Case Study
Project: Scikit-Learn
Seung Hee Lee

1)

Scikit-learn is a python module that is built for the machine learning library. In this project various machine learning algorithms could be used such as classification, regression and clustering. Since scikit-learn requires numerical and vector calculations it interoperates with the Numpy and Scipy libraries in Python. Scikit-learn is written mostly in Python but some core algorithms are written in Cython for optimization. Scikit-learn was developed and published in 2010 which is not long ago. Therefore if scikit-learn was written today, I assume that python was still used since Numpy is easily operates and compatible used library with scikit-learn.

Scikit-Learn is one of the python library that operates with numpy and pandas. When installing, you need "pip" and "distutils" as build system. When using pip, user needs to make sure "binary wheel" is used. For build tools, any text-editor, notebook web application (Jupyter Notebook), or Anaconda that runs python and its libraries will be compatible and buildable for scikit-learn.

Scikit-learn requires:

- Python (>= 2.7 or >= 3.4),
- NumPy (>= 1.8.2),
- SciPy (>= 0.13.3).

There are some tools that enhance scikit-learn's interoperability and framework. One of the experimentation frameworks is called Xcessiv. It is a notebook-like application hyperparameter stacked ensembles. Stacked ensembles are the combination of predictions of smaller models and feed those models to another model. In Machine Learning, feeding to another model can be much more complicated and cause a lot of time consuming. Xcessive is a tool that can assist the implementation optimization of stacked ensembles. Xcessive is a framework to keep track of model hyperparameter combination.

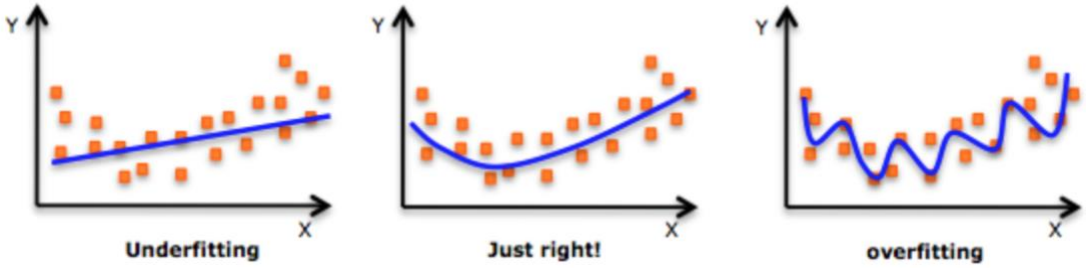
2)

As mentioned above, scikit-learn is the python library for data science or data analysis. One of the important concept that needs to be considered in data science is overfitting. In Machine Learning, preventing or minimizing **overfitting and underfitting** would result more accurate predictions. In order to avoid both of these issues, and find the middle ground between overfitting and underfitting, train/test split and cross validation could be used. To enhance the result of prediction rate or score we can use **train/test split and cross validation.**

a) **Underfitting**: The model does not fit the training data and it misses or mis interprets the trends of the dataset. Since it fails to find the trend of training data, it cannot generalize the result of the new data.

b) **Overfitting**: That model trained has trained too perfectly for only certain case. Eventually, it will not very accurate for new or trained data. Because the model is not generalizing the results and cannot interface with the other data set.

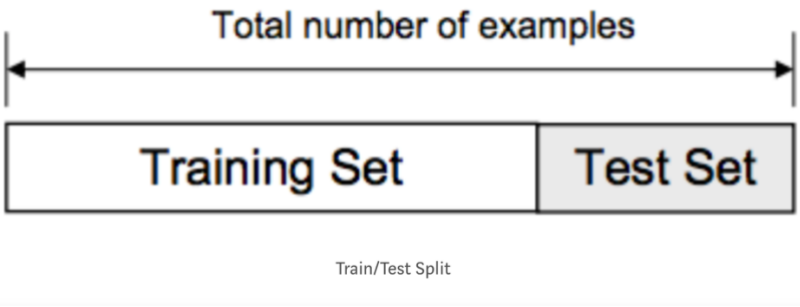
Fig.1



Method 1: Train/Test Split

The data used for scikit-learn usually split into training and test data. In train data the model contains the output also and use it as generalization for the later data that will be fit on to the model.

Fig.2



Train/Test Split

Depending on the data set, the ratio of training set to the test set would be different. By using train_test_split function in scikit-learn, the data can be split into test and train. For instance we could predetermine *test_size = 0.2* (Test size is 20 percent of overall dataset)**.**

```
# create training and testing vars
X_train, X_test, y_train, y_test = train_test_split(df, y,
test_size=0.2)
print X_train.shape, y_train.shape
print X_test.shape, y_test.shape

(353, 10) (353,)
(89, 10) (89,)
```

Then, we can fit the model on the training data and predict the output of test data. By using score function, we can see the how accurate the result of splitting data.

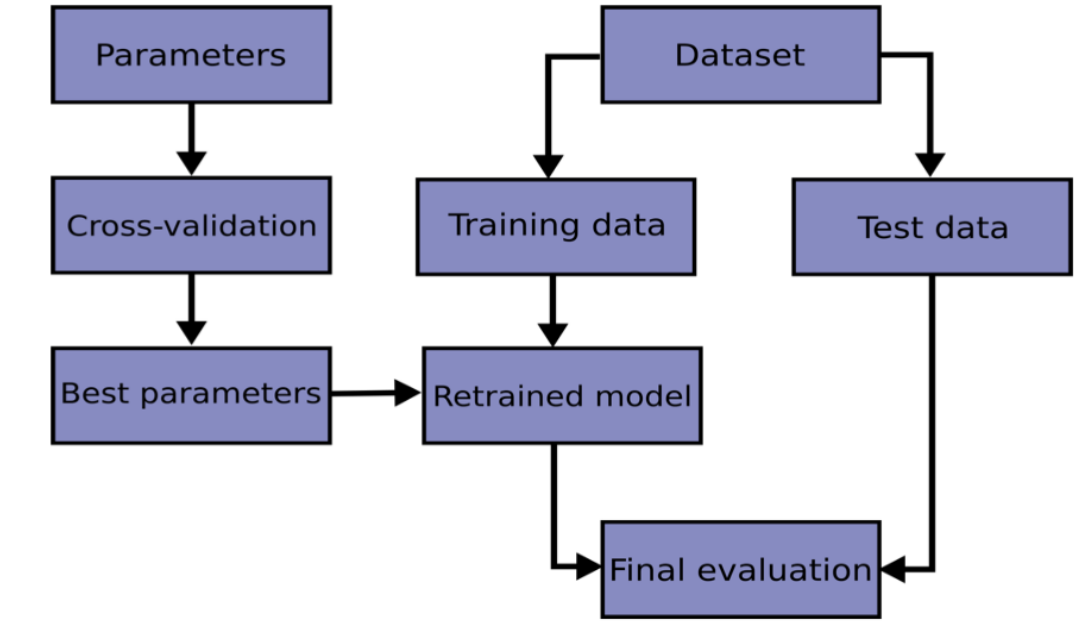
```
print "Score:", model.score(X_test, y_test)

Score: 0.485829586737
```

Train/test data is only safe when we are assure that it is random. For example if the subset of our data only certain part: people from a certain state, employees with a certain income, children from certain age. These are not the complete randomized situation. This could result the overfitting that we want to avoid. Here, cross validation can be used.

Method 2: Cross Validation

For the testing subsets of data, we can use cross validation. The bottom figure shows the general workflow of the cross validation. Cross validation is a model evaluation method that does not use entire data set when training a learner. Some of data is removed before training starts. When training model is done, the data that was not used can be tested as "new" data and give the result of performance of the model. The "train_test_split" method can be used when separating two sets of data.

Fig. 3



There is computing cross-validated metrics which is called “cross_val_score”. It is a helper function on the estimator and a data set. The score tells you the accuracy of the testing data that fits to the model.

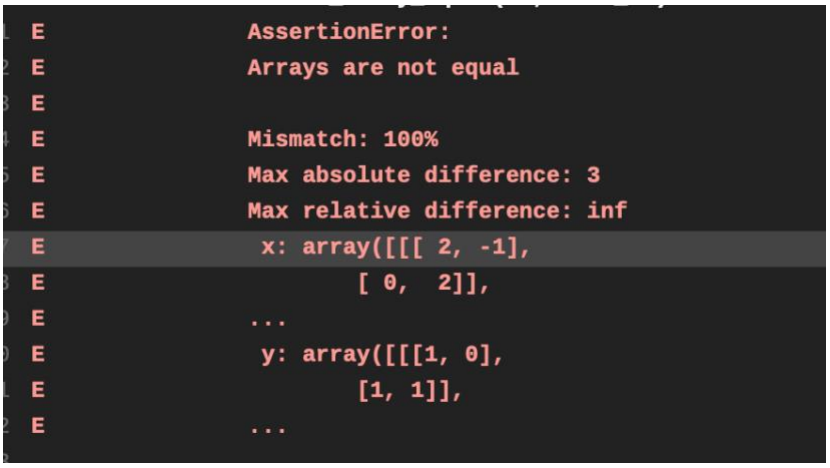
Overall, scikit-learn has estimator, helper function, score function. This helps users how the model is well-trained/generalized when new data set is fitted. However we can also enhance the estimator by using train-test-split method and cross validation.

Method 3: Testing via Assert

```
def test_y_mean_attribute_regressor():
    X = [[0]] * 5
    y = [1, 2, 4, 6, 8]
    # when strategy = 'mean'
    est = DummyRegressor(strategy='mean')
    est.fit(X, y)

    assert_equal(est.constant_, np.mean(y))
```

Due to the fact that Scikit-Learn is a opensource project with many functionalities and over 1000 contributors, it is hard to manually check all the functions by hand. So the Github repo of the project contains testing functions (Like the one shown above) to automatically check for invalid answers by using the asserting function in Python. For the example above, the code checks to see if the constant(mean) obtained from the Scikit-Learn function matches the mean from the Numpy

mean function for validation. Once the test goes through the new commit can proceed being added to the project. However if the assertion function deems to be wrong, the build (Test) fails.



Scikit-Learn uses Continuous Integration depends on the OS. According to the scikit-learn.org:

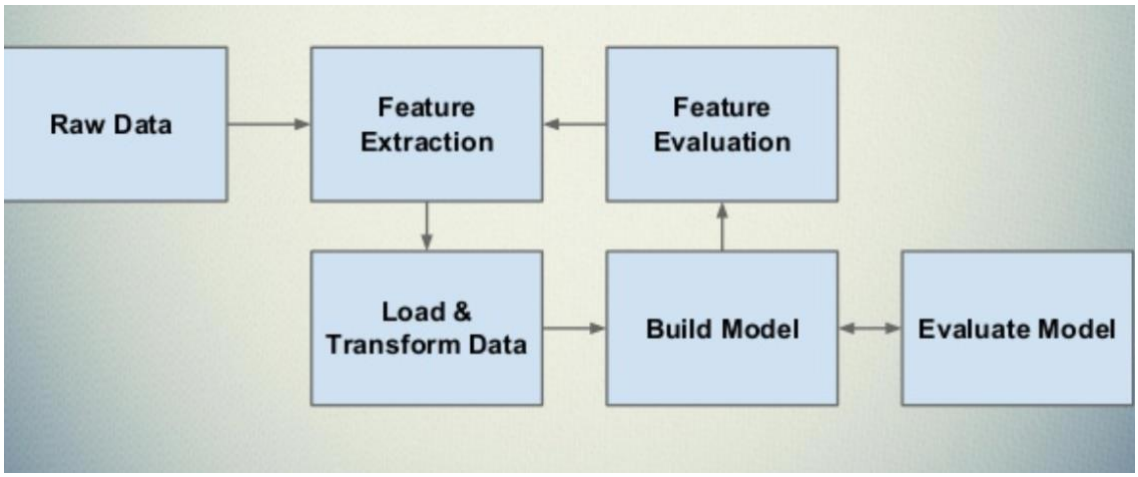
- Travis is used for testing on Linux platforms
- Appveyor is used for testing on Windows platforms
- CircleCI is used to build the docs for viewing, for linting with flake8, and for testing with PyPy on Linux

3)

Scikit-Learn Architecture Diagram Description

The bottom diagram reveals the general visualization how scikit-learn runs. First, we need to define which model we want to use: linear regression, clustering, KNN and so on (here, you also need to define if the data modeling will be supervised or unsupervised). Then load the dataset. Usually it is csv file that has headers. These dataset could be directly used as train dataset, or split into train and test dataset. By applying platform that scikit-learn provides, such as pandas, and numpy, dataset can be loaded and reshaped to be implemented in certain model. When the dataset loaded and reshaped, the features of scikit-learn has can be used such as, SVM, logistic regression and so on. Depends on the algorithm the user chooses, the form of output would be different. For instance, linear regression gives the numeric value for the prediction, whereas other classification methods gives class values such as “class 1” or “class 0”. These trained model can generalize the output of the dataset. Therefore, when the new dataset or test dataset is fit to the model, the output could be determined with certain accuracy. The accuracy could be determined by the testing, validation, score function, and CCR.

Diagram 1



Currently, Scikit-learn does not have visualization functionality. In order to plot something or graph, we need to use another Python library, called Matplotlib. Plotting can be tedious and requires many lines of coding when we have to use Matplotlib. If scikit-learn could include visualization functionality that automatically calls data that used for modeling, this could reduce time to write codes and efforts.

For instance, the bottom code is when plotting the linear regression model of dataset.

```
ols15 = LinearRegression().fit(X,rolling_fif);
ols15.score(X, rolling_fif)

roll15_vector = ols15.coef_*X + ols15.intercept_

plt.plot(X,rolling_fif)
plt.plot(X,roll15_vector)
plt.title("The Rolling Fifteen Train")
plt.show()
```

There are 4 lines of code were used to show a simple graph. If scikit learn can provide the function with simpler coding method such as “scikit_learn_graph(xdata, linearregression, fit=true)”, this could make visualization much easier for users.

External Program

Scikit-learn is not a standalone program. Apparently many external projects and businesses use Scikit-learn as their built-in function for operations. Scikit-learn uses other python libraries and frameworks to be fully operate. For instance, Pandas, NumPy, and Matplotlibs are mainly used libraries that interoperates with Scikit-learn. Pandas used for data structure, data format that loads for scikit-learn library(Same as the numpy). Without other frameworks, scikit-learn is almost not functioning or unable to be used. For the external projects, JP Morgan, Spotify, Booking.com are all big customers of scikit-learn. Scikit-learn in spotify is the main tool kit for recommending music using ML. Likewise, in Booking.com the scikit-learn is used for recommending hotels, flights, detecting fraudulent reservations, or scheduling our customer service agents. Scikit-learn is widely used tools for prediction tasks.

Parallelism

Scikit- learn manages its parallelism with Joblib. Joblib is a set of tools to provide lightweight pipelining in python. Joblib is used for simple parallel computing. Joblib has a multiprocessing backend for distributed systems.

For the model optimization, we need continuous repetition of computing in Machine Learning. We can reduce the processing time by using multiprocessing function in scikit-learn. One of the scikit-learn functions, there is function called GridSearchCV, and it has "n_jobs" parameter. The default value of n_jobs is 1. But if we increase the value of n_jobs, the internal multiprocessing is used and operate GridSearchCV. If we have enough number of CPU Core, when number of n_jobs increase, the speed of computing also increases.

We can easily compare the time difference between n_jobs = 1 and n_jobs =8 by using GridSearch.

```
from sklearn.model_selection import GridSearchCV

param_grid = {"gamma": np.logspace(-6, -1, 10)}
gs1 = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=1)
gs2 = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=8)
```

n_jobs = 1

```
CPU times: user 35.2 s, sys: 4 ms, total: 35.2 s
Wall time: 35.2 s
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'gamma': array([  1.00000e-06,   3.59381e-06,   1.29155e-05,   4.64159e-05,
         1.66810e-04,   5.99484e-04,   2.15443e-03,   7.74264e-03,
         2.78256e-02,   1.00000e-01])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='accuracy', verbose=0)
```

n_jobs = 8

```
CPU times: user 684 ms, sys: 104 ms, total: 788 ms
Wall time: 5.76 s
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=8,
       param_grid={'gamma': array([  1.00000e-06,   3.59381e-06,   1.29155e-05,   4.64159e-05,
         1.66810e-04,   5.99484e-04,   2.15443e-03,   7.74264e-03,
         2.78256e-02,   1.00000e-01])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='accuracy', verbose=0)
```

As you see, when n_jobs = 8 , the total time and CPU time decreases much more than when n_jobs =1. Because machine learning requires a lot of computation of same dataset for optimization of prediction, sklearn's parallel method helps the reducing processing time.

Functional Components

Overall, scikit-learn lean on functional components than object-oriented. Scikit-learn has many built in function for classification, prediction for Machine Learning. All users have to do is call the project and call the related functions that needs to be used. For instance, if mean squared error is needed for linear regression, sklearn already has the built-in function to compute it.

```
from sklearn.metrics import mean_squared_error

y_predict = regression_model.predict(X_test)

regression_model_mse = mean_squared_error(y_predict, y_test)

regression_model_mse
```

```
12.230963834602681
```

Just like the example above shows, scikit-learn library has necessary functions that users can use for machine learning rather than objects handling most of the tasks. Therefore, scikit-learn is functional components project.

Issues

For the issues related to this project, most of the defects are considered bugs rather a serious functional flaw. Most of the issues posted are bugs relating to a certain classifier and their behaviors to a certain input. One way to possibly to resolve this is to make each functions of the sklearn library show steps of process visually. For example, when we observe a bug in a logistic regression classifier, by showing how each datapoint reached their classes it would make easier for the users to debug and alter their code. This modification would require significant addition to the project since every function is subject to modification. It wouldn't require an overhaul of the architecture, but a steps() function in every classifier/regression models.

Since there isn't a significant "Flaw" to the project, one more addition regarding the bug issue would be a randomized testing method. If we look at the testing code in Github, all the tests of the machine learning models are conducted on a rather simple example. This could be problematic due to the fact that most of the actual implementation would be on a input of large dataset. For a better testing in various input sizes, we could implement a testing scheme that is randomized and also large in scale. This would help the testing module to figure out any bug when the input scales to bigger size. Although this would impact the runtime of CI platform testing, it would be better in terms of resolving a major bug issue in Scikit learn library. It

wouldn't require an overhaul of the architecture either, it would require additional testing code to the existing testing code.

5) Application

Testing Linear Regression:
Linear Regression is used to predict the value of the data. Based on previous plot, linear Dataset is based on sea_level_df

The X is the time line and Y is the level_variation.
Here, we want to find out the 2013 sea_level variation by using the linear regression.

First, I loaded the data to the file,

```
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt

from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LinearRegression
```

```
# Read training set
sea_level_df = pd.read_csv("sealevel_train.csv")
sea_level_df.head()
```

	time	level_variation
0	1993.011526	-37.52
1	1993.038692	-38.05
2	1993.065858	-37.61
3	1993.093025	-37.49
4	1993.120191	-36.48

```
# Read testing set
sea_level_df_test = pd.read_csv("sealevel_test.csv")
sea_level_df_test.head()
```

	time	level_variation
0	2013.453940	26.31
1	2013.481106	26.32
2	2013.508272	26.44
3	2013.535439	26.54
4	2013.562605	26.49

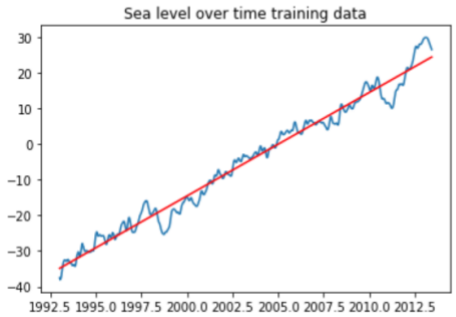
Then, I need to fit the LinearRegression(X,Y) which is the implemented function in scikit learn.

Since we know which will be the time value to predict level variation, we can just input the fitted data and compute the value. Here, I used Y = mX + b, where m is Linear Regression coefficient and b is the intercept.

```
# Part a - Fit linear regression model to training data (find OLS coefficients) #

# Read training set

X = sea_level_df[['time']]
Y = sea_level_df[['level_variation']]
print(type(X))



LR = LinearRegression().fit(X,Y)
LR.score(X, Y)

print("Coeffs:", LR.coef_)
print("Intercept:", LR.intercept_)


# Predict using OLS model

year_predict = 2013.453940

predict_val = year_predict*LR.coef_ + LR.intercept_
predict_vector = LR.coef_*X + LR.intercept_

print("The predicted value for 2013", predict_val)



# Plot training data along with the regression curve
plt.title ("Sea level over time training data")
plt.plot(X,Y)
plt.plot(X,predict_vector,'r')
```

Here is the result graph from original graph and the linear regression graph.

```
<class 'pandas.core.frame.DataFrame'>
Coeffs: [[2.90021144]]
Intercept: [-5815.0064476]
The predicted value for 2013 [[24.43570164]]

[<matplotlib.lines.Line2D at 0x124030f60>]
```



References:

https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
https://scikit-learn.org/stable/modules/cross_validation.html
https://scikit-learn.org/stable/developers/contributing.html#continuous-integration-ci
https://stackoverflow.com/questions/38601026/easy-way-to-use-parallel-options-of-scikit-learn-functions-on-hpc
https://www.slideshare.net/BenjaminBengfort/introduction-to-machine-learning-with-scikitlearn
https://datascienceschool.net/view-notebook/0e9636f97c3d4b47b87a08b2ef0349b7/
http://benalexkeen.com/linear-regression-in-python-using-scikit-learn/