University of North Carolina at Charlotte

ITCS 8190 – Cloud Computing For Data Analysis – Project Report

Implementing Distributed Logistic Regression with Mini-Batch Stochastic Gradient Descent in PySpark for Ad Click Prediction

By
Archit Parnami
(aparnami@uncc.edu)

1. Project Overview

This project focuses on how to leverage distributed computing environment for machine learning. Specifically in this project, I have implemented logistic regression with mini-batch stochastic gradient descent in a distributed fashion using PySpark for Ad Click prediction. The data is obtained from a Kaggle challenge and is of size 6 GB (uncompressed). It has 40 million records and 24 categorical features. For training the data is processed to randomly sample 1 million records and transforms categorical features to 8300 one-hot encoded features.

2. Context & Motivation

Online advertisement is a multi-billion dollar industry where companies target consumers of the internet who would be interested in their products and services. Online advertising companies such as Google & Facebook allow advertisers to display ads on their platform (YouTube, Google Search & Facebook). The advertising companies then can charge their customers anytime a user clicks on their Ad. Hence it is in great interest of advertising companies to estimate the likelihood that a user would click on a displayed Ad.

Apart from its financial importance to companies, ad click prediction in itself is a challenging machine learning problem to solve, mostly because of the high sparsity and the volume of the data. Hence in order to train a machine learning model with high dimensionality it becomes essential to move away from traditional computing and towards distributed computing environments.

3. Datasets

I have made use of two datasets for this project.

1. **Breast Cancer Wisconsin(Diagnostic) Data Set**
   - Source: UCI Machine Learning Repository
   - Number of Features: 32
   - Number of Instances: 569
   - Data Characteristics: Multivariate
   - Task: Classification
   - Purpose: To verify the correctness of implementation of logistic regression algorithm in Python.

2. **Avazu Click-Through Rate Prediction Data Set**
   - Source: Kaggle
   - Number Features: 24
   - Number of Instances: ~40 Million
   - Data Characteristics: Multivariate, Categorical
   - Task: Classification
   - Purpose: To solve the classification challenge and to verify the correctness of implementation of logistic regression in distributed fashion using PySpark.

Attribute	*Description*	*Unique Values*	*Type*
id	Ad id	28300275	Identifier
click	0/1 Click or No Click		Categorical
hour	YYMMDDHH format	14091123	Categorical
banner_pos	position of banner in application	7	Categorical
site_id	ID given to site where ad is posted	4555	Categorical
site_domain	Domain of the site where ad is posted	7087	Categorical
site_category	Each site is divided in various categories	26	Categorical
app_id	ID of the application where ad is posted	7926	Categorical
app_domain	Domain of the application where ad is posted	510	Categorical
app_category	Application is divided in various categories	34	Categorical
device_id	Each device has a unique ID	2134670	Categorical
device_ip	IP of the device used by user	5604672	Categorical
device_model	Model of the device used by user	7944	Categorical
device_type	Each device is divided in various device types	5	Categorical
device_conn_type	Connection type of the device	4	Categorical
C1, C14-C21	Anonymized variables given by the company		Categorical

4. Algorithm

- **Logistic Regression and Gradient Descent**

For a given hypothesis:

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

A logistic cost function can be formulated as:

$$J(\theta) = -\tfrac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

Where:

$$\begin{aligned} x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\ x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\ m &= \text{the number of training examples} \\ n &= \left|x^{(i)}\right|; \text{(the number of features)} \end{aligned}$$

Then the cost function can be vectorized as:

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1 - y)^T \log(1 - h)\right)$$

Gradient Descent is performed as:

$$Repeat\ \{ \quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \}$$

$$Repeat\ \{ \quad \theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad \}$$

- **Stochastic Gradient Descent**

Stochastic gradient descent is an alternative to classic(or batch) gradient descent and is more efficient and scalable to large data set. It is written out in a different by similar way:

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

J_{train} is now just the average of the cost applied to all of our training examples.

The algorithm is as follows:
1. Randomly shuffle the dataset.
2. For i=1……m

$$\Theta_j := \Theta_j - \alpha(h_\Theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

- **Mini – Batch Gradient Descent**
  Mini-batch gradient descent can sometimes be even faster than stochastic gradient descent. Instead of using all m examples as in batch gradient descent, and instead of using only 1 example as in stochastic gradient descent, we will use some in-between number of examples b.

  For example, with b=10 and m=1000:

  Repeat:

  For $i = 1, 11, 21, 31, \ldots, 991$

$$\theta_j := \theta_j - \alpha\frac{1}{10}\sum_{k=i}^{i+9}(h_\theta(x^{(k)}) - y^{(k)})x_j^{(k)}$$

- **Distributed Implementation of Mini – Batch Stochastic Gradient Descent**
  In a distributed environment, we can divide up batch gradient descent and dispatch the cost function for a subset of the data to many different machines so that we can train our algorithm in parallel. We can split our training set into z subsets corresponding to the number of machines we have. On each of those machines calculate

  $\sum_{i=p}^{q}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$, where we've split the data starting at p and ending at q.

  Map reduce will take all these dispatched(or 'mapped') jobs and reduce them by calculating:

$$\Theta_j := \Theta_j - \alpha\frac{1}{z}(temp_j^{(1)} + temp_j^{(2)} + \cdots + temp_j^{(z)})$$

For all j=0,….., n.

- **Regularization**
  - Regularization is designed to address the problem of overfitting.
  - High bias or underfitting is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features. e.g. if we take $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ then we are making an initial assumption that a linear model will fit the training data well and will be able to generalize but that may not be the case.
  - At the other extreme, overfitting or high variance is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.
  - We can regularize cost function of logistic regression by adding a term to the end:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1-y^{(i)})\ \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

  - Then gradient descent will be performed as:

```
Repeat {
    $\theta_0 := \theta_0 - \alpha\,\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$
    $\theta_j := \theta_j - \alpha\left[\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right]$        $j \in \{1, 2...n\}$
}
```

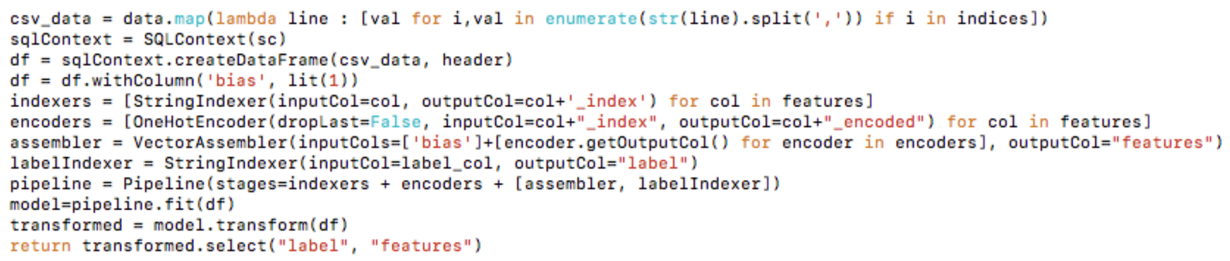
5. Implementation in PySpark

The following steps are involved in building and training the model for Ad Click Prediction using PySpark.

a) Data Preprocessing

The given dataset has 24 features and all of them are categorical values. Therefore in order to perform Logistic Regression on this dataset, these features needs to be converted into one-hot encoding.

- Create an RDD from input data.
- Apply map transformation on RDD to filter the required features
- Create a dataframe from the RDD
- For each feature apply the following transformations:
  - Create a StringIndexer()
  - Create a OneHotEncoder()
- Use VectorAssembler() to combine all the features
- Fit and Transform dataframe.



```
csv_data = data.map(lambda line : [val for i,val in enumerate(str(line).split(',')) if i in indices])
sqlContext = SQLContext(sc)
df = sqlContext.createDataFrame(csv_data, header)
df = df.withColumn('bias', lit(1))
indexers = [StringIndexer(inputCol=col, outputCol=col+'_index') for col in features]
encoders = [OneHotEncoder(dropLast=False, inputCol=col+"_index", outputCol=col+"_encoded") for col in features]
assembler = VectorAssembler(inputCols=['bias']+[encoder.getOutputCol() for encoder in encoders], outputCol="features")
labelIndexer = StringIndexer(inputCol=label_col, outputCol="label")
pipeline = Pipeline(stages=indexers + encoders + [assembler, labelIndexer])
model=pipeline.fit(df)
transformed = model.transform(df)
return transformed.select("label", "features")
```

Figure 1 Spark Pipeline for Data Preprocessing

- Sampling: 1010724 from 40428968

- Excluded Features: ['id', 'hour', 'site_id', 'site_domain', 'app_id', 'app_domain', 'device_id', 'device_ip']

- Included Features: ['C1', 'banner_pos', 'site_category', 'app_category', 'device_model', 'device_type', 'device_conn_type', 'C14', 'C15', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21']

- Data Schema:

  ```
  root
   |-- label: double (nullable = true)
   |-- features: vector (nullable = true)
  ```

  Sample data:

  ```
  +-----+--------------------+
  |label|            features|
  +-----+--------------------+
  |  0.0|(8300,[0,1,8,15,3...|
  |  0.0|(8300,[0,1,8,16,3...|
  |  0.0|(8300,[0,1,8,17,3...|
  |  1.0|(8300,[0,1,8,17,3...|
  |  0.0|(8300,[0,1,9,16,3...|
  +-----+--------------------+
  ```

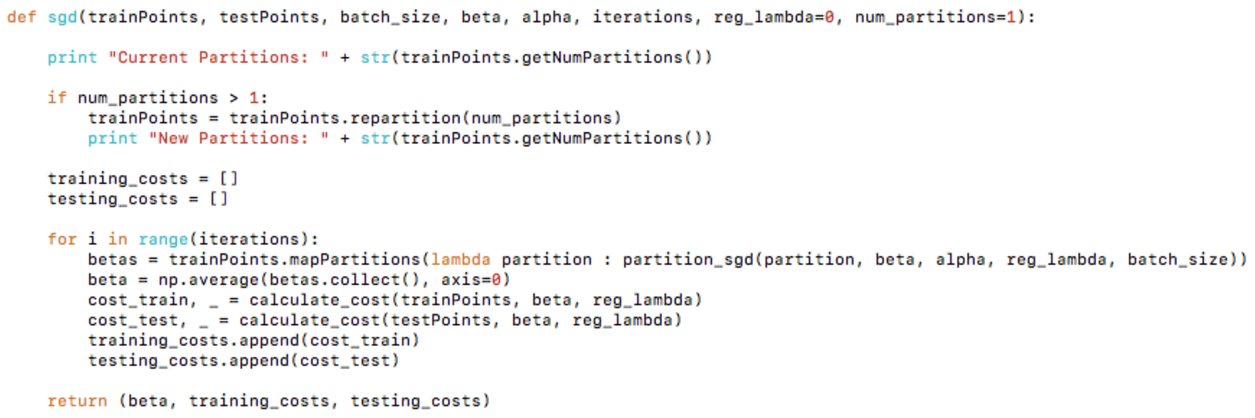
- After converting categorical features into one-hot encoded features, the number of dimensions increased to 8300. After sampling 2.5% of the dataset, the size of the sampled dataset has changed to **1010724 x 8300.**

- The processed data is the split into **80:20** ratio for training and testing.

b) Training

- The training data (**808579 x 8300)** is distributed to **200 partitions** on the cluster. Each partition performs mini batch stochastic gradient descent with batch size of 200 samples per iteration. Results from each partition is then aggregated to get updated parameters.

- Because of the large size of the data it takes about 69 seconds to complete one iteration for whole dataset and therefore a total of 2 hours in training for 100 iterations.



```
def sgd(trainPoints, testPoints, batch_size, beta, alpha, iterations, reg_lambda=0, num_partitions=1):

    print "Current Partitions: " + str(trainPoints.getNumPartitions())

    if num_partitions > 1:
        trainPoints = trainPoints.repartition(num_partitions)
        print "New Partitions: " + str(trainPoints.getNumPartitions())

    training_costs = []
    testing_costs = []

    for i in range(iterations):
        betas = trainPoints.mapPartitions(lambda partition : partition_sgd(partition, beta, alpha, reg_lambda, batch_size))
        beta = np.average(betas.collect(), axis=0)
        cost_train, _ = calculate_cost(trainPoints, beta, reg_lambda)
        cost_test, _ = calculate_cost(testPoints, beta, reg_lambda)
        training_costs.append(cost_train)
        testing_costs.append(cost_test)

    return (beta, training_costs, testing_costs)
```

Figure 2 Distributing training data to different partitions for stochastic gradient descent

- The learned parameters (beta) are then written to a file for testing.

c) Prediction

- The testing data (**20344 x 8300)**, along with the beta **parameter** is loaded and evaluated using Map Reduce.

6. Results & Evaluation

- In order to evaluate the correctness of the Logistic Regression implementation, we are first going to evaluate the performance on **Breast Cancer Diagnostic** Data Set.

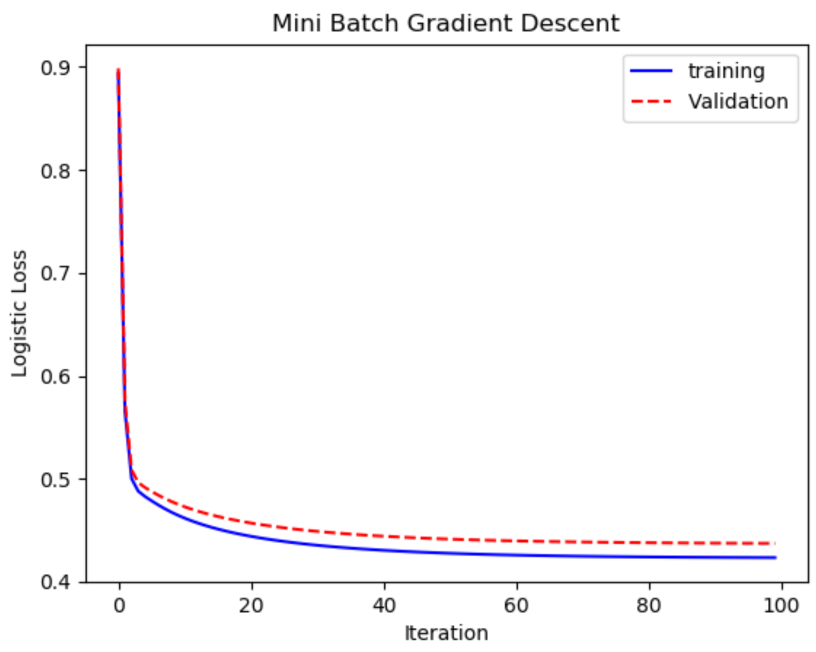
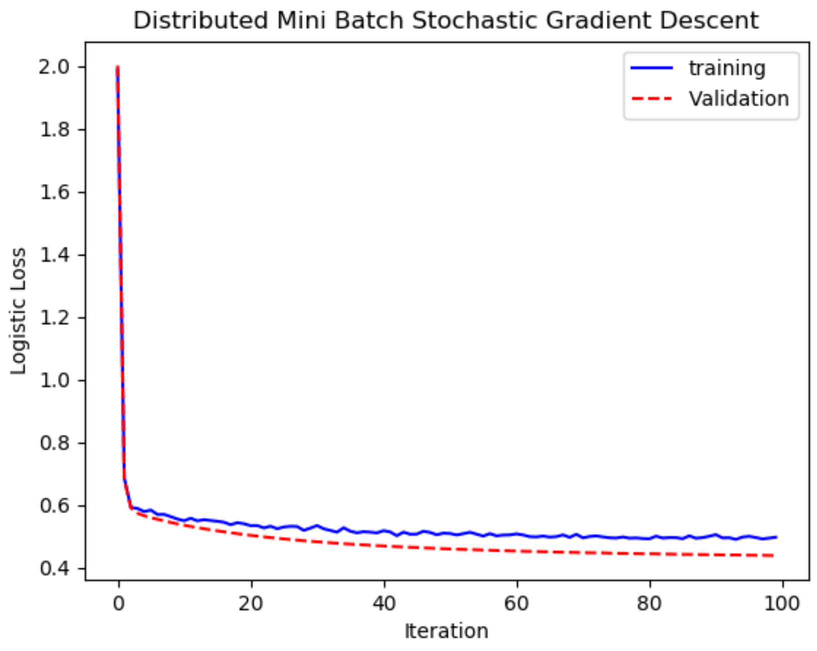


Figure 3 A sample run of gradient descent on Breast Cancer Prediction Dataset. Decreasing cost with iteration suggests that the gradient descent is working correctly.

Iterations	**Learning Rate**	**Regularization Parameter**	**Batch Size**	**Training Cost**	**Validation Cost**	**Classification Accuracy**
100	0.03	1	10	0.42	0.43	0.87
200	0.05	1	20	0.35	0.37	0.90
200	0.05	1	40	0.31	0.31	0.93
1000	0.05	1	40	0.29	0.29	0.94

- Next we evaluate the implementation of Logistic Regression in PySpark for **Ad Click Prediction** dataset.



  - Logistic Loss / Cost: 0.437591703032
  - Accuracy: **82.9849887961**

7. Project Completion Status

	DEFINITELY ACCOMPLISH	LIKELY ACCOMPLISH	IDEALLY ACCOMPLISH
STATED AT BEGINNING OF THE PROJECT	Implementation of Logistic Regression in PySpark with mini batch gradient descent.	The scale of the dataset might be problematic in training the model. So I might use scaled down version of the dataset.	A logistic regression model trained on the entire dataset to perform ad-click prediction.
STATUS	Completed	Completed	Yet to Accomplish

8. Comments & Observations

a. Challenges
- Data preprocessing was very challenging and interesting part about this project especially considering the high number of dimensions.
- Tuning the number of partitions on cluster to avoid reaching memory limit was interesting.
- One processing large datasets, feature engineering becomes important more than ever.

b. Surprises
- Bugs in production code can lead to lots of waste time in training and re-training.

c. Cool Enhancements
- Spark's feature to create a process flow **pipeline** which involves joining together different transformations + Machine Learning as one single operation on data is amazing.

9. References

- Kaggle Competition: https://www.kaggle.com/c/avazu-ctr-prediction/data
- Breast Cancer Wisconsin(Diagnostic) Data Set https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
- Algorithm and Theory https://www.coursera.org/learn/machine-learning/