June 08, 2015

Decision trees in python with scikit-learn and pandas

In this post I will cover decision trees (for classification) in python, using scikit-learn and pandas. The emphasis will be on the basics and understanding the resulting decision tree. I will cover:

- Importing a csv file using pandas,
- Using pandas to prep the data for the scikit-leaarn decision tree code,
- Drawing the tree, and
- Producing pseudocode that represents the tree.

The last two parts will go over what the tree has actually found– this is one of the really nice parts of a decision tree: the findings can be inspected and we can learn something about the patterns in our data. If this sounds interesting to you, read on. Also, if you have other ideas about how to do related things please leave comments below!

gist

Before we get going, the code is **available as a gist (https://gist.github.com/cstrelioff/8fefa9a43e82d96e9f0c)**, so you don't have to copy and paste (unless you want to). I'll go through the functions and usage from scratch here– usage of the gist code is detailed there in a README file.

imports

So, first we do some imports, including the `print_function` for python3-style print statements. I also import the usual suspects, using common abbreviations, which I'll discuss below:

```
from __future__ import print_function

import os
import subprocess

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, export_graphviz
```

data with pandas

Next, we need some data to consider. I'll use the famous iris data set, that has various measurements for a variety of different iris types. I think both pandas and sckit-learn have easy import options for this data, but I'm going to write a function to import from a csv file, using pandas. The point of this to demonstrate how pandas can be used with scikit-learn. So, we define a function for getting the iris data:

```
def get_iris_data():
    """Get the iris data, from local csv or pandas repo."""
    if os.path.exists("iris.csv"):
        print("-- iris.csv found locally")
        df = pd.read_csv("iris.csv", index_col=0)
    else:
        print("-- trying to download from github")
        fn = "https://raw.githubusercontent.com/pydata/pandas/" + \
             "master/pandas/tests/data/iris.csv"
        try:
            df = pd.read_csv(fn)
        except:
            exit("-- Unable to download iris.csv")

        with open("iris.csv", 'w') as f:
            print("-- writing to local iris.csv file")
            df.to_csv(f)

    return df
```

Notes:

- This function first tries to read the data locally, using pandas. This is why I import `os` above: to make use of the `os.path.exists()` method. If the **iris.csv** file is found in the local directory, pandas is used to read the file using `pd.read_csv()` – note that pandas has been import using `import pandas as pd`. This is typical usage for the package.
- If a local **iris.csv** is *not found*, pandas is used to grab the data from a url and a local copy is saved for future runs. A `try` and `except` are used to exit and provide a note if there are problems– maybe the user is not connected to the internet?

Hopefully the above codes gives a sense of how to load a csv data file, locally as well as from a remote location. The next step is to get the data and use the `head()` and `tail()` methods to see what the data is like– these show the start and end of the dataframe, respectively. So, first get the data:

```
df = get_iris_data()
```

```
-- iris.csv found locally
```

then, head and tail:

```
print("* df.head()", df.head(), sep="\n", end="\n\n")
print("* df.tail()", df.tail(), sep="\n", end="\n\n")
```

```
* df.head()
   SepalLength  SepalWidth  PetalLength  PetalWidth         Name
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa

* df.tail()
     SepalLength  SepalWidth  PetalLength  PetalWidth            Name
145          6.7         3.0          5.2         2.3  Iris-virginica
146          6.3         2.5          5.0         1.9  Iris-virginica
147          6.5         3.0          5.2         2.0  Iris-virginica
148          6.2         3.4          5.4         2.3  Iris-virginica
149          5.9         3.0          5.1         1.8  Iris-virginica
```

From this information we can talk about our goal: to predict **Name** (or, type of iris) given the features **SepalLength**, **SepalWidth**, **PetalLength** and **PetalWidth**. We can use pandas to show the three iris types:

```
print("* iris types:", df["Name"].unique(), sep="\n")
```

```
* iris types:
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

preprocessing

In order to pass this data into scikit-learn we need to encode the **Names** to integers. To do this we'll write another function and return the modified data frame as well as a list of the target (class) names:

```
def encode_target(df, target_column):
    """Add column to df with integers for the target.

    Args
    ----
    df -- pandas DataFrame.
    target_column -- column to map to int, producing
                     new Target column.

    Returns
    -------
    df_mod -- modified DataFrame.
    targets -- list of target names.
    """
    df_mod = df.copy()
    targets = df_mod[target_column].unique()
    map_to_int = {name: n for n, name in enumerate(targets)}
    df_mod["Target"] = df_mod[target_column].replace(map_to_int)

    return (df_mod, targets)
```

Let's see what we have (I'll show just **Name** and **Target** columns to prevent wrapping):

```
df2, targets = encode_target(df, "Name")
print("* df2.head()", df2[["Target", "Name"]].head(),
      sep="\n", end="\n\n")
print("* df2.tail()", df2[["Target", "Name"]].tail(),
      sep="\n", end="\n\n")
print("* targets", targets, sep="\n", end="\n\n")
```

```
* df2.head()
   Target         Name
0       0  Iris-setosa
1       0  Iris-setosa
2       0  Iris-setosa
3       0  Iris-setosa
4       0  Iris-setosa

* df2.tail()
     Target            Name
145       2  Iris-virginica
146       2  Iris-virginica
147       2  Iris-virginica
148       2  Iris-virginica
149       2  Iris-virginica

* targets
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

Looks good, **Iris-setosa** has been mapped to zero, **Iris-versicolor** to one, and **Iris-virginica** to three. Next, we get the names of the feature columns:

```
features = list(df2.columns[:4])
print("* features:", features, sep="\n")
```

```
* features:
['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']
```

fitting the decision tree with scikit-learn

Now we can fit the decision tree, using the `DecisionTreeClassifier` imported above, as follows:

```
y = df2["Target"]
X = df2[features]
dt = DecisionTreeClassifier(min_samples_split=20, random_state=99)
dt.fit(X, y)
```

Notes:

- We pull the `X` and `y` data from the pandas dataframe using simple indexing.
- The decision tree, imported at the start of the post, is initialized with two parameters: `min_samples_split=20` requires 20 samples in a node for it to be split (this will make more sense when we see the result) and `random_state=99` to seed the random number generator.

visualizing the tree

We can produce a graphic (if **graphviz (http://www.graphviz.org/)** is available on your system– if not check the site and see if you can install) using the following function:

```
def visualize_tree(tree, feature_names):
    """Create tree png using graphviz.

    Args
    ----
    tree -- scikit-learn DecsisionTree.
    feature_names -- list of feature names.
    """
    with open("dt.dot", 'w') as f:
        export_graphviz(tree, out_file=f,
                        feature_names=feature_names)

    command = ["dot", "-Tpng", "dt.dot", "-o", "dt.png"]
    try:
        subprocess.check_call(command)
    except:
        exit("Could not run dot, ie graphviz, to "
             "produce visualization")
```

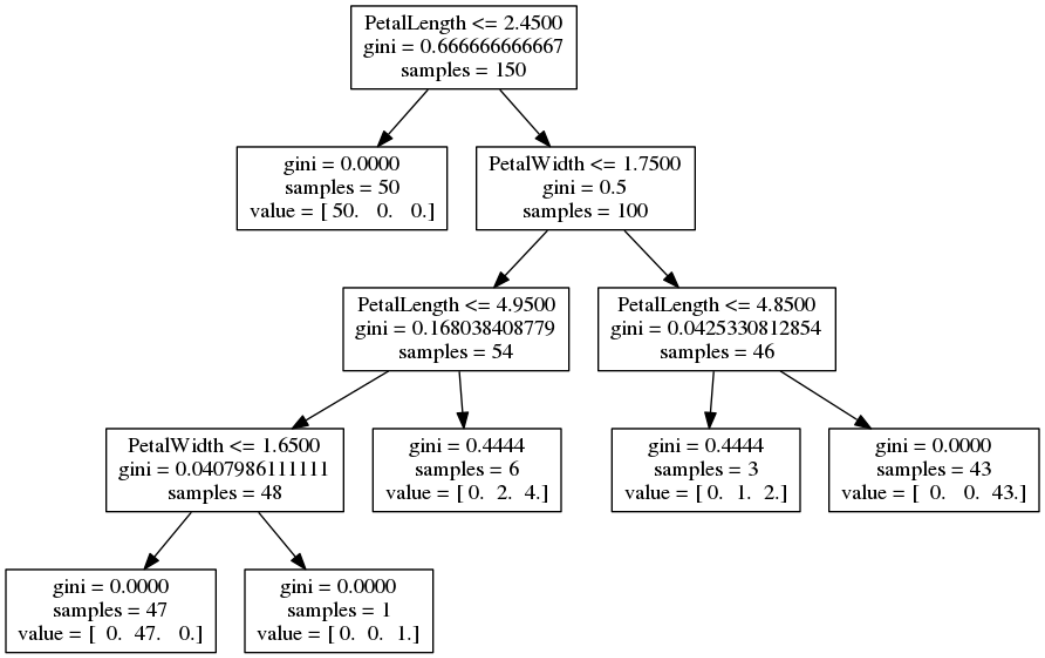
Notes:

- The `export_graphviz` method, imported from scikit-learn above, writes a dot file. This file is used to produce the graphic.
- `subprocess`, imported above, is used to process the dot file and generate the graphic **dt.png**– see the example below.

So, running the function:

```
visualize_tree(dt, features)
```

results in (click on the figure to see a larger version)



(../../../_images/dt.png)

Okay, what does this all mean? Well, we can use this figure to understand the patterns found by the decision tree:

- Imagine that all data (all rows) start in a *single bin* at the top of the tree.

- All features are considered to see how the data can be split in the most informative way– this uses the gini measure by default, but this can be changed to entropy if you prefer; see **decision tree classifier documentation (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)**.
- At the top we see the most informative condition is **PetalLength <= 2.4500**. If this condition is *true*, take the left branch to get to the 50 samples of **value = [50. 0. 0.]**. This means there are 50 examples of class/target 0, in this case **Iris-setosa**. Unfortunately, the default scikit-learn export to graphviz/dot does not seem to be able to include this information (but see below). The other 100 samples, of the 150 total, go to the right bin.
- This splitting continues until

    1. The split creates a bin with only one class– for example the bin with 50 Iris-setosa is not split again.
    2. Or, the resulting bin has less than 20 samples– this is because we set the `min_samples_split=20` when initializing the decision tree. If we **had not set this value**, the tree would keep splitting until all bins have a single class.

So, that's it for the visualization– you should be able to trace, from top to bottom, and see how the rules discussed above were applied to the iris data.

psuedocode for the decision tree

Finally let's consider generation of psuedocode that represents the learned decision tree. In particular, the target names (classes) and feature names should be included in the output so that it is simple to follow the patterns found. The function below is based on **the answer to a stackoverflow question (http://stackoverflow.com/a/30104792)**. I've made some additions to the function to meet the requirements I've stated above:

- The target names can be passed to the function and are included in the output. The output now shows both the features used for branching conditions as well as the class, or classes, found in the resulting node/bin.
- The **if/else** structure has indenting, using the `spacer_base` argument to make the output easier to read (I think).

That said, the function is:

```
def get_code(tree, feature_names, target_names,
             spacer_base="    "):
    """Produce psuedo-code for decision tree.

    Args
    ----
    tree -- scikit-leant DescisionTree.
    feature_names -- list of feature names.
    target_names -- list of target (class) names.
    spacer_base -- used for spacing code (default: "    ").

    Notes
    -----
    based on http://stackoverflow.com/a/30104792.
    """
    left      = tree.tree_.children_left
    right     = tree.tree_.children_right
    threshold = tree.tree_.threshold
    features  = [feature_names[i] for i in tree.tree_.feature]
    value = tree.tree_.value

    def recurse(left, right, threshold, features, node, depth):
        spacer = spacer_base * depth
        if (threshold[node] != -2):
            print(spacer + "if ( " + features[node] + " <= " + \
                  str(threshold[node]) + " ) {")
            if left[node] != -1:
                    recurse(left, right, threshold, features,
                            left[node], depth+1)
            print(spacer + "}\n" + spacer +"else {")
            if right[node] != -1:
                    recurse(left, right, threshold, features,
                            right[node], depth+1)
            print(spacer + "}")
        else:
            target = value[node]
            for i, v in zip(np.nonzero(target)[1],
                            target[np.nonzero(target)]):
                target_name = target_names[i]
                target_count = int(v)
                print(spacer + "return " + str(target_name) + \
                      " ( " + str(target_count) + " examples )")

    recurse(left, right, threshold, features, 0, 0)
```

and the resulting output for application to the iris data is:

```
get_code(dt, features, targets)
```

```
if ( PetalLength <= 2.45000004768 ) {
    return Iris-setosa ( 50 examples )
}
else {
    if ( PetalWidth <= 1.75 ) {
        if ( PetalLength <= 4.94999980927 ) {
            if ( PetalWidth <= 1.65000009537 ) {
                return Iris-versicolor ( 47 examples )
            }
            else {
                return Iris-virginica ( 1 examples )
            }
        }
        else {
            return Iris-versicolor ( 2 examples )
            return Iris-virginica ( 4 examples )
        }
    }
    else {
        if ( PetalLength <= 4.85000038147 ) {
            return Iris-versicolor ( 1 examples )
            return Iris-virginica ( 2 examples )
        }
        else {
            return Iris-virginica ( 43 examples )
        }
    }
}
```

This should be compared with the graphic output above– this is just a different representation of the learned decision tree. However, I think the addition of target/classes and features really make this useful.

Okay, that's it for this post. There are many topics I have not covered, but I think that I've provided some useful code for understanding a decision tree learned with scikit-learn. Useful links at the **scikit-learn (http://scikit-learn.org/stable/)** site, to dig deeper include:

- **decision tree classifier example (http://scikit-learn.org/stable/modules/tree.html#classification)** – a simple decision tree example.
- **decision tree classifier documentation (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier)** – documentation for the class. Be sure to check out the many parameters that can be set.
- **decision tree classifier plot boundaries (http://scikit-learn.org/stable/auto_examples/tree/plot_iris.html#example-tree-plot-iris-py)** – how to plot the decision boundaries for the iris data. Unfortunately, they *normalize* the data before training and plotting, resulting in *negative lengths*, which are very difficult to relate back to the original data.

Importantly, I have not covered how to set parameters and to avoid over fitting. However, that's beyond the scope of this post. The best place to start for this is the **cross-validation** tools in scikit-learn. Check out:

- **scikit-learn cross-validation (http://scikit-learn.org/stable/modules/cross_validation.html)** – there's even an example with the iris data to get you started!
- **scikit-learn grid search (http://scikit-learn.org/stable/modules/grid_search.html#grid-search)** – search through set of parameters to find the best setting(s).

As always, post comments and questions below. Corrections and typos are also welcomed!

Posted by Chris Strelioff
Tags: **python (../../../tags/python.html)**, **pandas (../../../tags/pandas.html)**, **scikit-learn**

(../../../tags/scikit_learn.html), **machine learning (../../../tags/machine_learning.html)**, **supervised learning (../../../tags/supervised_learning.html)**, **decision trees (../../../tags/decision_trees.html)**

« **PySoundFile and Python 3.4 on Ubuntu 14.04 (../16/pysoundfile_and_python_3_4_on_ubuntu_14_04.html)**

Revisiting the medical tests example with Python and Lea

(../../05/27/revisiting_the_medical_tests_example_with_python_and_lea.html) »

This comment was deleted.

Christopher Strelioff Mod → Guest • 9 days ago

Hi, I'm not sure what you want to do here. The function is meant to create the dt.png file using dot-- dot is not a python package and has no "show" or "display" method like matplotlib. However, if you really want to have the image pop up in a window you can use something like Pillow (imported as PIL below):

```
>>> from PIL import Image
>>> im = Image.open("dt.png")
>>> im.show()
```

You can find more information about Pillow here: http://bit.ly/1Qvi7Yd (including install instructions). I hope that helps!

Best, Chris

Brad Arvin • 4 months ago

Thank you for this post. I found it very useful for my assignments.

Christopher Strelioff Mod → Brad Arvin • 4 months ago

Glad you found it helpful, thanks for leaving the note.

Best,
Chris

Pages

- **Home (../../../index.html)**

Recent Posts

- **Arduino on Ubuntu 14.04 without the Arduino IDE (../../12/08/arduino_on_ubuntu_14_04_without_the_arduino_ide.html)**
- **Installing Fiona on Ubuntu 14.04 (../../08/20/installing_fiona_on_ubuntu_14_04.html)**
- **Decision trees in python again, cross-validation (../25/decision_trees_in_python_again_cross_validation.html)**
- **PySoundFile and Python 3.4 on Ubuntu 14.04 (../16/pysoundfile_and_python_3_4_on_ubuntu_14_04.html)**
- **Decision trees in python with scikit-learn and pandas**
- **Revisiting the medical tests example with Python and Lea (../../05/27/revisiting_the_medical_tests_example_with_python_and_lea.html)**
- **Probabilistic programming with Python and Lea (../../05/04/probabilistic_programming_with_python_and_lea.html)**
- **JOINs, and some VIEWs, in MySQL (../../04/30/joins_and_some_views_in_mysql.html)**
- **Employees database for MySQL, setup and simple queries (../../04/22/employees_database_for_mysql_setup_and_simple_queries.html)**
- **Installing MySQL on Ubuntu 14.04 (../../04/21/installing_mysql_on_ubuntu_14_04.html)**

Tags

api (../../../tags/api.html) (1), **arduino (../../../tags/arduino.html)** (1), **audio (../../../tags/audio.html)** (2), **audio features (../../../tags/audio_features.html)** (1), **Bayesian (../../../tags/bayesian.html)** (7), **Beta (../../../tags/beta.html)** (1), **blog setup (../../../tags/blog_setup.html)** (1), **bootstrap (../../../tags/bootstrap.html)** (1), **bottleneck (../../../tags/bottleneck.html)** (1), **c++ (../../../tags/c.html)** (1), **caret (../../../tags/caret.html)** (1), **cmpy (../../../tags/cmpy.html)** (1), **conditional probability (../../../tags/conditional_probability.html)** (6), **coursera (../../../tags/coursera.html)** (1), **coursera intro to data science (../../../tags/coursera_intro_to_data_science.html)** (3), **css (../../../tags/css.html)** (1), **cython (../../../tags/cython.html)** (1), **d3 (../../../tags/d3.html)** (1), **decision trees (../../../tags/decision_trees.html)** (2), **diy (../../../tags/diy.html)** (1), **dsp (../../../tags/dsp.html)** (1), **e1071 (../../../tags/e1071.html)** (1), **essentia (../../../tags/essentia.html)** (1), **garmin (../../../tags/garmin.html)** (1), **geojson (../../../tags/geojson.html)** (1), **ggplot2 (../../../tags/ggplot2.html)** (1), **gis (../../../tags/gis.html)** (2), **git (../../../tags/git.html)** (1), **gnuplot (../../../tags/gnuplot.html)** (1), **graphs (../../../tags/graphs.html)** (1), **html5 (../../../tags/html5.html)** (1), **igraph (../../../tags/igraph.html)** (1), **ipython (../../../tags/ipython.html)** (1), **javascript (../../../tags/javascript.html)** (2), **joint probability (../../../tags/joint_probability.html)** (6), **json (../../../tags/json.html)** (1), **LaTeX (../../../tags/latex.html)** (1), **LDA (../../../tags/lda.html)** (1), **Lea (../../../tags/lea.html)** (2), **machine learning (../../../tags/machine_learning.html)** (3), **marginal probability (../../../tags/marginal_probability.html)** (6), **matplotlib (../../../tags/matplotlib.html)** (1), **mir (../../../tags/mir.html)** (1), **music (../../../tags/music.html)** (2), **my python setup (../../../tags/my_python_setup.html)** (5), **my ubuntu setup (../../../tags/my_ubuntu_setup.html)** (10), **mysql (../../../tags/mysql.html)** (3), **networks (../../../tags/networks.html)** (1), **networkx (../../../tags/networkx.html)** (1), **nodejs (../../../tags/nodejs.html)** (1), **npm (../../../tags/npm.html)** (1), **numexpr (../../../tags/numexpr.html)** (1), **numpy (../../../tags/numpy.html)** (1), **octave (../../../tags/octave.html)** (1), **Open Oakland (../../../tags/open_oakland.html)** (2), **openpyxl (../../../tags/openpyxl.html)** (1), **pandas (../../../tags/pandas.html)** (3), **patsy (../../../tags/patsy.html)** (1), **pip (../../../tags/pip.html)** (1), **pweave (../../../tags/pweave.html)** (1), **pygraphviz (../../../tags/pygraphviz.html)** (1), **pymc (../../../tags/pymc.html)** (1), **PySoundFile (../../../tags/pysoundfile.html)** (2), **python (../../../tags/python.html)** (14), **Python (../../../tags/python.html)** (1), **python 2.7 (../../../tags/python_2_7.html)** (5), **python 3.4 (../../../tags/python_3_4.html)** (2), **pyyaml (../../../tags/pyyaml.html)** (1), **qgis (../../../tags/qgis.html)** (1), **R (../../../tags/r.html)** (1), **randomForest (../../../tags/randomforest.html)** (1), **restview (../../../tags/restview.html)** (1), **resume (../../../tags/resume.html)** (1), **rpart (../../../tags/rpart.html)** (1), **running (../../../tags/running.html)** (1), **scikit-learn (../../../tags/scikit_learn.html)** (3), **scipy (../../../tags/scipy.html)** (1), **screen (../../../tags/screen.html)** (1), **server setup (../../../tags/server_setup.html)** (1), **shapefile (../../../tags/shapefile.html)** (1), **social networks (../../../tags/social_networks.html)** (1), **Socrata (../../../tags/socrata.html)** (1), **sound (../../../tags/sound.html)** (2),

sphinx (../../../tags/sphinx.html) (1), **sql (../../../tags/sql.html)** (4), **sqlite3 (../../../tags/sqlite3.html)** (1), **ssh (../../../tags/ssh.html)** (1), **ssh keys (../../../tags/ssh_keys.html)** (1), **statsmodels (../../../tags/statsmodels.html)** (1), **supervised learning (../../../tags/supervised_learning.html)** (2), **sympy (../../../tags/sympy.html)** (1), **tableau (../../../tags/tableau.html)** (1), **tinkerer (../../../tags/tinkerer.html)** (1), **topic models (../../../tags/topic_models.html)** (1), **tree (../../../tags/tree.html)** (1), **ubuntu 14.04 (../../../tags/ubuntu_14_04.html)** (13), **vim (../../../tags/vim.html)** (1), **virtualbox (../../../tags/virtualbox.html)** (1), **virtualenv (../../../tags/virtualenv.html)** (3), **virtualenvwrapper (../../../tags/virtualenvwrapper.html)** (2), **VPS (../../../tags/vps.html)** (1), **yaml (../../../tags/yaml.html)** (1)