Random forest

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.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn import model_selection
```

[2]:
```
data = pd.read_csv("train.csv")
```

[3]:
```
data['Lead'].replace({'Male':1, 'Female':0}, inplace = True)
```

[4]:
```
from sklearn.model_selection import train_test_split

# 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)
```

1 Random forest classifier

1.0.1 Default random forest

[5]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score

# Random forest classifier training with default sklearn parameters.
rfc = RandomForestClassifier(random_state=123)
```

```
rfc.fit(X_train,y_train)

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

```
Training set metrics:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1: 1.0
```

Clearly overfitting, let's perform hyperparameter tuning via 3-fold cross-validation in the training set.

[6]:
```
# Random forest classifier training with default sklearn parameters.
rfc = RandomForestClassifier(random_state = 123)

rfc_cv_score = cross_val_score(rfc, X_train, y_train, cv = 3, scoring =␣
 ↪'accuracy')
print('\033[1m' + 'All Accuracy scores' + '\033[0m', '\n')
print(rfc_cv_score)

print('\033[1m' + 'Mean accuracy score' + '\033[0m')
print(rfc_cv_score.mean())
```

```
All Accuracy scores

[0.82307692 0.83461538 0.81081081]
Mean accuracy score
0.8228343728343729
```

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

# Parameters grid:
n_estimators =  [25, 50, 75, 100, 150, 200] # Number of trees in the random␣
 ↪forest
max_features = [13, 'sqrt'] # Number of features in consideration at every split
min_samples_split = [5, 6, 8, 9, 10, 15, 20, 25] # Minimum sample number to␣
 ↪split a node
min_samples_leaf = [5, 6, 7, 8, 9, 10, 15, 20, 25] # Minimum sample number that␣
 ↪can be stored in a leaf node
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

```
rfc_random = GridSearchCV(estimator = rfc, param_grid = random_grid, cv = 3,␣
 ↪verbose = 2, n_jobs = -1, scoring= "accuracy")
rfc_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 864 candidates, totalling 2592 fits
```

[7]:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=123),
             n_jobs=-1,
             param_grid={'max_features': [13, 'sqrt'],
                         'min_samples_leaf': [5, 6, 7, 8, 9, 10, 15, 20, 25],
                         'min_samples_split': [5, 6, 8, 9, 10, 15, 20, 25],
                         'n_estimators': [25, 50, 75, 100, 150, 200]},
             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: ', rfc_random.best_estimator_, ' \n')
```

```
Random grid:  {'n_estimators': [25, 50, 75, 100, 150, 200], 'max_features': [13,
'sqrt'], 'min_samples_split': [5, 6, 8, 9, 10, 15, 20, 25], 'min_samples_leaf':
[5, 6, 7, 8, 9, 10, 15, 20, 25]}

Best Parameters:  RandomForestClassifier(max_features=13, min_samples_leaf=7,
min_samples_split=5,
                       random_state=123)
```

1.0.2 Tuned random forest

[9]:

```
import warnings
warnings.filterwarnings("ignore")

rfc_tunned = RandomForestClassifier(n_estimators = 100, max_features = 13,␣
 ↪min_samples_leaf = 7,
                        min_samples_split = 5, random_state = 123)

rfc_tunned_cv_score = cross_val_score(rfc_tunned, X_train, y_train, cv = 3,␣
 ↪scoring = 'accuracy')

print('\033[1m' + 'All Accuracy scores' + '\033[0m', '\n')
print(rfc_tunned_cv_score)

print('\033[1m' + 'Mean accuracy score' + '\033[0m')
```

```
print(rfc_tunned_cv_score.mean())
```

All Accuracy scores

[0.84230769 0.86153846 0.83397683]
Mean accuracy score
0.845940995940996

[10]:
```
rfc_tunned.fit(X_train,y_train)

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

print('\n')

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

```
Training set metrics:
Accuracy: 0.9332477535301669
Precision: 0.9229559748427673
Recall: 0.9949152542372881
F1: 0.9575856443719413


Test set metrics:
Accuracy: 0.8307692307692308
Precision: 0.8355555555555556
Recall: 0.9641025641025641
F1: 0.8952380952380953
```

1.0.3 Feature importance

[11]:
```
from sklearn.inspection import permutation_importance
import seaborn as sns
from colour import Color
import eli5
from eli5.sklearn import PermutationImportance
import matplotlib.pyplot as plt

perm_imp = permutation_importance(rfc_tunned, X_test, y_test)
```

```python
# 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('randomforest.svg', format='svg', dpi=1200,  bbox_inches='tight',
    transparent = True)
plt.show()
```