Bagging

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
```

[2]:
```
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 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)
```

[3]:
```
from sklearn.ensemble import BaggingClassifier
bagg = BaggingClassifier(random_state=123)
bagg.fit(X_train, y_train)
```

[3]:
```
BaggingClassifier(random_state=123)
```

[4]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

y_train_pred = bagg.predict(X_train)
y_test_pred = bagg.predict(X_test)

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

```
0.9961489088575096
```

[4]: array([[187,   2],
       [1, 589]], dtype=int64)

[5]:
```
print(accuracy_score(y_test, y_test_pred))
confusion_matrix(y_test, y_test_pred)
```

```
0.823076923076923
```

[5]: array([[38, 27],
       [19, 176]], dtype=int64)

[6]:
```
from sklearn.model_selection import GridSearchCV

# Parameters grid:
n_estimators = [100, 200, 300, 500]
#max_depth = [5, 10, 15, 25, 30]
max_samples = [50, 75, 100]
max_features = [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13]

# Creating a dictionary for the hyper parameters
hyperbag = dict(n_estimators = n_estimators, max_samples = max_samples,
              max_features = max_features)

#Applying GridSearchCV to get the best value for hyperparameters
gridbag = GridSearchCV(bagg, hyperbag, cv = 3, verbose = 1, n_jobs = -1)
bestbag = gridbag.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 132 candidates, totalling 396 fits
```

[7]:
```
# We present the best parameters selected in the hyperparameter tuning.

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

# Best parameters:
print ('Best Parameters: ', gridbag.best_estimator_, ' \n')
```

```
Random grid:  GridSearchCV(cv=3, estimator=BaggingClassifier(random_state=123),
n_jobs=-1,
             param_grid={'max_features': [1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13],
                         'max_samples': [50, 75, 100],
                         'n_estimators': [100, 200, 300, 500]},
             verbose=1)

Best Parameters:  BaggingClassifier(max_features=13, max_samples=100,
n_estimators=300,
                  random_state=123)
```

[8]:
```python
# Fitting the bagging model with the best hyper parameters obtained through␣
↪GridSearchCV
bagg1 = BaggingClassifier(max_features = 13, max_samples = 100,n_estimators =␣
↪300, random_state = 123)
bagg1.fit(X_train, y_train)
```

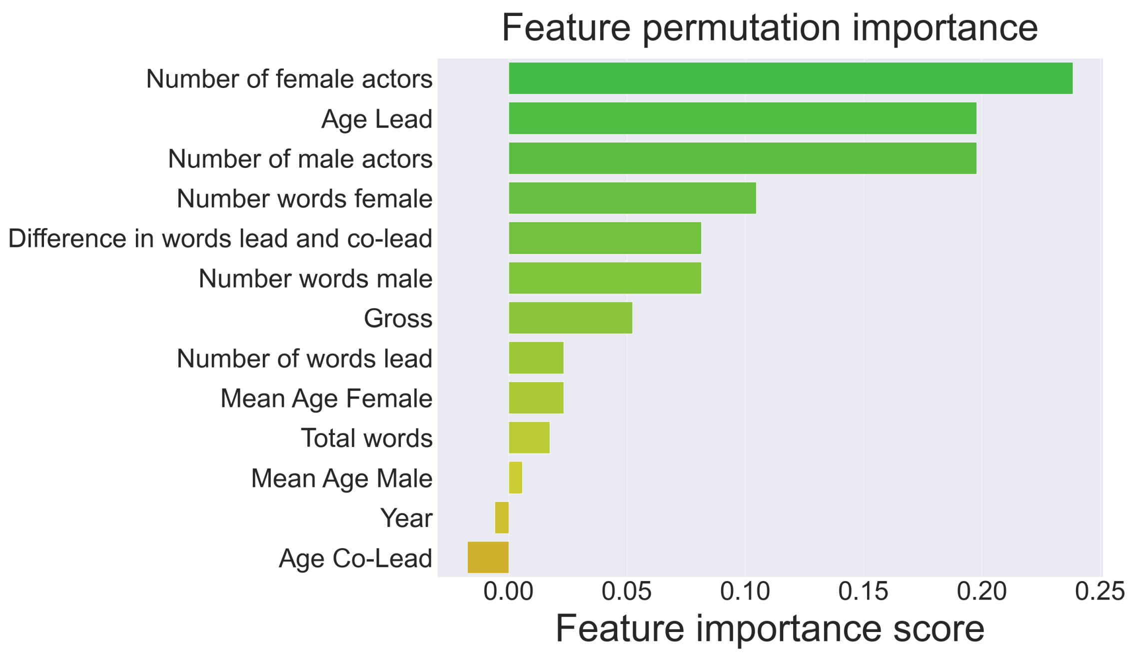
[8]:
```
BaggingClassifier(max_features=13, max_samples=100, n_estimators=300,
                  random_state=123)
```

[9]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

print('Training set metrics:')
print('Accuracy:', accuracy_score(y_train, bagg1.predict(X_train)))
print('Precision:', precision_score(y_train, bagg1.predict(X_train)))
print('Recall:', recall_score(y_train, bagg1.predict(X_train)))
print('F1:', f1_score(y_train, bagg1.predict(X_train)))

print('\n')

print('Test set metrics:')
print('Accuracy:', accuracy_score(y_test, bagg1.predict(X_test)))
print('Precision:', precision_score(y_test, bagg1.predict(X_test)))
print('Recall:', recall_score(y_test, bagg1.predict(X_test)))
print('F1:', f1_score(y_test, bagg1.predict(X_test)))
```

```
Training set metrics:
Accuracy: 0.8716302952503209
Precision: 0.8561046511627907
Recall: 0.9983050847457627
F1: 0.9217527386541471


Test set metrics:
Accuracy: 0.8038461538461539
Precision: 0.8025210084033614
Recall: 0.9794871794871794
F1: 0.8822170900692841
```

[10]:
```python
from sklearn.inspection import permutation_importance
import seaborn as sns
from colour import Color
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(bagg1, random_state=1).fit(X_test, y_test)
```

```
perm_imp = permutation_importance(bagg1, 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('bagging.svg', format='svg', dpi=1200,  bbox_inches='tight',␣
 ↪transparent = True)
plt.show()
```


Feature permutation importance
Number of female actors
Age Lead
Number of male actors
Number words female
Difference in words lead and co-lead
Number words male
Gross
Number of words lead
Mean Age Female
Total words
Mean Age Male
Year
Age Co-Lead
0.00
0.05
0.10
0.15
0.20
0.25
Feature importance score