Master EIT Digital in Data Science

Massively Parallel Machine Learning

Practical Assessment Task (2019-20)

Parallel Implementation and Evaluation of a supervised ML algorithm

Description of tasks to be completed

Using the labelled dataset "spam.data" that can be downloaded from https://web.stanford.edu/~hastie/ElemStatLearn/data.html, implement a parallel version of the logistic regression classifier on Spark with Python.

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

```
def readFile (filename):
    Arguments:
    filename – name of the spam dataset file
       58 columns: 57 features/dimensions (X) + 1 column with labels (Y)
                   Y -- Train labels (0 if no spam, 1 if spam)
        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 57 features (float number) of an example
    "y" is the 58th column of an example (integer 0/1)

def standardize (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 57 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, lambda_reg):
    Arguments:
    RDD_Xy --- RDD containing data examples. Each record of the RDD is a tuple
    (X,y).
    "X" is an array containing the 57 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
    lambda_reg – regularization rate

    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.

$$
\begin{aligned}
&\text{initialize } w_1;\ w_2;\ \dots\ w_n;\ b \\
&\qquad\qquad dw_1;\ dw_2;\ \dots\ dw_n;\ db \\
&\text{for } it \text{ in range } (iterations): \\
&\quad \text{compute } dw_1;\ dw_2;\ \dots\ dw_n;\ db \\
&\quad w_1 = w_1 - \alpha * dw_1 \\
&\quad w_2 = w_2 - \alpha * dw_2 \\
&\quad \dots \\
&\quad w_k = w_k - \alpha * dw_k \\
&\quad b = b - \alpha * db
\end{aligned}
$$

Cost function:

$$J(W) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)})) + \frac{\lambda}{2m}\sum_{i=1}^{k} w_i^2$$

Derivatives

$$
\begin{aligned}
dw_1 &= \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_1^{(j)} + \frac{\lambda}{m}w_1 \\
dw_2 &= \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_2^{(j)} + \frac{\lambda}{m}w_2 \\
&\dots \\
dw_k &= \frac{1}{m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)}) * x_k^{(j)} + \frac{\lambda}{m}w_k
\end{aligned}
$$

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

(1) Code the centralised version of readFile, standardize, 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 following 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, standardize, train and accuracy functions. Test the whole system using the previous main code.

(3) Implement a procedure to perform cross validation. You can read only one time the file from disk. Utilise the following pseudo code as template for the cross validation process:

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

# standardize
data=normalize(data)

num_blocks_cv=10
# Shuffle rows and transform data
data_cv= transform (data)

for i in range(num_blocks_cv):
    tr_data, test_data = get_block_data (data_cv,i)
    ws = train(tr_data, nIter, learningRate)
    acc = accuracy(test_data, ws)
    print ("acc:",acc)

print ("average acc:", avg_acc)
```

The purpose of cross validation is to estimate the performance of the model and to choose the best set of hyperparameters. You can modify hyperparameter (in this context the amount of regularization, lambda) to trade off between bias and variance

Hints:

- Normalize (or scale) the dataset before training the model. If possible parallelize this process.
- 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.
- 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.

Report structure

Written report including the following sections: Introduction, data set description, algorithm implementation, cross validation procedure, experiments (including performance and speedup curves as well as error metrics of the machine learning tasks), and conclusions.

The performance curve shows execution time versus number of workers. 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.

Plots of training and testing costs vs iterations for the final models can also be included.

The machine learning error metrics to measure are the ones seen in class for classification: cost error, accuracy, precision, recall, f1-score, confusion matrix and ROC and Precision-Recall curves. At the light of this metrics, comment critically the obtained results.

Oral presentation: set of slides

In addition to the written report, three separate jupyter notebooks will be delivered for the centralized, parallelized and cross validation versions. The set of slides for the oral presentation should be sent by email before this presentation.