Landon Buell

18 Feb 2020

Breaking Down Sci-kit Learn’s SGDClassifier Object

**Introduction**

Sci-kit learn (sklearn) is a standard, open source tool set module for python that primarily deals in functions, objects and algorithms pertaining to machine learning. There are various submodules within this tool set that give rise to methods for different uses. For this paper, I will break down the SGD Classifier object from the linear\_model submodule.  SGDClassifier is short for *Stochastic Gradient Descent Classifier*, which is multi-purpose optimization algorithm commonly used in machine learning practices. By breaking down this module, we will explore all what computations and operations are being performed ‘under the hood’ so that we may possibly manipulate the object to better suite a specific need. For this paper, *I will be using sklearn version 0.22.1.*

Documentation page for sklearn.linear\_model.SGDClassifier:

<https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html>

**The SGDClassifier Class Object**

The SGDClassifier is a pre-created python class object. It operates just as any object, it has a series of methods, attributes and initialization parameters, and its properties can be changed dynamically as well. To understand how the object works, we’ll examine the class a few of the class parameters, and methods. See the documentation page linked above for a more detailed explanations as needed.

We’ll start by loading in a toy dataset, the IRIS set, and the SGDClassifier object from the linear\_model submodule. We can extract the data matrix (called *X* by convention) and the target vector (called *y* by convention). We will also import a few other modules that will be needed later on. Create and instance of the SGD object, and we set a few parameters. See documentation for details.

Documentation page of IRIS dataset:

<https://scikit-learn.org/stable/datasets/index.html#iris-dataset>



Now, we can right click on the *SGDClassifier* text , to open a pop-up window, and then selected *‘Go to definition’*. In most IDE’s (like Visual Studio) , this opens up a new tab, with the source code for that particular class or module – called \_stochastic\_gradient\_.py. Starting at line 751, we can see the SGD classifier definitions. It is also a child class to the parent class *BaseSGDClassifier*.

**SGDClassifier.\_\_init\_\_()**

By convention, when a class instance is created, the method \_\_init\_\_(), short for initialization is called. This method begins at line 979. It assigns to this particular instance all of the necessary values, most are given by default. If a user sets a parameter manually, then that overrides the default setting. After this method is finished, the instance has been created and the necessary attributes and parameters have been attached to the object.

**BaseSGDClassifier.fit()**

The fit() method is the primary method used to train the classifier object. To use this, we need to create a set of training data and the corresponding training labels. We implement a standard splitting algorithm, also from sklearn, and then use the training set and targets to fit the class instance. This is “training the model”

Documentation for sklearn.model\_selection.train\_test\_split():

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>

****

Now we have functionally trained the model and are ready to make predictions of our testing data set or another validation set. But first, we examine the fit() method closer to see what is really happening. This method actually calls the parent class “BaseSGDClassifier” on like 472. The method itself is defined on 717.

The method takes the arguments X and y which are the training data and labels respectively, coef\_init, intercept\_init, and sample\_rate which are all set to *None* types by default- they can be overwritten by a user if chosen. When the fit() methodis called, it returns the \_fit() method on line 563. This is where the operations of the method begin to take place. We can break this method down section by section. For reference, in the documentation it appears as:



**BaseSGDClassifier.\_fit() Method**

Most of this method is dedicated to validating the input parameters given to the algorithm. It serves to ensure that all the arguments passed into the function are reasonable and comply with expected inputs. (See line 122 for self.\_validate\_params() method details). After this is done, the method sets an iteration counter, and then calls the self.\_partial\_fit() method on line 516. At this point, it is important to know that even after calling the fit() method on the classifier, we have not yet performed any computations, but merely called other functions that organize the arguments and validate them.

**BaseSGDClassifier.\_partial\_fit()**

This method is defined on line 516. It takes the training matrix, *X*, the target vector *y* and other arguments that have been attached as sttribute to the class, or overwritten by the user. Firstly the program establishes the number of training samples, and the number of features by counting the row and columns respectively in the X matrix. After words, it identifies the number of classes that the data is being sorted into. For a *k-fold* classifier, the number of classes is the integer *k*. If k > 2, the method self.\_fit\_multiclass() is called and if k = 2, the method self.\_fit\_binary() is called. For the deconstruction, we will assume k > 2 because we can always generalize it back to k = 2.

**BaseSGDClassifier.\_fit\_multiclass()**

The \_fit\_multiclass() method is defined on line 632. The documentation string indicate that this method creates a multiclass classifier by combining several binary classifiers. It predicts a particular case, *kn* against all other *k-1* classes. This is commonly referred to as a OvA (one vs. all) classifier.



The program splits the data into a training and validation set, and returns a binary array where a 0 indicates the sample for training, and 1 indicates for validation. Next, the algorithm uses a module called *joblib* to apply some parallel computation to the fitting algorithm.

Documentation page of joblib.Parallel object:

<https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html>

The program creates an instance of the Parallel class called *result*. The parameter n\_jobs = self.n\_jobs is telling the instance to prepare to make *n* computational jobs, in this case, one for each of the *k* classes (for the OvA classifier). Once this has been executed, the next line calls the fit\_binary() method , and the subsequent lines pass the necessary arguments to that function.

fit\_binary()

The binary classification fitting function (not class method) is globally defined on line 367. The function takes in the SGD object being fit, the class of interest, the data matrix and target labels as well as some other parameters from the SGD object. This function then calls a preparation function, which returns a vector one floating point 1’s with the dimensions of the target vector, a vector of weights to apply to each feature, and a few other parameters attached to the SGD instance.

Next the program test whether or not the parameter average has been set to *True,* *False* or an integer. This parameter determines the number of training samples to see before averaging the input features. If This parameter is set to *False*, then sklearn runs the plain\_sgd() method, otherwise, it runs the average\_sgd() method. (So far I have not found the source code for the \_sgd\_fast.py code. It is buries within the sklearn. Module folder, and not a traditions .py file.) The program finishes by returning three parameters: standard\_coeff, standard\_intercept and n\_iter\_. As best as I can tell these are produced as linear coefficients to some best fit model. N\_iter\_ gives the number of iterations passed over the current model, as it is used elsewhere in the program and is added to a global counter.