

Feature Importance and Feature Selection With XGBoost in Python

A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model.

In this post you will discover how you can estimate the importance of features for a predictive modeling problem using the XGBoost library in Python.

After reading this post you will know:

- How feature importance is calculated using the gradient boosting algorithm.
- How to plot feature importance in Python calculated by the XGBoost model.
- How to use feature importance calculated by XGBoost to perform feature selection.

Let's get started.

- **Update Jan/2017:** Updated to reflect changes in scikit-learn API version 0.18.1.
- **Update March/2018:** Added alternate link to download the dataset as the original appears to have been taken down.

Feature Importance in Gradient Boosting

A benefit of using gradient boosting is that after the boosted trees are constructed, it is relatively straightforward to retrieve importance scores for each attribute.

Generally, importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance.

This importance is calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared to each other.

Importance is calculated for a single decision tree by the amount that each attribute split point improves the performance measure, weighted by the number of observations the node is responsible for. The performance measure may be the purity (Gini index) used to select the split points or another more specific error function.

The feature importances are then averaged across all of the the decision trees within the model.

For more technical information on how feature importance is calculated in boosted decision trees, see Section 10.13.1 “*Relative Importance of Predictor Variables*” of the book [The Elements of Statistical Learning: Data Mining, Inference, and Prediction](#), page 367.

Also, see Matthew Drury answer to the StackOverflow question “[Relative variable importance for Boosting](#)” where he provides a very detailed and practical answer.

Manually Plot Feature Importance

A trained XGBoost model automatically calculates feature importance on your predictive modeling problem.

These importance scores are available in the **feature_importances_** member variable of the trained model. For example, they can be printed directly as follows:

```
1 print(model.feature_importances_)
```

We can plot these scores on a bar chart directly to get a visual indication of the relative importance of each feature in the dataset. For example:

```
1 # plot
2 pyplot.bar(range(len(model.feature_importances_)), model.feature_importances_)
3 pyplot.show()
```

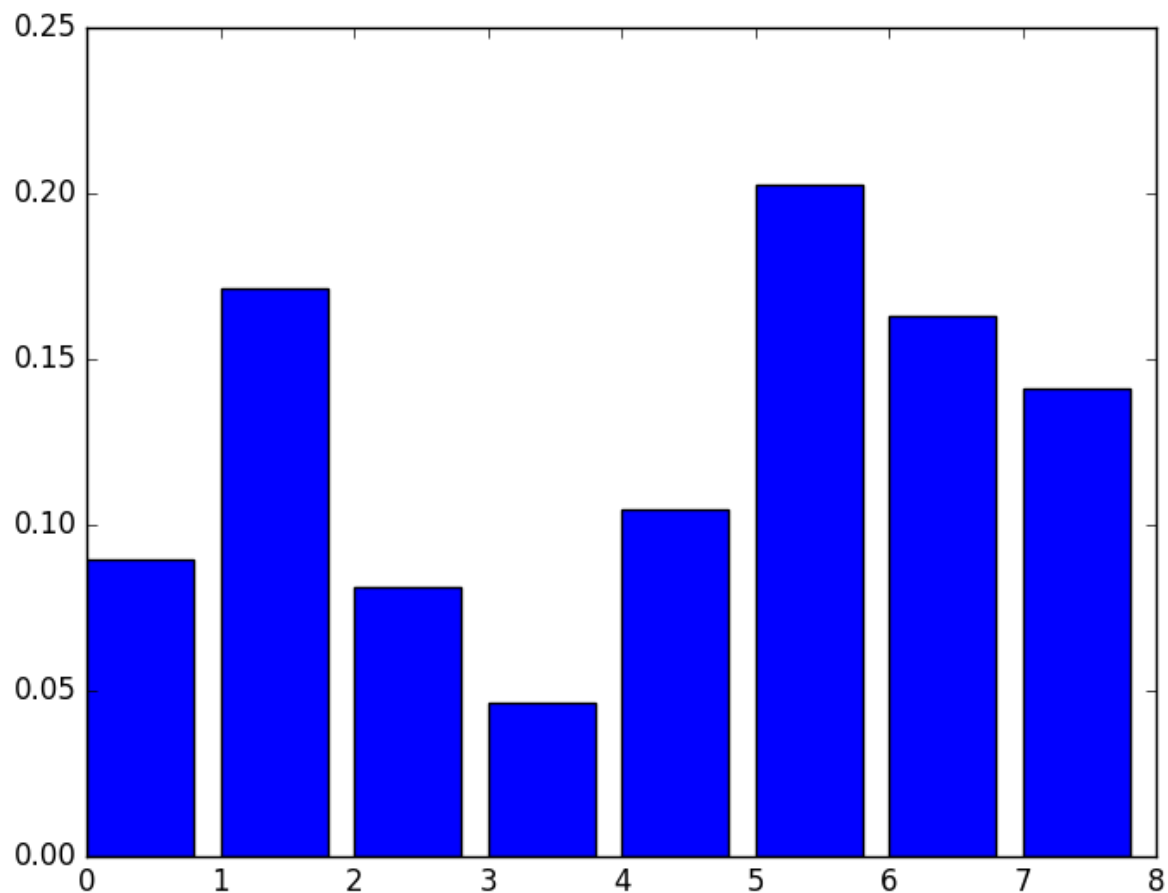
We can demonstrate this by training an XGBoost model on the [Pima Indians onset of diabetes dataset](#) and creating a bar chart from the calculated feature importances (update: [download from here](#)).

```
1 # plot feature importance manually
2 from numpy import loadtxt
3 from xgboost import XGBClassifier
4 from matplotlib import pyplot
5 # load data
6 dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
7 # split data into X and y
8 X = dataset[:,0:8]
9 y = dataset[:,8]
10 # fit model no training data
11 model = XGBClassifier()
12 model.fit(X, y)
13 # feature importance
14 print(model.feature_importances_)
15 # plot
16 pyplot.bar(range(len(model.feature_importances_)), model.feature_importances_)
17 pyplot.show()
```

Running this example first outputs the importance scores:

```
1 [ 0.089701  0.17109634  0.08139535  0.04651163  0.10465116  0.2026578  0.1627907  0.141196]
```

We also get a bar chart of the relative importances.



Manual Bar Chart of XGBoost Feature Importance

A downside of this plot is that the features are ordered by their input index rather than their importance. We could sort the features before plotting.

Thankfully, there is a built in plot function to help us.

Using the Built-in XGBoost Feature Importance Plot

The XGBoost library provides a built-in function to plot features ordered by their importance.

The function is called `plot_importance()` and can be used as follows:

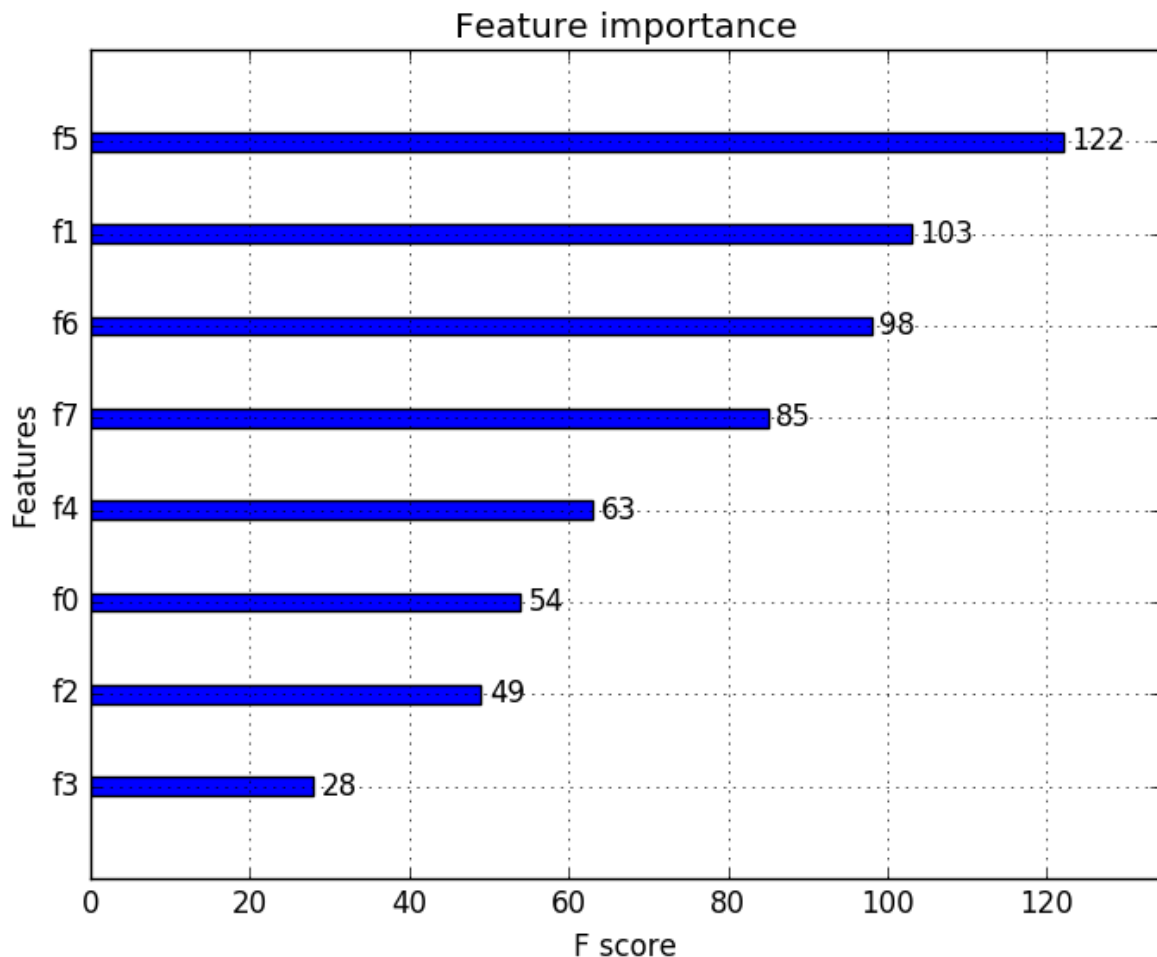
```
1 # plot feature importance
2 plot_importance(model)
3 pyplot.show()
```

For example, below is a complete code listing plotting the feature importance for the Pima Indians dataset using the built-in `plot_importance()` function.

```
1 # plot feature importance using built-in function
2 from numpy import loadtxt
3 from xgboost import XGBClassifier
4 from xgboost import plot_importance
5 from matplotlib import pyplot
6 # load data
7 dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
8 # split data into X and y
```

```
9 X = dataset[:,0:8]
10 y = dataset[:,8]
11 # fit model no training data
12 model = XGBClassifier()
13 model.fit(X, y)
14 # plot feature importance
15 plot_importance(model)
16 pyplot.show()
```

Running the example gives us a more useful bar chart.



XGBoost Feature Importance Bar Chart

You can see that features are automatically named according to their index in the input array (X) from F0 to F7.

Manually mapping these indices to [names in the problem description](#), we can see that the plot shows F5 (body mass index) has the highest importance and F3 (skin fold thickness) has the lowest importance.

Feature Selection with XGBoost Feature Importance Scores

Feature importance scores can be used for feature selection in scikit-learn.

This is done using the [SelectFromModel](#) class that takes a model and can transform a dataset into a subset with selected features.

This class can take a pre-trained model, such as one trained on the entire training dataset. It can then use a threshold to decide which features to select. This threshold is used when you call the **transform()** method on the **SelectFromModel** instance to consistently select the same features on the training dataset and the test dataset.

In the example below we first train and then evaluate an XGBoost model on the entire training dataset and test datasets respectively.

Using the feature importances calculated from the training dataset, we then wrap the model in a **SelectFromModel** instance. We use this to select features on the training dataset, train a model from the selected subset of features, then evaluate the model on the testset, subject to the same feature selection scheme.

For example:

```
1 # select features using threshold
2 selection = SelectFromModel(model, threshold=thresh, prefit=True)
3 select_X_train = selection.transform(X_train)
4 # train model
5 selection_model = XGBClassifier()
6 selection_model.fit(select_X_train, y_train)
7 # eval model
8 select_X_test = selection.transform(X_test)
9 y_pred = selection_model.predict(select_X_test)
```

For interest, we can test multiple thresholds for selecting features by feature importance. Specifically, the feature importance of each input variable, essentially allowing us to test each subset of features by importance, starting with all features and ending with a subset with the most important feature.

The complete code listing is provided below.

```
1 # use feature importance for feature selection
2 from numpy import loadtxt
3 from numpy import sort
4 from xgboost import XGBClassifier
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import accuracy_score
7 from sklearn.feature_selection import SelectFromModel
8 # load data
9 dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
10 # split data into X and y
11 X = dataset[:,0:8]
12 Y = dataset[:,8]
13 # split data into train and test sets
14 X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
15 # fit model on all training data
16 model = XGBClassifier()
17 model.fit(X_train, y_train)
18 # make predictions for test data and evaluate
19 y_pred = model.predict(X_test)
20 predictions = [round(value) for value in y_pred]
21 accuracy = accuracy_score(y_test, predictions)
22 print("Accuracy: %.2f%%" % (accuracy * 100.0))
23 # Fit model using each importance as a threshold
24 thresholds = sort(model.feature_importances_)
25 for thresh in thresholds:
26     # select features using threshold
27     selection = SelectFromModel(model, threshold=thresh, prefit=True)
28     select_X_train = selection.transform(X_train)
29     # train model
30     selection_model = XGBClassifier()
```

```
31 selection_model.fit(select_X_train, y_train)
32 # eval model
33 select_X_test = selection.transform(X_test)
34 y_pred = selection_model.predict(select_X_test)
35 predictions = [round(value) for value in y_pred]
36 accuracy = accuracy_score(y_test, predictions)
37 print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (thresh, select_X_train.shape[1], accuracy))
```

Running this example prints the following output:

```
1 Accuracy: 77.95%
2 Thresh=0.071, n=8, Accuracy: 77.95%
3 Thresh=0.073, n=7, Accuracy: 76.38%
4 Thresh=0.084, n=6, Accuracy: 77.56%
5 Thresh=0.090, n=5, Accuracy: 76.38%
6 Thresh=0.128, n=4, Accuracy: 76.38%
7 Thresh=0.160, n=3, Accuracy: 74.80%
8 Thresh=0.186, n=2, Accuracy: 71.65%
9 Thresh=0.208, n=1, Accuracy: 63.78%
```

We can see that the performance of the model generally decreases with the number of selected features.

On this problem there is a trade-off of features to test set accuracy and we could decide to take a less complex model (fewer attributes such as $n=4$) and accept a modest decrease in estimated accuracy from 77.95% down to 76.38%.

This is likely to be a wash on such a small dataset, but may be a more useful strategy on a larger dataset and using cross validation as the model evaluation scheme.

Summary

In this post you discovered how to access features and use importance in a trained XGBoost gradient boosting model.

Specifically, you learned:

- What feature importance is and generally how it is calculated in XGBoost.
- How to access and plot feature importance scores from an XGBoost model.
- How to use feature importance from an XGBoost model for feature selection.

Do you have any questions about feature importance in XGBoost or about this post? Ask your questions in the comments and I will do my best to answer them.