Dr. Marc Sanchez-Artigas
marc.sanchez@urv.cat

Secure Distributed Systems Architectures 2024/25

Assignment#3

Guidelines

- Please submit your solutions by **Friday, January 17, 23:55 PM** through Moodle. Only electronic submission is accepted. Please only use PostScript or PDF. We must be able to print out your submissions on a standard monochrome printer so that they are fully readable.
- This assignment can be solved in **teams of two or three students**.
- By submitting any processed exercise you hereby commit to Ethics rules promulgated by URV (see http://wwwa.urv.cat/la_urv/3_organs_govern/secretaria_general/legislacio/2_propia/comunitat/estudiants/inst_regl_reg_disc_est.pdf). This means that you should try to write the answers in your own words. However, if you use external materials, you have to cite them correctly.

1. Overview

The goal of this assignment is to implement a distributed version of an ML algorithm. **You can propose** which ML problem to solve. For those students who are double minded in choosing an algorithm, an illustrative example is logistic regression with stochastic gradient descent.

2. Logistic regression with SGD

Logistic regression is a very popular technique borrowed by Machine Learning (ML) from the field of statistics. Indeed, it can be viewed as the `go-to` method for binary classification problems (problems with two class values).

In addition to the implementation, this problem requires a good understanding of what logistic regression is. Consequently, **it may require significant self-learning by the students if they are newcomers to ML**. For starters, you can find here a good description of logistic regression: https://en.wikipedia.org/wiki/Logistic_regression.

2.1. Objective

In this assignment, you will try to predict the probability that a person earns more than $50k per year in the **Adult** dataset (http://www.cs.toronto.edu/~delve/data/adult/adultDetail.html).

To help you with this task, you will have available the skeleton of some functions. You will have to modify the `sgd.py` file and adapt it to run logistic regression in a distributed manner. In this file, you will have several functions that you should implement, starting from the `logistic` function up to the `train` and `update` functions used for training the model. Very importantly, and as can been seen in the parameters of the `train` function, you should implement **logistic regression with L2 regularization**, where `rate` denotes the learning rate and `lam` is the **L2 regularization parameter**:

```python
# TODO: Train model using training data
def train(data, epochs, rate, lam):
    model = initialize_model(len(data[0]['features']))
    return model
```

The training should run for some number of `epochs` performing **Stochastic Gradient Descent (SGD)** [2].

2.2. Sequential version of SGD

A non-parallel version of SGD consists of the following steps: first, a point is randomly selected from the dataset. Next, the error for that point is calculated and the model weights are adjusted based on the gradient of that error. This is different from Batch Gradient Descent, where a full pass over the dataset, i.e., a complete epoch, is made before updating the model. SGD converges faster but can also be less stable because you have a noisy estimate of the gradient instead of the true gradient. In practice, it is often much better to use SGD than full batch gradient descent.

To implement SGD in a distributed manner, each worker will evolve a local model by performing B sequential SGD updates corresponding to B randomly chosen points (**minibatch**), before communicating its local model to the server, who will finally average all the model weights. To be more precise, Figure 3 contains the pseudo code for the parallel distributed implementation of SGD.

```
Define $\mathrm{B} = N/m$, where $m$ is the number of workers and $N$ is the number of data points.
Initialize the model weights $w_j$ (use function initialize_model(k) in the file sgd.py)
1:  for all epochs do:
2:      randomly partition the points, giving $\mathrm{B}$ points to each worker
3:      for all $i = 1$ to $m$ parallel do
4:          fetch the model weights $w_j$ from the server
5:          for all $b = 1$ to $\mathrm{B}$ do
6:              take the $b^{th}$ point on worker $i$
7:              update the model weights with the gradient
8:          end for
9:          send the local weights $w_{j(i)}$ to the server
10:     end for
11:     server averages all the weights from the workers: $w_j = (1/m) \sum_{i=1..m} w_{j(i)}$
12: end for
```

Figure 3. Detailed steps for distributed SGD.

In this specific assignment, a prediction for a new data point p will be:

```
prediction = P(income > 50k | w, p) = logistic(w·p)
```

while the loss or error function will be:

```
loss = (prediction - true_label{1 or 0})^2 + lam*||w||
```

There are two components to the loss:

- The squared error from `(prediction - true_label{1 or 0})^2`
- the loss from regularization, `lam*||w||`

By minimizing the `loss`, the model learns to make correct predictions and use small weights to avoid overfitting. To adjust the model, you will need calculate the gradient of the loss function at a given point. The gradient will come from two sources, namely, the error and the regularization.

3. Evaluation

Tests for your program are in `test.py`. To test your implementation, run:

```
python test.py
```

You should see the number of tests your implementation passes and any problems that could arise. This file will also tell the current accuracy of the training and validation set. **By default, the validation set is also the complete dataset!** To get a more accurate evaluation you can modify `data.py` to use different training and validation sets by splitting your data.

Finally, you should study at least the following two aspects of your serverless implementation:

1. How varies the accuracy of the model as function of the number of epochs.

2. Since the goal of this task is to implement a parallel version of Logistic Regression over Lithops, **you can experiment with different number of workers** `m`, **and with different batch sizes** `B`. As metric, you can use the execution time, or even the speed-up, defined as the ratio:

   Speedup = (Execution time for 1 worker)/(Execution time for `m` workers).

Finally, plot the results as a function of these parameters.

4. Submission

Please, submit a three-page document describing your proposed solution along with the asked plots. Further, an in-depth explanation of the results is MANDATORY. The document should also include full references for the papers and other sources that you have consulted. If you are working in a group, you must say who has been responsible for each portion of the work.

Finally, create a `.zip` file called `SDSA_A3_name1_name2_name3.zip`, where `name1`, `name2` and `name3` will be the family name of each component of the group. Include in the `.zip` file the document in `.pdf` format along with all the Python code.

References

[1] - Josep Sampé, Gil Vernik, Marc Sánchez-Artigas, Pedro García-López, "Serverless Data Analytics in the IBM Cloud". Middleware Industry 2018: 1-8
[2] - Stochastic Gradient Descent, https://en.wikipedia.org/wiki/Stochastic_gradient_descent