Big Data Applications for IoT (Master in IoT)

Practical Assessment Task (2021-22)

Parallel Implementation and Evaluation of a supervised ML algorithm

Description of tasks to be completed

Using the labelled dataset "botnet_tot_syn_l.csv", implement a parallel version of the logistic regression classifier on Spark with Python.

Implement parallel versions of readFile, normalize, train and accuracy functions:

```
def readFile (filename):
    Arguments:
    filename – name of the spam dataset file
       12 columns: 11 features/dimensions (X) + 1 column with labels (Y)
                   Y -- Train labels (0 if normal traffic, 1 if botnet)
        m rows: number of examples (m)

    Returns:
    An RDD containing the data of filename. Each example (row) of the file
    corresponds to one RDD record. Each record of the RDD is a tuple (X,y).
    "X" is an array containing the 11 features (float number) of an example
    "y" is the 12th column of an example (integer 0/1)

def normalize (RDD_Xy):
    Arguments:
    RDD_Xy is an RDD containing data examples. Each record of the RDD is a tuple
    (X,y).
    "X" is an array containing the 11 features (float number) of an example
    "y" is the label of the example (integer 0/1)

    Returns:
    An RDD rescaled to N(0,1) in each column (mean=0, standard deviation=1)


def train (RDD_Xy, iterations, learning_rate):
    Arguments:
    RDD_Xy --- RDD containing data examples. Each record of the RDD is a tuple
    (X,y).
    "X" is an array containing the 11 features (float number) of an example
    "y" is the label of the example (integer 0/1)
    iterations -- number of iterations of the optimization loop
    learning_rate -- learning rate of the gradient descent


    Returns:
    A list or array containing the weights "w" and bias "b" at the end of the
    training process

def accuracy (w, b, RDD_Xy):
    Arguments:
    w -- weights
    b -- bias
    RDD_Xy – RDD containing examples to be predicted

    Returns:
    accuracy -- the number of predictions that are correct divided by the number
    of records (examples) in RDD_xy.
    Predict function can be used for predicting a single example
```

```
def predict (w, b, X):
    Arguments:
    w -- weights
    b -- bias
    X – Example to be predicted

    Returns:
    Y_pred – a value (0/1) corresponding to the prediction of X
```

The train procedure will be implemented using Gradient Descent. You can use the implementation below but are encouraged to do research to find possible alternatives, optimizations or improvements of any kind.

initialize $w_1;\ w_2;\ \ldots\ w_n;\ b$
$dw_1;\ dw_2;\ \ldots\ dw_n;\ db$
for it in range $(iterations)$:
compute $dw_1;\ dw_2;\ \ldots\ dw_n;\ db$
$w_1 = w_1 - \alpha * dw_1$
$w_2 = w_2 - \alpha * dw_2$
$\ldots$
$w_k = w_k - \alpha * dw_k$
$b = b - \alpha * db$

Cost function:

$$J(W) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)}))$$

Derivatives

$$dw_1 = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_1^{(j)}$$
$$dw_2 = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_2^{(j)}$$
$$\ldots$$
$$dw_k = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_k^{(j)}$$

$$db = \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)})$$

(1) Code the centralised version of readFile, normalize, train and accuracy functions and check that the algorithm converges during training (i.e., the cost value decreases asymptotically at each iteration)

In the centralized version substitute RDDs by numpy arrays in the input parameters and return values of the proposed functions.

Use the below main code for testing the whole system:

```
# read data
data= readFile(path)

# standardize
data=normalize(data)

ws = train(data, nIter, learningRate)
acc = accuracy(data, ws)
print ("acc:",acc)
```

(2) Code the parallel versions of readFile, normalize, train and accuracy functions. Test the whole system using the previous main code.

- Print the value of the cost function at the end of each gradient descent iteration to observe how the training process is converging (i.e., the cost value decreases asymptotically).
- Try to parallelize with Spark computations that involve processing all the elements of the dataset.
- Do not apply the "collect" method to any potentially big data RDD.

Optional work (Written Report)

- Explore different values for the learning rate ("alpha" hyperparameter) and plot training and testing costs vs iterations for the final model. Comment critically the results.
- Plot performance and speedup curves.
  The performance curve shows execution time (y axis) versus number of workers (x axis). The speedup curve shows the ratio (running time using 1 worker) /(running time using n workers) versus the number of workers. Comment critically the results with respect to parallelization.

Note that ONLY the Spark functions seen in class can be used for section 2. Do not use complex functions. Only *map, flatMap, reduce and reduc*eByKey are allowed. In case you need to use another function, send me an email explaining why you need this extra function.

Hints:
- Convert python lists to numpy arrays and use numpy vectorized operations (e.g., dot, multiply) for operating vectors and matrices and speed up the computations.

Two separate jupyter notebooks will be delivered for the centralized and parallelized versions. The written report (pdf) is not mandatory.