.
       2   >50K.
       3   >50K.
       4  <=50K.
```

```
[169]: # assign int values to salary for model.
encoder = preprocessing.OrdinalEncoder()
golden_int_cols['salary'] = encoder.fit_transform(np.
 ↪array(golden_int_cols['salary']).reshape(-1,1))
golden_int_cols.salary.unique()
```

```
[169]: array([0., 1.])
```

```
[170]: # create testing and training set.
x_train, x_test, y_train, y_test = train_test_split(golden_int_cols.loc[:
 ↪,['age',
                                                                               ␣
 ↪'fnlwgt',
                                                                               ␣
 ↪'education-num',
                                                                               ␣
 ↪'capital-gain',
                                                                               ␣
 ↪'capital-loss',
                                                                               ␣
 ↪'hours-per-week']],
                                                         golden_int_cols.loc[:
 ↪,'salary'], train_size = .8)
```

```
[171]: # run Random Forest on golden data set.
rf_model = RandomForestClassifier(criterion = 'entropy', max_depth = 5)
rf_model.fit(x_train, y_train)
rf_pred_vals = rf_model.predict(golden_int_cols.drop(['salary'], axis = 1))

# run Decision Tree on golden data set.
dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5)
dt_model.fit(x_train, y_train)
dt_pred_vals = dt_model.predict(golden_int_cols.drop(['salary'], axis = 1))
```

5 Question 1 Ans

[172]:
```
# check accuracy for each model.
rf_acc = accuracy_score(golden_int_cols.salary, rf_pred_vals)
dt_acc = accuracy_score(golden_int_cols.salary, dt_pred_vals)
print(f'Accuracy for RandomForest: {rf_acc}')
print(classification_report(golden_int_cols.salary, rf_pred_vals))
```

```
Accuracy for RandomForest: 0.8309686137215159
              precision    recall  f1-score   support

         0.0       0.83      0.98      0.90     12435
         1.0       0.84      0.35      0.50      3846

    accuracy                           0.83     16281
   macro avg       0.83      0.67      0.70     16281
weighted avg       0.83      0.83      0.80     16281
```

[192]:
```
print(f'Accuracy for DecisionTree: {dt_acc}')
print(classification_report(golden_int_cols.salary, dt_pred_vals))
```

```
Accuracy for DecisionTree: 0.8272219151157791
              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.74      0.41      0.53      3846

    accuracy                           0.83     16281
   macro avg       0.79      0.68      0.71     16281
weighted avg       0.82      0.83      0.81     16281
```

[174]:
```
golden.dtypes
```

[174]:
```
age                int64
workclass         object
fnlwgt             int64
education         object
education-num      int64
marital-status    object
occupation        object
relationship      object
race              object
sex               object
capital-gain       int64
capital-loss       int64
```

```
hours-per-week     int64
native-country    object
salary            object
dtype: object
```

[175]:
```python
# check for na values.
print(np.array(golden.isna().sum()))

golden['salary'] = encoder.fit_transform(np.array(golden_int_cols['salary']).
 ↪reshape(-1,1))

# organize the dataframe to contain int cols first and then object cols.
golden = golden[['salary',
                 'age',
                 'fnlwgt',
                 'education-num',
                 'capital-gain',
                 'capital-loss',
                 'hours-per-week',
                 'workclass',
                 'education',
                 'marital-status',
                 'occupation',
                 'relationship',
                 'race',
                 'sex',
                 'native-country']]

# remove white space from values in df.
golden = golden.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

[179]:
```python
non_num_cols = list(golden.select_dtypes(include='object').columns)

golden['workclass'] = encoder.fit_transform(np.array(golden['workclass']).
 ↪reshape(-1,1))
golden['education'] = encoder.fit_transform(np.array(golden['education']).
 ↪reshape(-1,1))
golden['marital-status'] = encoder.fit_transform(np.
 ↪array(golden['marital-status']).reshape(-1,1))
golden['occupation'] = encoder.fit_transform(np.array(golden['occupation']).
 ↪reshape(-1,1))
golden['relationship'] = encoder.fit_transform(np.array(golden['relationship']).
 ↪reshape(-1,1))
golden['race'] = encoder.fit_transform(np.array(golden['race']).reshape(-1,1))
golden['sex'] = encoder.fit_transform(np.array(golden['sex']).reshape(-1,1))
```

```
golden['native-country'] = encoder.fit_transform(np.
↪array(golden['native-country']).reshape(-1,1))
```

[177]:
```
rf_classification_reports_list = []
dt_classification_reports_list = []
```

[193]:
```
'''
Create a list containing a list of columns that adds
one column at a time. Allows to run models by adding one
column at a time and evaluating the effect each column will have.
'''
iter_var = 7
col_groups = []
col_len = len(golden.columns)
while iter_var <= col_len:
    col = golden.iloc[:,0:iter_var]
    x_train, x_test, y_train, y_test = train_test_split(col.drop(['salary'],␣
↪axis = 1),
                                                        col['salary'],
                                                        train_size = 0.8)
    # run Random Forest on golden data set.
    rf_model = RandomForestClassifier(criterion = 'entropy', max_depth = 5)
    rf_model.fit(x_train, y_train)
    rf_pred_vals = rf_model.predict(col.drop(['salary'], axis = 1))

    # run Decision Tree on golden data set.
    dt_model = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5)
    dt_model.fit(x_train, y_train)
    dt_pred_vals = dt_model.predict(col.drop(['salary'], axis = 1))

    rf_acc = classification_report(golden.salary, rf_pred_vals)
    dt_acc = classification_report(golden.salary, dt_pred_vals)

    rf_classification_reports_list.append(rf_acc)
    dt_classification_reports_list.append(dt_acc)
    iter_var += 1
```

6 Ans 2

[196]:
```
for i,j in enumerate(rf_classification_reports_list):
    print(j)
```

```
0.8333640439776426
0.8257478041889319
0.8283274983109146
0.8234137952214238
```

```
0.8240894293962288
0.8298630305263804
0.8285731834653891
0.8296787666605245
0.8281432344450587
0.8291873963515755
              precision    recall  f1-score   support

         0.0       0.83      0.98      0.90     12435
         1.0       0.85      0.35      0.50      3846

    accuracy                           0.83     16281
   macro avg       0.84      0.67      0.70     16281
weighted avg       0.84      0.83      0.80     16281

              precision    recall  f1-score   support

         0.0       0.82      0.99      0.90     12435
         1.0       0.94      0.28      0.44      3846

    accuracy                           0.83     16281
   macro avg       0.88      0.64      0.67     16281
weighted avg       0.85      0.83      0.79     16281

              precision    recall  f1-score   support

         0.0       0.82      0.99      0.90     12435
         1.0       0.89      0.32      0.47      3846

    accuracy                           0.83     16281
   macro avg       0.86      0.65      0.68     16281
weighted avg       0.84      0.83      0.80     16281

              precision    recall  f1-score   support

         0.0       0.83      0.99      0.90     12435
         1.0       0.88      0.32      0.47      3846

    accuracy                           0.83     16281
   macro avg       0.85      0.66      0.69     16281
weighted avg       0.84      0.83      0.80     16281

              precision    recall  f1-score   support

         0.0       0.83      0.98      0.90     12435
         1.0       0.83      0.34      0.48      3846

    accuracy                           0.83     16281
```

```
   macro avg       0.83      0.66      0.69     16281
weighted avg       0.83      0.83      0.80     16281

              precision    recall  f1-score   support

         0.0       0.83      0.98      0.90     12435
         1.0       0.84      0.33      0.48      3846

    accuracy                           0.83     16281
   macro avg       0.83      0.66      0.69     16281
weighted avg       0.83      0.83      0.80     16281

              precision    recall  f1-score   support

         0.0       0.83      0.98      0.90     12435
         1.0       0.85      0.35      0.49      3846

    accuracy                           0.83     16281
   macro avg       0.84      0.66      0.70     16281
weighted avg       0.83      0.83      0.80     16281

              precision    recall  f1-score   support

         0.0       0.82      0.99      0.90     12435
         1.0       0.88      0.31      0.46      3846

    accuracy                           0.83     16281
   macro avg       0.85      0.65      0.68     16281
weighted avg       0.84      0.83      0.79     16281

              precision    recall  f1-score   support

         0.0       0.82      0.99      0.90     12435
         1.0       0.89      0.32      0.47      3846

    accuracy                           0.83     16281
   macro avg       0.86      0.65      0.68     16281
weighted avg       0.84      0.83      0.80     16281
```

[197]:
```
for i,j in enumerate(dt_classification_reports_list):
    print(j)
```

```
0.8157975554327129
0.8250107487255083
0.8227995823352374
0.8159203980099502
0.8193599901725939
```

```
0.8194828327498311
0.8157361341440943
0.8251950125913642
0.8272219151157791
              precision    recall  f1-score   support

         0.0       0.81      0.99      0.89     12435
         1.0       0.90      0.25      0.39      3846

    accuracy                           0.82     16281
   macro avg       0.86      0.62      0.64     16281
weighted avg       0.83      0.82      0.77     16281

              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.73      0.41      0.52      3846

    accuracy                           0.82     16281
   macro avg       0.78      0.68      0.71     16281
weighted avg       0.81      0.82      0.80     16281

              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.73      0.41      0.52      3846

    accuracy                           0.82     16281
   macro avg       0.78      0.68      0.71     16281
weighted avg       0.81      0.82      0.80     16281

              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.73      0.42      0.54      3846

    accuracy                           0.83     16281
   macro avg       0.79      0.69      0.72     16281
weighted avg       0.82      0.83      0.81     16281

              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.72      0.41      0.52      3846

    accuracy                           0.82     16281
   macro avg       0.78      0.68      0.71     16281
weighted avg       0.81      0.82      0.80     16281
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.72      0.41      0.52      3846

    accuracy                           0.82     16281
   macro avg       0.78      0.68      0.71     16281
weighted avg       0.81      0.82      0.80     16281

              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.73      0.43      0.54      3846

    accuracy                           0.83     16281
   macro avg       0.79      0.69      0.72     16281
weighted avg       0.82      0.83      0.81     16281

              precision    recall  f1-score   support

         0.0       0.84      0.94      0.89     12435
         1.0       0.70      0.42      0.53      3846

    accuracy                           0.82     16281
   macro avg       0.77      0.68      0.71     16281
weighted avg       0.81      0.82      0.80     16281

              precision    recall  f1-score   support

         0.0       0.84      0.95      0.89     12435
         1.0       0.73      0.42      0.54      3846

    accuracy                           0.83     16281
   macro avg       0.79      0.69      0.72     16281
weighted avg       0.82      0.83      0.81     16281
```

[ ]: