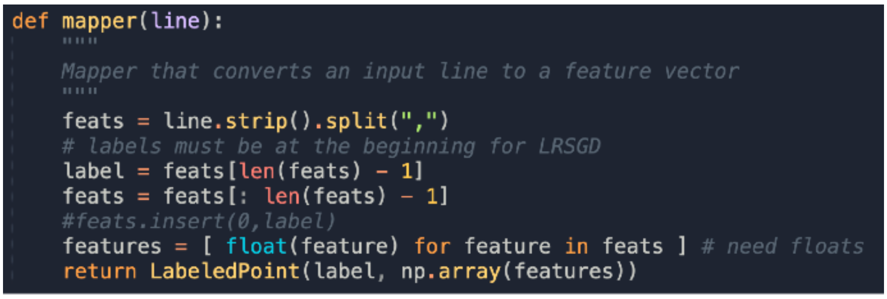
HW4 Spark - Logistic Regression

ID: R08921A07
NAME: 曾梓豪

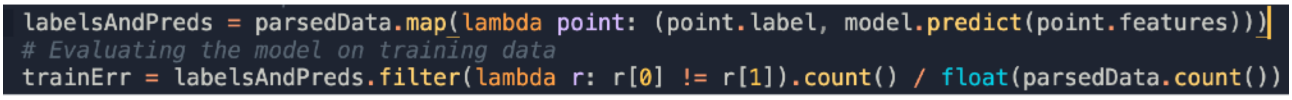
1. Reproduce the results using your own spark cluster (can be in standalone, or in Google/AWS/Azure Cloud).
Following is my reference site:
https://www.hackerearth.com/practice/notes/samarthbhargav/logistic-regression-in-apache-spark/

When parsing the data, I use LabeledPoint to wrap the label and features.
Because, the class “LabeledPoint” can automated wrap the label with features, I don’t need to concatenate label with features manually.



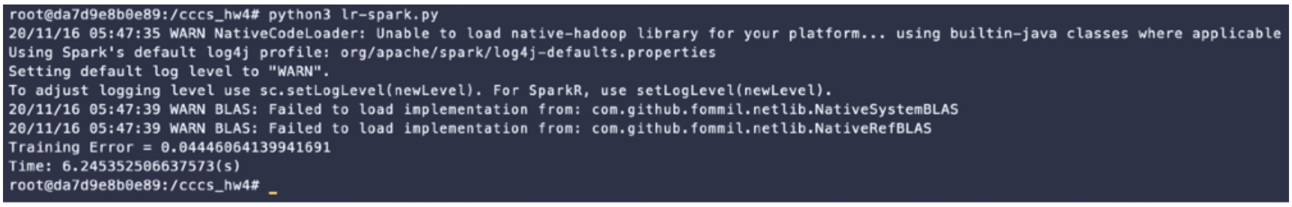
```python
def mapper(line):
    """
    Mapper that converts an input line to a feature vector
    """
    feats = line.strip().split(",")
    # labels must be at the beginning for LRSGD
    label = feats[len(feats) - 1]
    feats = feats[: len(feats) - 1]
    #feats.insert(0,label)
    features = [ float(feature) for feature in feats ] # need floats
    return LabeledPoint(label, np.array(features))
```

We also need to change the lambda function, because “LabeledPoint” can use predefined attribute to load the label and features.



```python
labelsAndPreds = parsedData.map(lambda point: (point.label, model.predict(point.features)))
# Evaluating the model on training data
trainErr = labelsAndPreds.filter(lambda r: r[0] != r[1]).count() / float(parsedData.count())
```

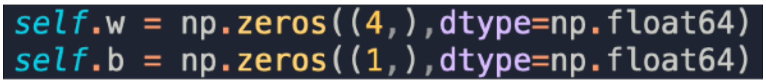
The following is the result of spark cluster example:
The error rate is 0.04446064139941691, and it cost 6.25 second to process.



```
root@da7d9e8b0e89:/cccs_hw4# python3 lr-spark.py
20/11/16 05:47:35 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/11/16 05:47:39 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
20/11/16 05:47:39 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
Training Error = 0.04446064139941691
Time: 6.245352506637573(s)
root@da7d9e8b0e89:/cccs_hw4# _
```

2. Write your own SGD (stochastic gradient descent or simple gradient descent) function of logistic regression. And compare the results.

Because of the dimension of features, we can initialize of the weight and bias in the following way:



```
self.w = np.zeros((4,),dtype=np.float64)
self.b = np.zeros((1,),dtype=np.float64)
```

For logistic regression, the gradient of weight is:

$$\frac{err(W)}{\partial w_i} = -\sum_n (\hat{y^n} - y^n)x_i^n$$

and the update function of w_i is :

$$w_i = w_i - \eta \sum_n -(\hat{y^n} - y^n)x_i^n,$$

η is the learning rate, which can control the speed of converge.

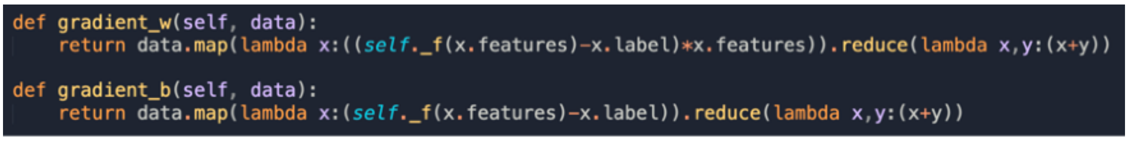
Let b be the bias of hypothesis, and the gradient of b is:

$$\frac{err(W)}{\partial b} = -\sum_n (\hat{y^n} - y^n)$$

and the bias update function is:

$$b = b - \eta \sum_n -(\hat{y^n} - y^n)$$

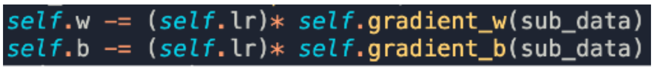
Following, are the implementation of gradient of weight and bias:



```
def gradient_w(self, data):
    return data.map(lambda x:((self._f(x.features)-x.label)*x.features)).reduce(lambda x,y:(x+y))

def gradient_b(self, data):
    return data.map(lambda x:(self._f(x.features)-x.label)).reduce(lambda x,y:(x+y))
```

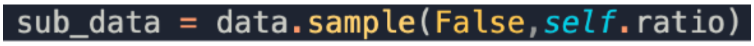
and the following is the weight and bias update function, and "self.lr" is the learning rate:



```
self.w -= (self.lr)* self.gradient_w(sub_data)
self.b -= (self.lr)* self.gradient_b(sub_data)
```

Because we need to implement the stochastic gradient descent, I use "sample" to select some data point to train.

Following is the implementation of SGD, "data" is the RDD and "self.ratio" is the sampling ratio:

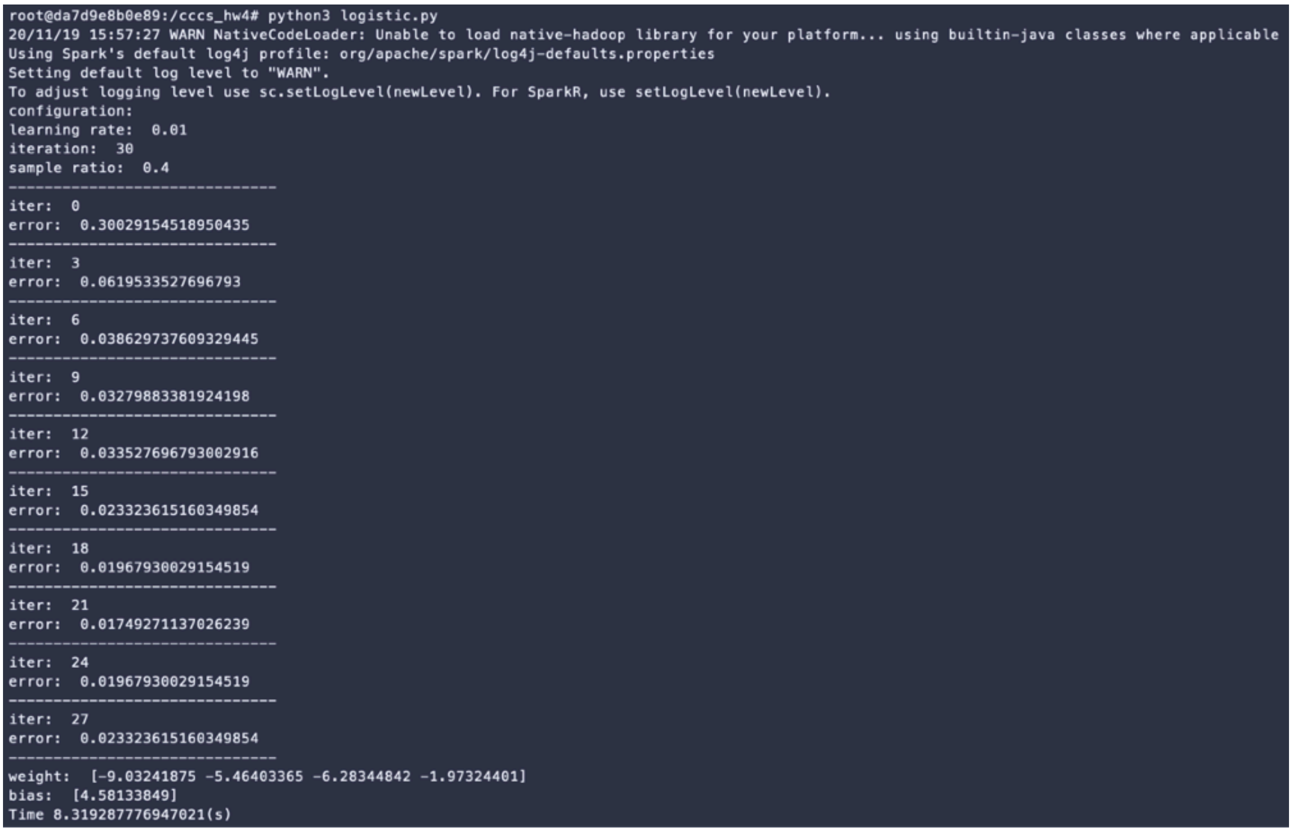


```
sub_data = data.sample(False,self.ratio)
```

I set my parameter as following:
learning rate: 0.01,
iteration: 30,
sample ratio: 0.4

and following is the result of self implement of SGD logistic regression:
The error rate is 0.02332, and it took 8.3 second to process.

Self implementation may get the lower error rate, because we can tuning more parameter than spark version. However, spark pre-implement may be faster.



```
root@da7d9e8b0e89:/cccs_hw4# python3 logistic.py
20/11/19 15:57:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
configuration:
learning rate:  0.01
iteration:  30
sample ratio:  0.4
------------------------------
iter:  0
error:  0.30029154518950435
------------------------------
iter:  3
error:  0.0619533527696793
------------------------------
iter:  6
error:  0.038629737609329445
------------------------------
iter:  9
error:  0.03279883381924198
------------------------------
iter:  12
error:  0.033527696793002916
------------------------------
iter:  15
error:  0.023323615160349854
------------------------------
iter:  18
error:  0.01967930029154519
------------------------------
iter:  21
error:  0.01749271137026239
------------------------------
iter:  24
error:  0.01967930029154519
------------------------------
iter:  27
error:  0.023323615160349854
------------------------------
weight:  [-9.03241875 -5.46403365 -6.28344842 -1.97324401]
bias:  [4.58133849]
Time 8.319287776947021(s)
```

3. Source project in github (URL) :

https://github.com/how123480/simple-spark