SOCIETY OF ACTUARIES

Article from:

Forecasting & Futurism

December 2014 – Issue 10

Modeling With Python And Scikit-Learn

By Jeff Heaton

R and Python are the two most popular computer languages for data science, as reported by a 2013 KDNuggets survey. (KDNuggets, 2013) Both R and Python have a variety of data science frameworks available for them. These frameworks standardize the implementations of the many different models that data scientists use. This article will introduce the Scikit-learn (http://scikit-learn.org/) package for Python. (Pedregosa, et al., 2011) A similar package, called CARET (http://topepo.github.io/caret/index.html) is available for the R programming language. (Kuhn, 2008)

Scikit-learn is an open source machine-learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines (SVM), logistic regression, naive Bayes, random forests, gradient boosting machines (GBM) and k-means. Scikit-learn is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. The project's original codebase was later extensively rewritten by other developers. Scikit-learn is under active development and is sponsored by INRIA[1] and occasionally Google. Scikit-learn was used for a number of successful Kaggle (http://www.kaggle.com) competitions.

Kaggle is a platform for competitive data science that allows the top data scientists from around the world to compete on predictive accuracy. Kaggle has hosted a number of competitions of interest to the insurance industry. Allstate hosted a purchase prediction challenge, Liberty Mutual hosted a fire-loss challenge, and Practice Fusion hosted a challenge to predict diabetes in patients. The next article in this series will demonstrate scikit-learn modeling with data from a Kaggle competition.

INSTALLING SCIKIT-LEARN

There are many things to love about Python. However, multiple Python versions and installing new packages is not one of those things. Despite Python 3 being released in 2008, as of 2014 Python 2.x still has an active following. Backwards compatibility was severely broken when the switch to Python 3 occurred. Whenever using a Python example, it is very important to understand if you are using Python 3.x or Python 2.x code. The code in this article will work with either Python 3.x or Python 2.x.

Package management presents its own unique challenge in Python. Pure python packages can be installed with one of several package managers such as "pip" or "easy_install." Unfortunately, many of the Python packages contain compiled code based on C/C++ or even Fortran. This is the case with scikit-learn and some of the numerical packages it depends on. Fully describing how to install scikit-learn in Python 3 is beyond the scope of this article. I wrote a setup guide for Python that can be found at the following URL.

http://goo.gl/l94xQG

SCIKIT-LEARN BASIC LINEAR REGRESSION

Scikit-learn makes it very easy to switch between different model types. To start, consider a model to convert between Celsius and Fahrenheit temperatures.

```
x = [
 [-40],
 [10],
 [25],
 [30]
 ]
y = [-40,50,77,86]
```

The x-values represent the Fahrenheit input to the model, and the y-values represent the Celsius expected output. This is univariate data; it is also possible to use multivariate input data, as seen here.

```
x = [
 [1,2],
 [3,4],
 [5,6],
 [7,8]]
```

The above input contains two observations, or features, per sample.

For this example, we will use the first, univariate data set. Once we've defined the input and expected output for each observation it is easy to create and fit a model. The following code will fit a linear regression model.

```
from sklearn import datasets, linear_
model
model = linear_model.LinearRegres-
sion()
model.fit(x, y)
```

Now that the model is setup, we can query it with the "predict" command. To find out the Fahrenheit temperature for Celsius 10, use the following command.

```
print(model.predict(10))
```

The model will respond with 50 degrees. We can also predict a value not in the data set.

```
print(model.predict(15))
```

The model should return with 59 degrees.

We can also display the coefficients, RSS and variance for this linear regression using the following commands.

```
import numpy as np
# The coefficients
print('Coefficients: \n', model.coef_)

# The mean square error
print ("Residual sum of squares: %.2f"
%
    np.mean((model.predict(x) - y) **
2))

# Explained variance score: 1 is per-
fect prediction
print ('R^2 score: %.2f' % model.
score(x, y))
```

This results in the following output.

```
('Coefficients: \n', array([[ 1.8]]))
Residual sum of squares: 0.00
R^2: 1.00
```

Of course, the temperature conversion fit perfectly, so the RSS is zero and the R squared is 1.0.

SCIKIT-LEARN WITH A DECISION TREE

What if we wanted to use exactly the same data, only use a CART decision tree? Scikit-learn makes this very easy. The following code fits the temperature data using a regression decision tree.

CONTINUED ON **PAGE 38**

```
from sklearn import tree

model = tree.DecisionTreeRegressor()
model.fit(x, y)
print(model.predict(15))
```

Notice that the code is nearly the same? We always use the "fit" command to fit the model. Likewise, we always use the "predict" command to perform a prediction.

VISUALIZING A DECISION TREE

Scikit-learn allows visualizations of some of the model types. Decision trees are a model type that is particularly easy to visualize. To see how to visualize a decision tree, consider the following highly contrived data.

These observations represent individuals that applied for a particular type of auto insurance. The x vector contains their ages in the first column, gender in the second, and so on. For gender, one means male, and zero female. Martial has a value of one for married, or zero for single. The DUI and accident columns contain counts of each infraction. Finally, the y-vector contains a one for insured and a zero for declined.

```
features = ['age','gender',"marital",
"dui", "accident"]
x = [
  [16,0,0,0,0],
  [21,1,0,0,1],
  [42,0,1,0,0],
  [16,1,0,2,2],
  [34,0,1,0,1],
  [55,1,1,1,0]
]


y = [1,0,1,0,1,1]
```

Fitting this to a decision tree is easy enough, with the following commands.

```
model = tree.DecisionTreeClassifier()
model.fit(x, y)
```

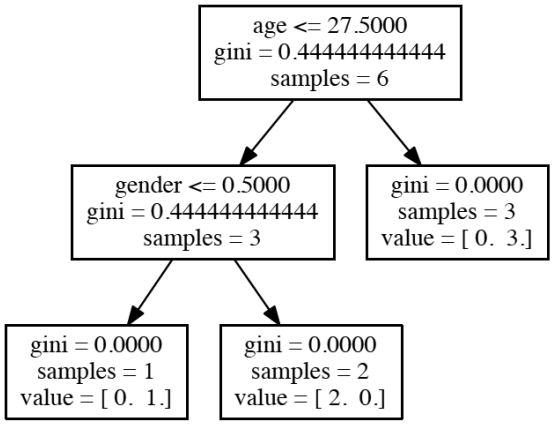
To visualize this as a tree, we use the following commands.

There are a number of "overhead" commands in the above code sequence. The main part to understand is that your tree will be written to "tree_pdf."

```
from sklearn.externals.six import
StringIO
import pydot
dot_data = StringIO()
tree.export_graphviz(model, out_
file=dot_data,feature_names=features)
graph = pydot.graph_from_dot_data(dot_
data.getvalue())
graph.write_pdf("tree.pdf")
```

This will result in the following tree.

The scikit-learn tree can be difficult to read, until you understand its format. Each node contains a single binary decision. If the condition is false, the tree will proceed to the left, similarly, the tree will proceed to the right if the condition is true.



The number of samples that support each tree node is displayed. As the tree descends, and specializes, the number of samples will decrease. Likewise, the Gini value should decrease as the tree specializes. Gini is specific to the CART algorithm and acts as a loss function to minimize.

Perhaps the most confusing line to understand is the "value." Only final decision nodes (leafs) will contain a value. Because we were classifying into two sets, there will always be two values in the "value" array.

The first number in the value array specifies the number of class 0, or decline, samples. The second number in the value array specifies the number of class 1, or accepts, samples. Ideally, only one of these has a value, and the other is zero. If this is the case, the Gini has the optimal value of zero. The output of the tree is usually interpreted to be the decision node's value with the most number of samples. For example, the right-most node on the above tree would output (be accept is this correct?), because there were three accept samples, and no decline samples.

SCIKIT-LEARN WITH OTHER MODEL TYPES

Scikit-learn supports many different model types. The following code would make use of a random forest.

```
from sklearn.ensemble import Random-
ForestClassifier
model = tree.DecisionTreeRegressor()
model.fit(x, y)
print(model.predict(15))
```

Similarly, the following code would make use of a Gradient Boosting Machine (GBM).

```
from sklearn import ensemble
model = ensemble.GradientBoostingRe-
gressor()
model.fit(x, y)
print(model.predict(15))
```

These advanced machine-learning models are overkill for this very simple linear data setup. For this completely linear, noiseless data set, the linear regression model is actually the most accurate.

OTHER FEATURES OF SCIKIT-LEARN

Scikit-learn includes many other features that assist in modeling. Model selection can be automated by trying many different model parameters. This slow process can be sped up using multiple processing cores on your computer. Scikit-learn also contains functions for feature selection, normalization, dimensionality reduction and many other common modeling tasks.

The next article in this series, titled "Titanic Pythonic Mortality Modeling" will look at the Kaggle dataset for the Titanic. This is a very simple mortality question, given statistics about the passengers, how accurately can we predict who survives and who perishes. The next article will demonstrate using Python and scikit-learn to accumulate, prepress and then model the Titanic data set.

CONTINUED ON **PAGE 40**

MODELING WITH PYTHON … | FROM PAGE 39

REFERENCES

KDNuggets. (2013, 8 1). *Languages used for analytics / data mining / data science* . Retrieved 7 31, 2014, from http://www.kdnuggets.com/polls/2013/languages-analytics-data-mining-data-science.html

Kuhn, M. (2008). Building Predictive Models in R Using the caret Package. *Journal of Statistical Software , 28* (5).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011, 12). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* , 2825--2830.

Jeff Heaton, is EHR data scientist at RGA Reinsurance Company and author of several books on artificial intelligence. He can be reached at jheaton@rgare.com ▼

Jeff Heaton

ENDNOTES

1 The **French Institute for Research in Computer Science and Automation** (French: Institut national de recherche en informatique et en automatique, **INRIA**) is a French national research institution focusing on computer science and applied mathematics.

Jeff Heaton, is data scientist at RGA Reinsurance Company and author of several books on artificial intelligence. He can be reached at *jheaton@rgare.com.*