Tune_Random_forests

December 10, 2018

1 Random Forests

```
In [54]: import pandas as pd
         import matplotlib.pyplot as plt
         import numpy as np

         from IPython.display import set_matplotlib_formats
         set_matplotlib_formats('svg')

         from sklearn.ensemble import RandomForestClassifier
         from sklearn.metrics import accuracy_score, balanced_accuracy_score, make_scorer
         from sklearn.metrics import roc_auc_score, roc_curve
         from sklearn.model_selection import train_test_split, RandomizedSearchCV
         from sklearn.model_selection import cross_val_score
```

```
In [2]: # Get features
        samples = pd.read_csv("samples.csv", index_col=0)
        microbes = pd.read_csv("microbes.csv", index_col=0)
        metabolites = pd.read_csv("metabolites.csv", index_col=0)
        combined_features = pd.concat([microbes, metabolites], axis=1)
        # Label vector
        labels = samples.case
```

1.1 Metabolites

```
In [3]: # We'll use 80% for training, and 20% for testing
        X_train, X_test, labels_train, labels_test = train_test_split(
            metabolites, labels, test_size=0.2, random_state=42)
```

Random forest

```
In [37]: rf = RandomForestClassifier(n_estimators=30, min_samples_split=10, min_samples_leaf=1
                                     max_features='sqrt', max_depth=80, bootstrap=True, random_
         #scores = cross_val_score(rf, X_train, y_train, cv=5)
         #print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
         scores = cross_val_score(rf, X_train, labels_train, cv=5,
                                  scoring=make_scorer(roc_auc_score))
         print("AUC: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
AUC: 0.65 (+/- 0.18)
```

```
In [39]: rf = RandomForestClassifier(n_estimators=11, min_samples_split=2, min_samples_leaf=1,
                                     max_features='sqrt', max_depth=10, bootstrap=True, random_
         #scores = cross_val_score(rf, X_train, y_train, cv=5)
         #print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
         scores = cross_val_score(rf, X_train, labels_train, cv=5,
                                  scoring=make_scorer(roc_auc_score))
         print("AUC: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
AUC: 0.64 (+/- 0.15)
```

```
In [ ]: {'random_state': 42,
         'n_estimators': 30,
         'min_samples_split': 10,
         'min_samples_leaf': 1,
         'max_features': 'sqrt',
         'max_depth': 80,
         'bootstrap': True}
```

1.2 Generate hyperparameter grid

```
In [40]: # Number of trees in random forest
         n_estimators = [int(x) for x in np.linspace(start = 11, stop = 50, num = 30)]
         # Number of features to consider at every split
         max_features = ['sqrt']
         # Maximum number of levels in tree
         max_depth = [int(x) for x in np.linspace(2, 80, num = 11)]
         max_depth.append(None)
         # Minimum number of samples required to split a node
         min_samples_split = [2, 5, 10]
         # Minimum number of samples required at each leaf node
         min_samples_leaf = [1, 2, 4]
         # Method of selecting samples for training each tree
         bootstrap = [True, False]

         # Create the random grid
         random_grid = {'n_estimators': n_estimators,
                        'max_features': max_features,
                        'max_depth': max_depth,
                        'min_samples_split': min_samples_split,
                        'min_samples_leaf': min_samples_leaf,
                        'bootstrap': bootstrap,
                        'random_state': [42]}

         random_grid
```

```
Out[40]: {'n_estimators': [11,
           12,
           13,
           15,
           16,
           17,
           19,
           20,
           21,
           23,
           24,
           25,
           27,
           28,
           29,
           31,
           32,
           33,
           35,
           36,
           37,
           39,
           40,
           41,
           43,
           44,
           45,
           47,
           48,
           50],
          'max_features': ['sqrt'],
          'max_depth': [2, 9, 17, 25, 33, 41, 48, 56, 64, 72, 80, None],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 2, 4],
          'bootstrap': [True, False],
          'random_state': [42]}
```

```
In [45]: # Use the random grid to search for best hyperparameters
         # First create the base model to tune
         rf = RandomForestClassifier()
         # Random search of parameters, using 3 fold cross validation,
         # search across 100 different combinations, and use all available cores
         rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
                                        n_iter = 400, cv = 5, verbose=2, random_state=42,
                                        n_jobs = -1, scoring=make_scorer(roc_auc_score))

         # Fit the random search model
         rf_random.fit(X_train, labels_train)
```

```
Fitting 5 folds for each of 400 candidates, totalling 2000 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed:   19.3s
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed:   33.8s
[Parallel(n_jobs=-1)]: Done 1005 tasks      | elapsed:   53.9s
[Parallel(n_jobs=-1)]: Done 1450 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 1977 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 2000 out of 2000 | elapsed:  1.8min finished
/home/ravila/Software/miniconda/lib/python3.6/site-packages/sklearn/model_selection/_search.py
  DeprecationWarning)
```

```
Out[45]: RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criteri
                     max_depth=None, max_features='auto', 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, n_estimators='warn', n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False),
                   fit_params=None, iid='warn', n_iter=400, n_jobs=-1,
                   param_distributions={'n_estimators': [11, 12, 13, 15, 16, 17, 19, 20, 21, 2
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score='warn', scoring=make_scorer(roc_auc_score),
                   verbose=2)
```

```
In [46]: rf_random.best_params_
```

```
Out[46]: {'random_state': 42,
          'n_estimators': 12,
          'min_samples_split': 5,
          'min_samples_leaf': 1,
          'max_features': 'sqrt',
          'max_depth': 56,
          'bootstrap': True}
```

```
In [47]: rf_random.best_score_
```

```
Out[47]: 0.64938186813186816
```

Default model

```
In [50]: base_model = RandomForestClassifier(n_estimators = 10, random_state = 42)
         base_model.fit(X_train, labels_train)
         pred = base_model.predict(X_test)
         balanced_accuracy = balanced_accuracy_score(labels_test, pred)
         accuracy = accuracy_score(labels_test, pred)
         auc = roc_auc_score(y_true=labels_test, y_score=[j for i, j in base_model.predict_prob
         print("Accuracy_score:", accuracy)
         print("Balanced accuracy score:", balanced_accuracy)
         print("AUC score:", auc)
```

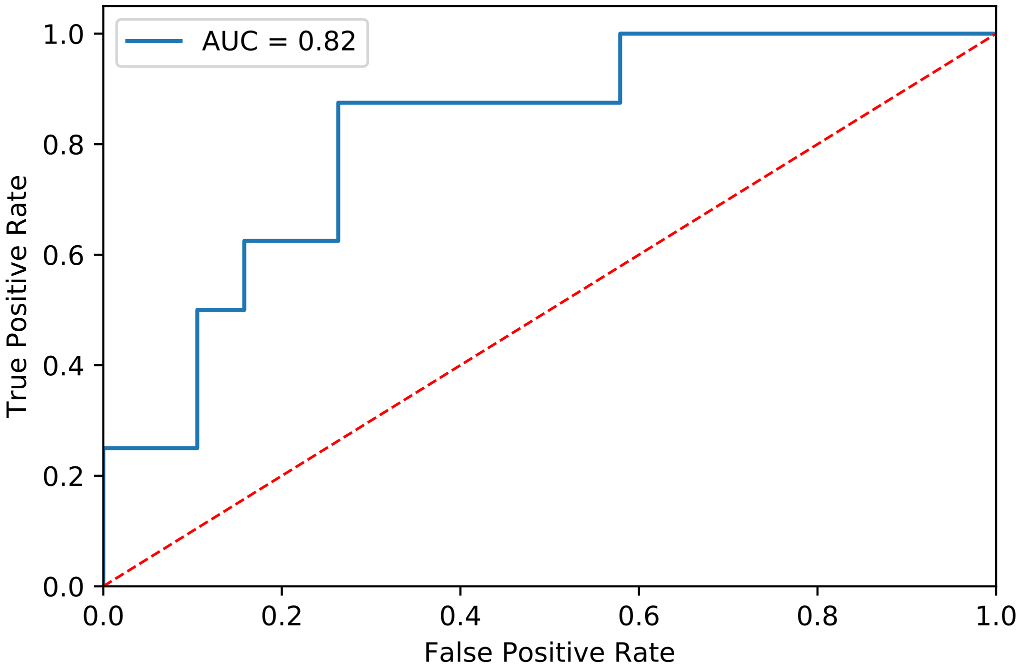
```
Accuracy_score: 0.851851851852
Balanced accuracy score: 0.75
AUC score: 0.799342105263
```

```
In [51]: rf_best = rf_random.best_estimator_
         rf_best.fit(X_train, labels_train)
         pred = rf_best.predict(X_test)

         balanced_accuracy = balanced_accuracy_score(labels_test, pred)
         accuracy = accuracy_score(labels_test, pred)
         auc = roc_auc_score(y_true=labels_test, y_score=[j for i, j in rf_best.predict_proba(X
         print("Accuracy_score:", accuracy)
         print("Balanced accuracy score:", balanced_accuracy)
         print("AUC score:", auc)
```

```
Accuracy_score: 0.777777777778
Balanced accuracy score: 0.625
AUC score: 0.815789473684
```

```
In [59]: # Plot an ROC curve
         fpr, tpr, _ = roc_curve(y_true=labels_test, y_score=[j for i, j in rf_best.predict_pro
         auc = roc_auc_score(y_true=labels_test, y_score=[j for i, j in rf_best.predict_proba(X
         plt.plot(fpr, tpr, label="AUC = {0:.2f}".format(auc))
         plt.plot([0, 1], [0, 1], color='red', lw=1, linestyle='--')
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend()
         plt.show()
```


AUC = 0.82
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0