Machine Learning with Python scikit-learn review

Jorge F. Monardes

Pratt School of Engineering | Duke University
jorge.monardes@duke.edu
Git Repository: https://github.com/juxtux/cs571D

Abstract

This project is a simple review of scikit-learn, which is an open source machine-learning library for Python. In this project I review and explain the following: Guidance on how to install Python and all the libraries you need for scientific analyses; brief explanation of what are the libraries and tools for scientific Python; give you sources from where you can continue learning using Python as your scientific analysis programming language; and review some of the classification methods seen in class (Ordinary Least Square Regression, Ridge Regression, Logistic Regression, Support Vector Machine, and Clustering).

I. Introduction and Basics

This project is a simple review of scikit-learn, which is an open source machine-learning library for Python. Before starting with scikit-learn, I am going to assume you do not have Python 3.0 installed in your computer and that you use an iOS (i.e., you have a Mac). Nevertheless, if you have Linux or Windows you can use the multiple packaged installer for scientific Python (Anaconda or EPD Free) [1]. From now on, all the basic concepts will be explained from the perspective of a Mac.

In what follows, I will explain step by step how to install scikit-learn and all the libraries you need in your Mac without using a multiple packaged installer (mainly, because I think they don't work). It is important to know that iOS has Python 2.7.X pre-installed, and that this document aims Python 3.5.0 or later versions. Furthermore, if you want to build some experience with Python's syntax, I strongly recommend Code Academy's Python free course [2].

Xcode. It acts as the virtual machine to compile Python and the other installs you will need in your Mac. Go to the App Store and download/install it for free. Also, you can use Xcode editor for future coding, I personally use PyCharm as my editor and IDE (Integrated Development Environment).

Homebrew. It is an excellent package manager for Mac. To install it, you will need to use ruby from the terminal (ruby is pre-installed in your iOS). Type into the terminal:

```
ruby -e "$(curl -fsSkL raw.github.com/mxcl/homebrew/go)"
```

Since Homebrew installs things to usr/local you don't need sudo permissions. You do need to edit your *.profile* file or *.bash_profile* file (probably the latter), in order to specify the path where Python will be installed. In the terminal type:

```
open /Applications/TextEdit.app .bash_profile
```

The file will open into the TextEdit.app, write down the following two lines:

```
PATH="/usr/local/bin:/usr/local/share/python:${PATH}"
export PATH
```

Python. Now that we have Homebrew, the rest is simple [3]. In your terminal, indicate Homebrew to install Python:

```
brew install python
```

Homebrew usually picks the newest python version to install automatically, if not, choose the latest, which today corresponds to Python 3.5.0.
After the installation is complete (2-10 minutes), close the terminal and open a fresh one again. It is important to do so, because the next commands we will call depends on the previous installation, so we need to restart the terminal for these commands to have action in a fresh terminal window.
In this new terminal window check the PATH of the installed Python typing:

```
which python
```

You should see "/usr/local/bin/python" or "/usr/bin/python". You can open and edit the *.profile* file if its necessary.

Pip. Is highly probably you will use more of Homebrew for installing system packages and programming tools in the future. However, what you need now to install Python packages is pip. For doing this, you need to use the Python package manager easy_install. In terminal type:

```
easy_install pip
```

gfortran. Before installing the scientific libraries, we need to tell Homebrew to install gfortran, which is not included in the Xcode tools. In terminal type:

```
brew install gfortran
```

Numpy. Fundamental package for scientific computing with Python. Technically, contains the tools for integrating C/C++ and Fortran code [4]. **SciPy.** Is the core library that makes up the SciPy stack. It provides many user-friendly and efficient numerical routines such as numerical integration and optimization [5]. **Scikit-learn.** The open source machine learning library for Python and main reason of this document [6], [7]. In terminal type:

```
pip install -U numpy scipy scikit-learn
```

matplotlib. Finally, I recommend installing matplotlib, which is a Python 2D plotting library that produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms [8]. To install matplotlib we need to revisit Homebrew in the terminal:

```
brew install pkg-config
```

Then with pip:

```
pip install matplotlib
```

If the last command does not work, try to install it from the matplotlib development repository:

```
pip install git+git://github.com/matplotlib/matplotlib.
    git
```

Now you should have everything you need for start working in a Scientific Python Environment.

PyCharm. So far, you can access from the terminal to Python 3.5 typing (in the terminal):

```
python3
```

However, I personally have a high preference for using an IDE, particularly PyCharm. You can download it directly from its website jetbrains.com or download any other IDE of your preference from python.org.

II. Problem 1: Ordinary Least Square regression, Ridge regression, and Logistic regression.

This problem uses a dataset given in homework 1, corresponding to Leukemia gene expression, and it is divided in 4 parts [9].

1. Part 1: Create a script with functions that read the Leukemia .txt files [10].

I recommend saving the script below in a document termed fns.py or downloading the files from the Git repository. If you copy/paste the code snippets of this paper, **be careful with the indentation**, Python is indent- and whitespace-sensitive.

```
import numpy as np

def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

def file_to_matrix(file_x):
    r_list = list()

    while 1:
        rows = file_x.readline()
        if not rows or rows == '\n':
            break
        rows = rows.strip().split('\t')
        row = [float(num) for num in rows if
    is_number(num)]
        r_list.append(row)

    r_array = np.array(r_list)
    r_array = np.transpose(r_array)
    return np.delete(r_array,0,1)

def file_to_y(file_y):
    r_list = []
    while 1:
        row = file_y.readline()
        if not row:
            break
        if is_number(row[0:2]):
            row_s = row.split('\t\t')
            result = row_s[1].strip()
            r_list.append([int(row[0:2]), result])
    return r_list

def y_array(y, zeros, ones):
    r_array = [-1 if tuple[1] == zeros else 1 if
    tuple[1] == ones else 'error_value' for tuple
     in y]
    return np.array(r_array)
```

2. Part 2: Script with Ordinary Least Square (OLS) regression.

The code below shows OLS regression over the Leukemia dataset and some steps to analyze this linear regression [9]. These steps are:

- How to read the dataset from the Leukemia gene expression files.
- How to process this dataset and develop an OLS linear regression model with scikit-learn.
- How to design a 2D chart of the data and the OLS prediction using matplotlib.
- Additionally, this script prints in the console the residual sum of squares and the variance score of the OLS-prediction.

```
from fns import *
from sklearn import linear_model
import matplotlib.pyplot as plt

# Step1: Read dataset from Leukemia files.
train_f = open("train.txt", "r")
train_x = file_to_matrix(train_f)
train_f.close()

test_f = open("test.txt", "r")
test_x = file_to_matrix(test_f)
test_f.close()

samples_f = open("samples.txt", "r")
y = file_to_y(samples_f)
samples_f.close()

y = y_array(y, 'ALL', 'AML')
train_y = y[0:38]
test_y = y[38:]
# Step2: Linear model on training data and
    predictions with test data.
ols = linear_model.LinearRegression(fit_intercept=
    False)
ols.fit(train_x, train_y)
print("Residual sum of squares: %.2f" % np.sum((
    ols.predict(test_x) - test_y) ** 2))
""" Variance score: 1 is perfect prediction, 0 is
    that the model is not better than random
    choice, and negative value is that the model
    is WORST than random prediction. """
print('Variance score: %.2f' % ols.score(test_x,
    test_y))

# Predictions visualization in 2D chart:
axis_x = [i for i in range(0,len(test_y))]
sep_y = [0 for i in range(0,len(test_y))]
plt.plot(axis_x, sep_y, color='black', linewidth
    =2)
plt.scatter(axis_x, test_y, color='r', alpha=.5, s
    =100, label='y-test data')
plt.scatter(axis_x, ols.predict(test_x), color='b
    ', marker='+', s=60, label='y-test prediction
    ')
plt.xlabel('samples (patients)')
plt.ylabel('Leukemia Classification -- ALL (down)
    | AML (up)')
plt.title('Ordinary Least Square Linear Model',
    fontsize=18, fontweight='bold')
plt.axis('tight')
plt.grid(True)
plt.ylim(-2, 2)
plt.savefig("ols.png")
plt.legend()
plt.show()
```

3. Part 3: Script with Ridge regression.

Similarly, using the libraries loaded in part 2, a Ridge regression model for the Leukemia dataset is developed. Additionally, four different lambdas were chosen to illustrate the shrinkage in the model [9].

```
ridge = linear_model.Ridge(fit_intercept=False)
ridge.set_params(alpha=(10 ** 10))
ridge.fit(train_x, train_y)

print("Residual sum of squares: %.2f" % np.sum((
    ridge.predict(test_x) - test_y) ** 2))
print('Variance score: %.2f' % ridge.score(test_x,
     test_y))
# Predictions visualization:
axis_x = [i for i in range(0,len(test_y))]
sep_y = [0 for i in range(0,len(test_y))]
plt.plot(axis_x, sep_y, color='black', linewidth
    =2)
plt.scatter(axis_x, test_y, color='r', alpha=.5, s
    =100, label='y-test data')
plt.scatter(axis_x, ridge.predict(test_x), color='
    b', marker='+', s=60, label='y-test
    prediction')
plt.xlabel('samples (patients)')
plt.ylabel('Leukemia Classification -- ALL (down)
    | AML (up)')
plt.title('Ridge Regression | Lambda = 10^10',
    fontsize=18, fontweight='bold')
plt.axis('tight')
plt.grid(True)
plt.ylim(-2, 2)
plt.legend()
plt.show()

# Using various Lambdas:
ridge1 = linear_model.Ridge(fit_intercept=False)
ridge1.set_params(alpha=(.5))
ridge1.fit(train_x, train_y)
ridge2 = linear_model.Ridge(fit_intercept=False)
ridge2.set_params(alpha=(10 ** 3))
ridge2.fit(train_x, train_y)
ridge3 = linear_model.Ridge(fit_intercept=False)
ridge3.set_params(alpha=(10 ** 9))
ridge3.fit(train_x, train_y)
ridge4 = linear_model.Ridge(fit_intercept=False)
```

```
ridge4.set_params(alpha=(10 ** 10))
ridge4.fit(train_x, train_y)
axis_x = [i for i in range(0,len(test_y))]
sep_y = [0 for i in range(0,len(test_y))]

# 2D chart with four subplots:
fig, axarr = plt.subplots(2, 2)
fig.suptitle('Ridge Regression', fontsize=18, fontweight='bold')

axarr[0, 0].plot(axis_x, sep_y, color='black', linewidth=1)
axarr[0, 0].scatter(axis_x, test_y, color='r', alpha=.5, s=100, label='y-test data')
axarr[0, 0].scatter(axis_x, ridge1.predict(test_x), color='b', marker='+', s=60, label='y-test prediction')
axarr[0, 0].set_title('Lambda = 0.5', fontweight='bold')
axarr[0, 1].plot(axis_x, sep_y, color='black', linewidth=1)
axarr[0, 1].scatter(axis_x, test_y, color='r', alpha=.5, s=100, label='y-test data')
axarr[0, 1].scatter(axis_x, ridge2.predict(test_x), color='b', marker='+', s=60, label='y-test prediction')
axarr[0, 1].set_title('Lambda = 10^3', fontweight='bold')
axarr[1, 0].plot(axis_x, sep_y, color='black', linewidth=1)
axarr[1, 0].scatter(axis_x, test_y, color='r', alpha=.5, s=100, label='y-test data')
axarr[1, 0].scatter(axis_x, ridge3.predict(test_x), color='b', marker='+', s=60, label='y-test prediction')
axarr[1, 0].set_title('Lambda = 10^9', fontweight='bold')
axarr[1, 1].plot(axis_x, sep_y, color='black', linewidth=1)
axarr[1, 1].scatter(axis_x, test_y, color='r', alpha=.5, s=100, label='y-test data')
axarr[1, 1].scatter(axis_x, ridge4.predict(test_x), color='b', marker='+', s=60, label='y-test prediction')
axarr[1, 1].set_title('Lambda = 10^10', fontweight='bold')

for i in range(0, 2):
    for j in range(0, 2):
        axarr[i, j].set_xlabel('samples (patients)')
        axarr[i, j].set_ylabel('Leukemia Classification -- ALL (down) | AML (up)')
        axarr[i, j].grid(True)
        axarr[i, j].set_ylim([-2, 2])
        axarr[i, j].legend(loc='upper left', framealpha=0.6)

fig.subplots_adjust(hspace=.25)
plt.show()
""" Additionally, you can do a Cross-validation to set lambda/alpha. """
ridge_cv = linear_model.RidgeCV(alphas=[0.5, 10**3, 10**9, 10**10], fit_intercept=False)
ridge_cv.fit(train_x, train_y)
print(ridge_cv.alpha_)
```

4. Part 4: Script with Logistic regression.

Finally, we can submit our dataset to logistic regression and likewise the previous models, calculate its residual sum of squares, its variance score, and chart each patient's data vs prediction [9].

```
lreg = linear_model.LogisticRegression(penalty='l2', C=1e-5, fit_intercept=False, max_iter=100)
# C: inverse of regularization strength; must be a positive float. Smaller values specify stronger regularization.
lreg.fit(train_x, train_y)
print("Residual sum of squares: %.2f" % np.sum((lreg.predict(test_x) - test_y) ** 2))
print('Variance score: %.2f' % lreg.score(test_x, test_y))
# Predictions visualization:
axis_x = [i for i in range(0, len(test_y))]
sep_y = [0 for i in range(0, len(test_y))]
plt.plot(axis_x, sep_y, color='black', linewidth=2)
plt.scatter(axis_x, test_y, color='r', alpha=.5, s=100, label='y-test data')
plt.scatter(axis_x, lreg.predict(test_x), color='b', marker='+', s=60, label='y-test prediction')
plt.xlabel('samples (patients)')
plt.ylabel('Leukemia Classification -- ALL (down) | AML (up)')
plt.title('Logistic Regression | Inv Regularization Strength = 1e-5', fontweight='bold')
plt.axis('tight')
plt.grid(True)
plt.ylim(-2, 2)
plt.legend()
plt.show()
```

III. Problem 2: Support Vector Machines (SVM).

An academic example of SVM was developed using a classic dataset of the iris flower (setosa, versicolour, and virginica). This data is part of the standard datasets that scikit-learn posses, you do not require to download any file from some external website.

The idea for the next code snippet is to illustrate the classification with SVM in three-dimensions. Using information of the sepal width, sepal length, and petal length. The last is something that is not easy

to found in literature or in the web, thus the importance to develop a visual understanding of the cloud regions of belonging in higher dimensional spaces. The code below allows you to generate a three-dimensional illustration of SVM classification using a Linear Kernel, a Radial Basis Function (Gaussian) Kernel, or a Polynomial Kernel of third degree [9].

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import
    LinearSegmentedColormap
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn import svm

""" Data: """
iris_plant = datasets.load_iris()
X = iris_plant.data
y = iris_plant.target

X_train = X[:, 0:3]
X1, X2, X3 = X_train[:50], X_train[50:100],
    X_train[100:150]
y1, y2, y3 = y[:50], y[50:100], y[100:150]

""" 3D Meshgrid: """
line_space = np.linspace(0, 9, 20)
line_space = np.around(line_space, decimals=2)
xy, xz = np.meshgrid(line_space, line_space)
for z in line_space:
    xy_z = np.c_[np.ravel(xy), np.ravel(xz)]
    xy_z = np.insert(xy_z, 2, z, axis=1)
    space3D = xy_z if z == 0 else np.append(
    space3D, xy_z, axis=0)


""" SVM: (default) Linear Kernel """
C = 1.0     # SVM regularization parameter
svm = svm.SVC(kernel='linear', C=C).fit(X_train, y
    )
svm2 = svm.SVC(kernel='rbf', gamma=0.2, C=C).fit(
    X_train, y)  # Gaussian Kernel
svm3 = svm.SVC(kernel='poly', degree=3, C=C).fit(
    X_train, y)  # Polynomial Kernel

y_hat = svm.predict(space3D)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X3[:, 0], X3[:, 1], X3[:, 2], c='b',
    label='Virginica')
ax.scatter(X2[:, 0], X2[:, 1], X2[:, 2], c='g',
    label='Versicolour')
ax.scatter(X1[:, 0], X1[:, 1], X1[:, 2], c='r',
    label='Setosa')

ax.set_xlabel('sepal length (cm)')
ax.set_ylabel('sepal width (cm)')
ax.set_zlabel('petal length (cm)')

vmax = 3.0
cmap = LinearSegmentedColormap.from_list('mycmap',
     [(0 / vmax, 'r'),

      (1 / vmax, 'g'),

      (3 / vmax, 'b')]
                                          )
ax.scatter(space3D[:, 0], space3D[:, 1], space3D
    [:, 2], c=y_hat, cmap=cmap, marker='s', s
    =100, alpha=.05)

plt.title('SVM Classification (Linear Kernel) |
    Regularization: C = 1', fontweight='bold')
plt.legend()
plt.show()
```

IV. Problem 3: Clustering Mean-Shift.

This last example is a minor modification of the example given in the scikit-learn website about mean-shift clustering. I chose it to explain with more detail in comments throughout the script [9].

```
import numpy as np
from sklearn.cluster import MeanShift,
    estimate_bandwidth
from sklearn.datasets.samples_generator import
    make_blobs

""" Generate sample data: """
# 4 centroids:
centers = [[1, 1], [-1, -1], [1, -1], [-1.5, 3]]
# Data with 15,000 data-points, 4 potential
    centroids, and specific standard deviation:
X, _ = make_blobs(n_samples=15000, centers=centers
    , cluster_std=0.6)

# The following bandwidth can be automatically
    detected using:
bandwidth = estimate_bandwidth(X, quantile=0.1,
    n_samples=500)
# Clustering with MeanShift and specific bandwidth
     for the RBF Kernel to use.
ms = MeanShift(bandwidth=bandwidth, bin_seeding=
    True)
ms.fit(X)

labels = ms.labels_
cluster_centers = ms.cluster_centers_    # Get
    centroids of the model

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)

print("number of estimated clusters : %d" %
    n_clusters_)

""" Plot result: """
```

```
26 import matplotlib.pyplot as plt
27 from itertools import cycle
28
29 plt.figure(1)
30 plt.clf()       # just clear the current figure
31
32 colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
       # Cycle list. e.g., in this case with 4
     centroids its going to use just 'bgrc'
33 for k, col in zip(range(n_clusters_), colors):
34     my_members = labels == k  # Boolean vector
     identifying the clustering labels
35     cluster_center = cluster_centers[k]  #
     centroid of the corresponding label
36     plt.plot(X[my_members, 0], X[my_members, 1],
     col + '.')
37     plt.plot(cluster_center[0], cluster_center[1],
      'o', markerfacecolor=col, markeredgecolor='k
     ', markersize=14)
38 plt.title('Estimated number of clusters: %d' %
     n_clusters_)
39 plt.show()
```

V. Final words

The experience developing this project showed me the powerful scientific programming language that Python is. Each script can be run in a regular Macbook Pro taking less than one second in developing models and charts. Even for the SVM examples shown in 3-dimension take less than a second to run.

Furthermore, I was abled to see part of the complexity that scikit-learn has for each of it methods, unraveling all the constants that can be set for the mathematical models. Additionally, I analyzed the sensitivity of processing efficiency parameters, which can optionally be adjusted for the advantage of the model (cache size, max number of iterations, verbose, etc.)

Likewise with Matplotlib, it was highly satisfying to see my results in elegant and high quality figures, being able to adjust dozens of different parameters in order to acquire the level of illustration of the information that I needed.

The project in general was intellectually rewarding and gave me enough motivation to continue increasing my knowledge & skills in Machine Learning.

References

[1] Install Python All-in-One. (2012, July 25). Retrieved October 28, 2015, from http://penandpants.com/install-python/

[2] Code Academy. (2015). Retrieved October 29, 2015, from https://www.codecademy.com/

[3] Welcome to Python.org. (2001). Retrieved October 28, 2015, from https://www.python.org/

[4] NumPy. (2013). Retrieved October 28, 2015, from http://www.numpy.org/

[5] SciPy. (2015). Retrieved October 28, 2015, from http://www.scipy.org/

[6] Scikit-learn: Machine learning in Python. (2013). Retrieved November 6, 2015, from http://scikit-learn.org/

[7] Install Python, NumPy, SciPy, and Matplotlib on Mac OS X. (2012, February 24). Retrieved October 28, 2015, from http://penandpants.com/2012/02/24/install-python/

[8] Matplotlib. (2012). Retrieved October 28, 2015, from http://matplotlib.org/

[9] Monardes, J. (2015). CS571-Python Files Git Repository. Retrieved December 2, 2015, from https://github.com/juxtux/cs571D.git

[10] Mukherjee, S. (2015). ML Homework 1 Dataset. Retrieved December 2, 2015, from https://stat.duke.edu/ sayan/561/2015/homeworks/hw1/