Classification tree

December 22, 2022

[1]:
```
# We import some useful libraries.
import pandas as pd
import warnings
import numpy as np
import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import model_selection
from sklearn.model_selection import train_test_split

data= pd.read_csv("train.csv")

data['Lead'].replace({'Male':1, 'Female':0}, inplace = True)

# Separate the target variable from the dataframe as we cannot train the model␣
↪with the target variable.
X = data.drop(columns = ["Lead"])
y = data['Lead']

# We split the balanced data into train and test dataframes.
# random_state seed gives us the same train and test datasets no matter the␣
↪times we split it.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 4045)
```

0.0.1 Default tree

[2]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

[2]: DecisionTreeClassifier()

[3]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)
```

```python
print(accuracy_score(y_train, y_train_pred))
confusion_matrix(y_train, y_train_pred)
```

```
1.0
```

[3]:
```
array([[189,   0],
       [  0, 590]], dtype=int64)
```

[4]:
```python
y_test_pred = dt.predict(X_test)
print(accuracy_score(y_test, y_test_pred))
confusion_matrix(y_test, y_test_pred)
```

```
0.7846153846153846
```

[4]:
```
array([[ 34,  31],
       [ 25, 170]], dtype=int64)
```

Clearly overfits. Let's set a maximum tree depth.

[5]:
```python
dt = DecisionTreeClassifier(max_depth = 3)

dt.fit(X_train, y_train)
y_train_pred = dt.predict(X_train)

print(accuracy_score(y_train, y_train_pred))
confusion_matrix(y_train, y_train_pred)
```

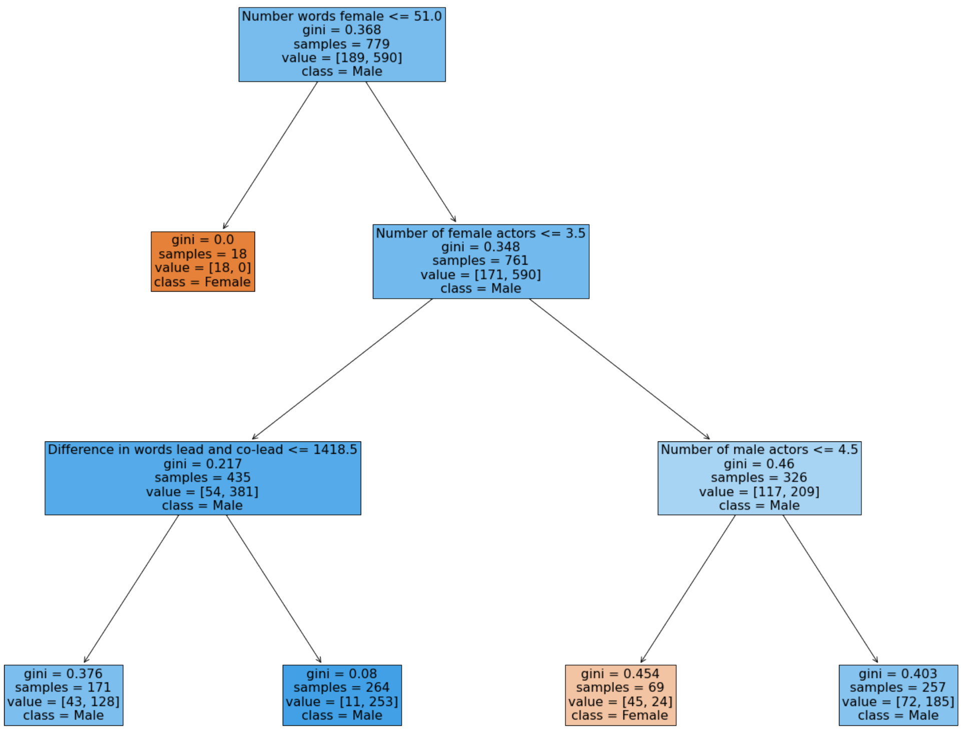
```
0.8074454428754814
```

[5]:
```
array([[ 63, 126],
       [ 24, 566]], dtype=int64)
```

We now get a more reasonable accuracy. Let's graph how is this tree.

[6]:
```python
import graphviz
from IPython.display import SVG
from sklearn import tree


# dot_data = tree.export_graphviz(dt, out_file = None, feature_names = X_train.
 ↪columns,
#                                  class_names = ['Female', 'Male'], filled =␣
 ↪True, rounded=True,
#                                  leaves_parallel=True, proportion=True)
# graph = graphviz.Source(dot_data)
# graph.format = 'svg'
# graph.render('dt',view=True)
# graph
```

```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(dt,
                   feature_names=X.columns,
                   class_names=['Female', "Male"],
                   filled=True, fontsize=16)
plt.savefig("dt.svg", format = "svg", dpi=300, transparent = True, bbox_inches␣
 ↪= 'tight')
plt.show()
```



0.0.2 Hyperparameter tuning

[7]:
```
from sklearn.model_selection import GridSearchCV

# Parameters grid:
min_samples_split = [2, 3 ,4 ,5, 6, 8, 9, 10, 15, 20, 25, 30, 40, 50] # Minimum␣
 ↪sample number to split a node
min_samples_leaf = [2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50] #␣
 ↪Minimum sample number that can be stored in a leaf node
max_depth = [2, 3, 4]
```

```
max_leaf_nodes = [2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50]
random_grid = {'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'max_depth': max_depth,
               'max_leaf_nodes' : max_leaf_nodes}



# Hyperparameter tuning with the RandomizedSearchCV function with a 10-fold␣
 ↪cross-validation and  n_iter = 200 random iterations from the parameter grid:
dt_tunned = GridSearchCV(estimator = dt, param_grid = random_grid, cv = 3,␣
 ↪verbose=2, n_jobs = -1, scoring = "accuracy")
dt_tunned.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 8820 candidates, totalling 26460 fits
```

[7]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(max_depth=3), n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4],
                         'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20,
                                            30, 40, 50],
                         'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 15,
                                              20, 25, 30, 40, 50],
                         'min_samples_split': [2, 3, 4, 5, 6, 8, 9, 10, 15, 20,
                                               25, 30, 40, 50]},
             scoring='accuracy', verbose=2)
```

[8]:
```
# We present the best parameters selected in the hyperparameter tuning.

# Grid selected:
print ('Random grid: ', random_grid, '\n')

# Best parameters:
print ('Best Parameters: ', dt_tunned.best_params_, ' \n')
```

```
Random grid:  {'min_samples_split': [2, 3, 4, 5, 6, 8, 9, 10, 15, 20, 25, 30,
40, 50], 'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40,
50], 'max_depth': [2, 3, 4], 'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 15,
20, 30, 40, 50]}

Best Parameters:  {'max_depth': 3, 'max_leaf_nodes': 5, 'min_samples_leaf': 15,
'min_samples_split': 2}
```

0.0.3 Tuned tree

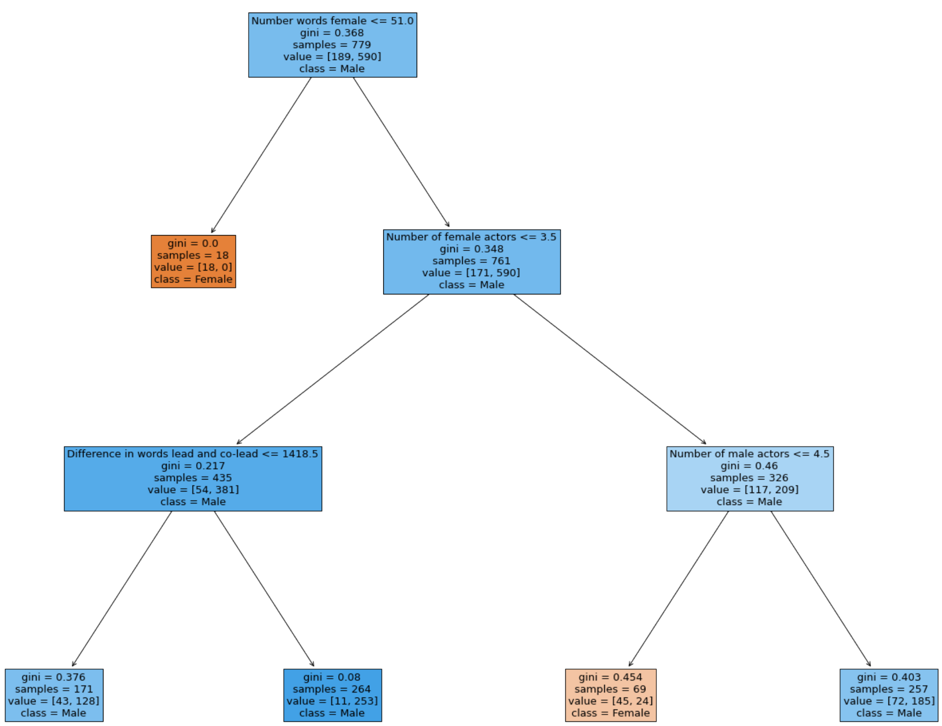
Now we train the model in all the training data with the best hyperparameters.

[9]:
```
dt_tunned = DecisionTreeClassifier(max_depth = 3, max_leaf_nodes = 5 ,␣
    ↪min_samples_leaf = 15, min_samples_split = 2)
dt_tunned.fit(X_train, y_train)

y_train_pred = dt_tunned.predict(X_train)

print(accuracy_score(y_train, y_train_pred))
confusion_matrix(y_train, y_train_pred)
```

```
0.8074454428754814
```

[9]:
```
array([[ 63, 126],
       [ 24, 566]], dtype=int64)
```

[10]:
```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(dt_tunned,
                   feature_names = X.columns,
                   class_names=['Female', "Male"],
                   filled = True, fontsize=13)
plt.savefig("dt_tunned.svg", format = "svg", dpi = 300, transparent = True,␣
    ↪bbox_inches = 'tight')
plt.show()
```



```
[11]: y_test_pred = dt_tunned.predict(X_test)
print(accuracy_score(y_test, y_test_pred))
confusion_matrix(y_test, y_test_pred)
```

```
0.8038461538461539
```

```
[11]: array([[ 21,  44],
       [  7, 188]], dtype=int64)
```

```
[12]: from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
    ↪f1_score
dt_tunned.fit(X_train,y_train)

print('Training set metrics:')
print('Accuracy:', accuracy_score(y_train, dt_tunned.predict(X_train)))
print('Precision:', precision_score(y_train, dt_tunned.predict(X_train)))
print('Recall:', recall_score(y_train, dt_tunned.predict(X_train)))
print('F1:', f1_score(y_train, dt_tunned.predict(X_train)))

print('\n')

print('Test set metrics:')
print('Accuracy:', accuracy_score(y_test, dt_tunned.predict(X_test)))
print('Precision:', precision_score(y_test, dt_tunned.predict(X_test)))
print('Recall:', recall_score(y_test, dt_tunned.predict(X_test)))
print('F1:', f1_score(y_test, dt_tunned.predict(X_test)))
```

```
Training set metrics:
Accuracy: 0.8074454428754814
Precision: 0.8179190751445087
Recall: 0.9593220338983051
F1: 0.8829953198127924


Test set metrics:
Accuracy: 0.8038461538461539
Precision: 0.8103448275862069
Recall: 0.9641025641025641
F1: 0.8805620608899296
```

0.0.4 Gini and permutation feature importances

```
[13]: dt_tunned.feature_importances_
```

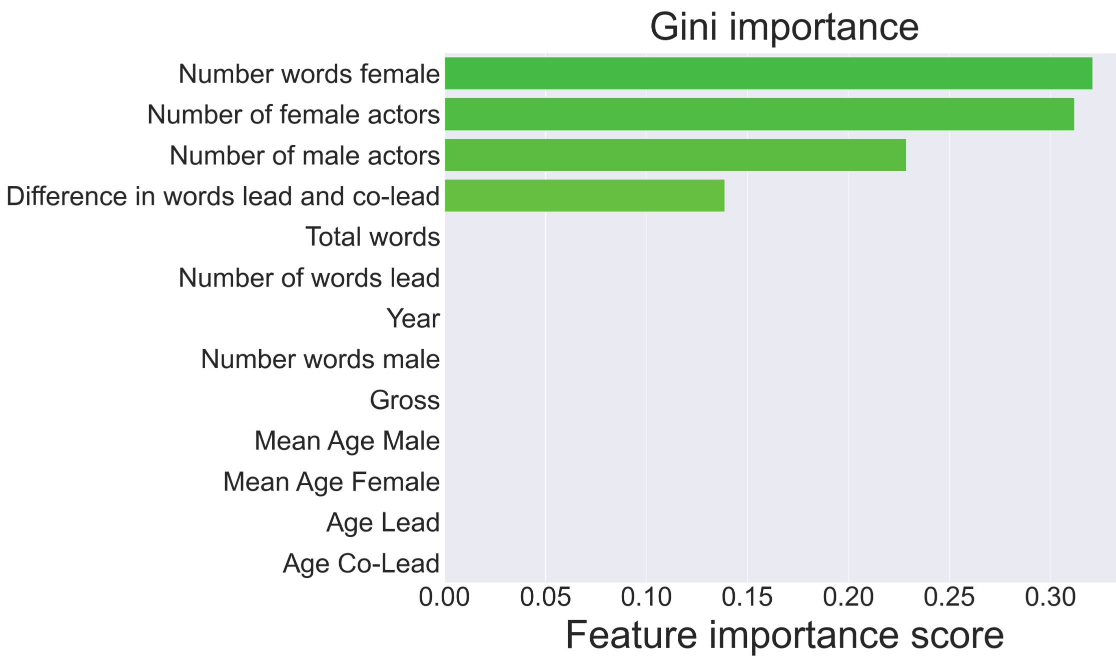
[13]: array([0.32091716, 0.   , 0.   , 0.13868864, 0.22857655,
        0.   , 0.31181765, 0.   , 0.   , 0.   ,
        0.   , 0.   , 0.   ])

[14]:
```
feature_scores = pd.Series(dt_tunned.feature_importances_, index = X.columns).
 ↪sort_values(ascending = False)
feature_scores
```

[14]:
```
Number words female                     0.320917
Number of female actors                 0.311818
Number of male actors                   0.228577
Difference in words lead and co-lead    0.138689
Total words                             0.000000
Number of words lead                    0.000000
Year                                    0.000000
Number words male                       0.000000
Gross                                   0.000000
Mean Age Male                           0.000000
Mean Age Female                         0.000000
Age Lead                                0.000000
Age Co-Lead                             0.000000
dtype: float64
```

[15]:
```
# Creating a seaborn bar plot:
from colour import Color
import seaborn as sns
import matplotlib.pyplot as plt

# Font size:
sns.set(font_scale = 6)

# Gradient colour, green - more contribution, red -less contribution
limegreen= Color("limegreen")
colors = list(limegreen.range_to(Color("red"),21))
colors = [color.rgb for color in colors]

f, ax = plt.subplots(figsize=(30, 24))
ax = sns.barplot(x = feature_scores, y = feature_scores.index, palette = colors)
ax.set_title("Gini importance", y = 1.02,  fontsize = 95)
ax.set_xlabel("Feature importance score", fontsize = 95)
ax.xaxis.set_label_coords(0.5, -.07)


# We save the plot for the project:
f.savefig('GiniImportance.svg', format = 'svg', dpi = 1200, bbox_inches =␣
 ↪'tight', transparent = True)
# bbox_inches='tight' because the feature names were cut off
```

```
plt.show()
```



[16]:
```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(dt_tunned, random_state=1).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

[16]: <IPython.core.display.HTML object>

[17]:
```
from sklearn.inspection import permutation_importance
perm_imp = permutation_importance(dt_tunned, X_test, y_test)

# View the feature scores as a dataframe to plot them:
feature_permutation_scores = pd.Series(perm_imp.importances_mean, index=X.
 ↪columns).sort_values(ascending=False)
feature_permutation_scores

# Normalise the feature scores to sum 1, so we can compare its relative␣
 ↪contribution to the model output change and compare it
# to the Gini importance scores.
normalized_feature_permutation_scores= feature_permutation_scores /␣
 ↪sum(feature_permutation_scores)

sns.set(font_scale=6)
```

```
limegreen= Color("limegreen")
colors = list(limegreen.range_to(Color("red"),21))
colors = [color.rgb for color in colors]

f, ax = plt.subplots(figsize=(30, 24))
ax = sns.barplot(x=normalized_feature_permutation_scores,␣
↪y=normalized_feature_permutation_scores.index,palette=colors)
ax.set_title("Feature permutation importance",y=1.03, fontsize=95)
ax.set_xlabel("Feature importance score", fontsize=95)
ax.xaxis.set_label_coords(0.5, -.07)

f.savefig('FeaturePermutation.svg', format='svg', dpi=1200, ␣
↪bbox_inches='tight', transparent = True)
plt.show()
```