D.J. Anderson dra2zp 10/11/2017

In-Class Activity 5

Source Code:

```
#

#==============================

#OOB Errors for Random Forests

#==============================

#

#The ``RandomForestClassifier`` is trained using *bootstrap
      aggregation*, where

#each new tree is fit from a bootstrap sample of the training
      observations

#:math:`z_i = (x_i, y_i)`. The *out-of-bag* (OOB) error is the average
      error for

#each :math:`z_i` calculated using predictions from the trees that do
      not

#contain :math:`z_i` in their respective bootstrap sample. This allows
      the

#``RandomForestClassifier`` to be fit and validated whilst being
      trained [1]_.

#

#The example below demonstrates how the OOB error can be measured at
      the

#addition of each new tree during training. The resulting plot allows
      a

#practitioner to approximate a suitable value of ``n_estimators`` at
      which the

#error stabilizes.

#

#.. [1] T. Hastie, R. Tibshirani and J. Friedman, "Elements of
      Statistical

#       Learning Ed. 2", p592-593, Springer, 2009.

#=================================================================

#
```

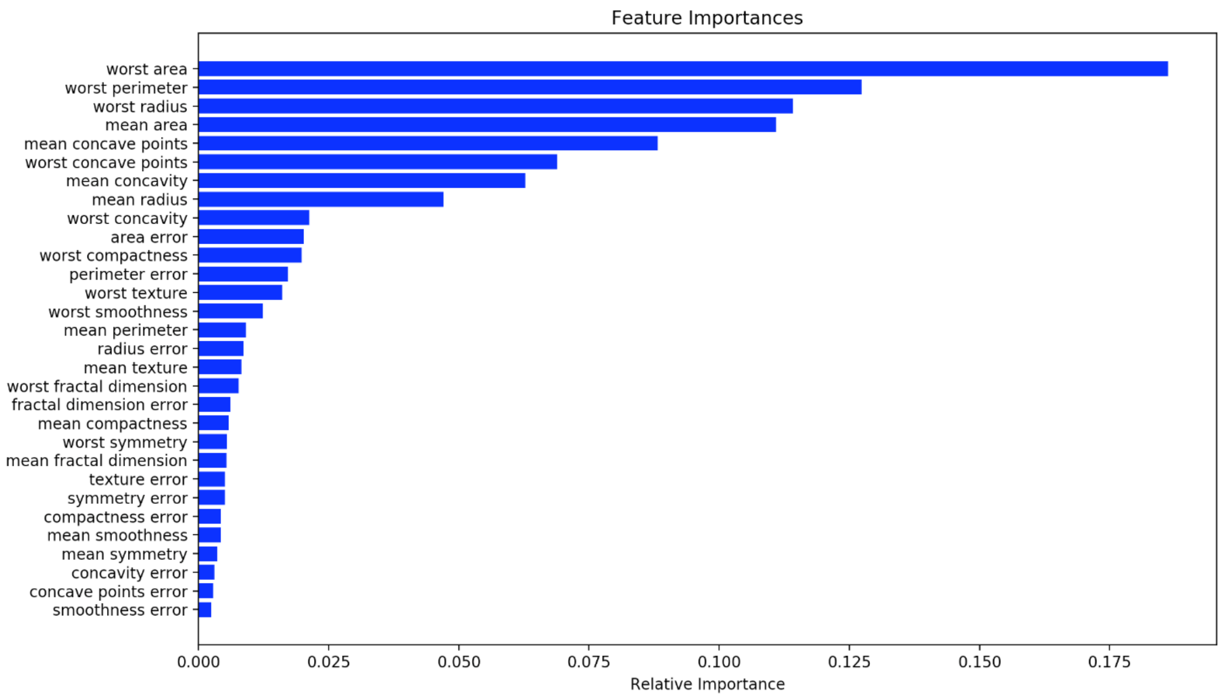
```
import matplotlib.pyplot as plt


from collections import OrderedDict

from sklearn.datasets import make_classification

from sklearn.ensemble import RandomForestClassifier,
     ExtraTreesClassifier


# Reference: scikit-learn.org


print(__doc__)


RANDOM_STATE = 123


# Generate a binary classification dataset.

X, y = make_classification(n_samples=500, n_features=25,

                           n_clusters_per_class=1, n_informative=15,

                           random_state=RANDOM_STATE)


# NOTE: Setting the `warm_start` construction parameter to `True`
     disables

# support for parallelized ensembles but is necessary for tracking the
     OOB

# error trajectory during training.

ensemble_clfs = [

    ("RandomForestClassifier, max_features='sqrt'",

        RandomForestClassifier(warm_start=True, oob_score=True,

                               max_features="sqrt",

                               random_state=RANDOM_STATE)),

    ("RandomForestClassifier, max_features='log2'",

        RandomForestClassifier(warm_start=True, max_features='log2',
                              oob_score=True,
                              random_state=RANDOM_STATE)),
    ("RandomForestClassifier, max_features=None",
        RandomForestClassifier(warm_start=True, max_features=None,
                              oob_score=True,
                              random_state=RANDOM_STATE))
]

# Map a classifier name to a list of (<n_estimators>, <error rate>)
     pairs.
error_rate = OrderedDict((label, []) for label, _ in ensemble_clfs)

# Range of `n_estimators` values to explore.
min_estimators = 15
max_estimators = 175

for label, clf in ensemble_clfs:
    for i in range(min_estimators, max_estimators + 1):
        clf.set_params(n_estimators=i)
        clf.fit(X, y)

        # Record the OOB error for each `n_estimators=i` setting.
        oob_error = 1 - clf.oob_score_
        error_rate[label].append((i, oob_error))

# Generate the "OOB error rate" vs. "n_estimators" plot.
for label, clf_err in error_rate.items():
    xs, ys = zip(*clf_err)
    plt.plot(xs, ys, label=label)

plt.xlim(min_estimators, max_estimators)
plt.title("How an appropriate number of features might be chosen")

plt.xlabel("n_estimators (number of trees)")

plt.ylabel("Out-Of-Bag (OOB) error rate")

plt.legend(loc="upper right")

plt.show()




## Modified from the example written by yhat that can be found here:

#http://blog.yhathq.com/posts/random-forests-in-python.html


from sklearn.ensemble import RandomForestClassifier as RFC

from sklearn.datasets import load_breast_cancer


import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


breast = load_breast_cancer()

df = pd.DataFrame(breast.data, columns=breast.feature_names)

df['is_train'] = np.random.uniform(0, 1, len(df)) <= .75

df['species'] = pd.Categorical.from_codes(breast.target,
     breast.target_names)


train, test = df[df['is_train']==True], df[df['is_train']==False]

f1 = df.columns[26:27]

f2 = df.columns[25:26]

f3 = df.columns[23:24]

f4 = df.columns[6:7]

f5 = df.columns[10:11]

# features = f1
# features = f2

# features = f3

# features = f4

# features = f5

forest = RFC(n_jobs=2,n_estimators=50)

y, _ = pd.factorize(train['species'])

forest.fit(train[features], y)


preds = breast.target_names[forest.predict(test[features])]

print (pd.crosstab(index=test['species'], columns=preds,
     rownames=['actual'], colnames=['preds']))


importances = forest.feature_importances_

indices = np.argsort(importances)


plt.figure(1)

plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color='b',
     align='center')

plt.yticks(range(len(indices)), features[indices])

plt.xlabel('Relative Importance')

plt.show()
```

Questions:

1 We used the breast cancer data set from http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29.

2 The original data set has 32 features.

3



4 (Removed all the columns except worst area, worst perimeter, worst radius, mean area, and mean concave points. See source code above.)

5 The model performed better since we only chose to keep the features that impacted the result (malignant vs. benign) the most, and we didn't remove too many features.