Feature Selection Using Random Forest

20 Dec 2017

Often in data science we have hundreds or even millions of features and we want a way to create a model that only includes the most important features. This has three benefits. First, we make our model more simple to interpret. Second, we can reduce the variance of the model, and therefore overfitting. Finally, we can reduce the computational cost (and time) of training a model. The process of identifying only the most relevant features is called "feature selection."

Random Forests are often used for feature selection in a data science workflow. The reason is because the tree-based strategies used by random forests naturally ranks by how well they improve the purity of the node. This mean decrease in impurity over all trees (called gini impurity). Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees. Thus, by pruning trees below a particular node, we can create a subset of the most important features.

In this tutorial we will:

1. Prepare the dataset
2. Train a random forest classifier
3. Identify the most important features
4. Create a new 'limited featured' dataset containing only those features
5. Train a second classifier on this new dataset

Note: There are other definitions of importance, however in this tutorial we limit our discussion to gini importance.

Preliminaries

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
```

Create The Data

The dataset used in this tutorial is the famous iris dataset. The Iris target data contains 50 samples from three species of Iris, `y` and four feature variables, `X`.

```
# Load the iris dataset
iris = datasets.load_iris()

# Create a list of feature names
feat_labels = ['Sepal Length','Sepal Width','Petal Length','Petal Width']

# Create X from the features
X = iris.data

# Create y from output
y = iris.target
```

```
# View the features
X[0:5]
```

```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

```
# View the target data
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Split The Data Into Training And Test Sets

```
# Split the data into 40% test and 60% training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
```

Train A Random Forest Classifier

```
# Train the classifier
clf.fit(X_train, y_train)

# Print the name and gini importance of each feature
for feature in zip(feat_labels, clf.feature_importances_):
    print(feature)
```

```
('Sepal Length', 0.11024282328064565)
('Sepal Width', 0.016255033655398394)
('Petal Length', 0.45028123999239533)
('Petal Width', 0.42322090307156124)
```

The scores above are the importance scores for each variable. There are two things to note. First, all the importance scores add up to 100%. Second, `Petal Length` and `Petal Width` are far more important than the other two features. Combined, `Petal Length` and `Petal Width` have an importance of ~0.86! Clearly these are the most importance features.

Identify And Select Most Important Features

```
# Create a selector object that will use the random forest classifier to identify
# features that have an importance of more than 0.15
sfm = SelectFromModel(clf, threshold=0.15)

# Train the selector
sfm.fit(X_train, y_train)
```

```
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10000, n_jobs=-1, oob_score=False, random_state=0,
            verbose=0, warm_start=False),
        prefit=False, threshold=0.15)
```

```
# Print the names of the most important features
for feature_list_index in sfm.get_support(indices=True):
    print(feat_labels[feature_list_index])
```

```
Petal Length
Petal Width
```

Create A Data Subset With Only The Most Important Features

```
# Transform the data to create a new dataset containing only the most important features
# Note: We have to apply the transform to both the training X and test X data.
X_important_train = sfm.transform(X_train)
X_important_test = sfm.transform(X_test)
```

Train A New Random Forest Classifier Using Only Most Important Features

```
# Train the new classifier on the new dataset containing the most important features
clf_important.fit(X_important_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10000, n_jobs=-1, oob_score=False, random_state=0,
            verbose=0, warm_start=False)
```

Compare The Accuracy Of Our Full Feature Classifier To Our Limited Feature Classifier

```
# Apply The Full Featured Classifier To The Test Data
y_pred = clf.predict(X_test)

# View The Accuracy Of Our Full Feature (4 Features) Model
accuracy_score(y_test, y_pred)
```

```
0.93333333333333335
```

```
# Apply The Full Featured Classifier To The Test Data
y_important_pred = clf_important.predict(X_important_test)

# View The Accuracy Of Our Limited Feature (2 Features) Model
accuracy_score(y_test, y_important_pred)
```

As can be seen by the accuracy scores, our original model which contained all four features is 93.3% accurate while the our 'limited' model which contained only two features is 88.3% accurate. Thus, for a small cost in accuracy we halved the number of features in the model.

Find an error or bug?

Everything on this site is available on GitHub. Head to and submit a change.