CS777

Big Data Analytics

Spring 2021

Professor Kia Teymourian

Assignment 4

Kunfei Chen

U15575304

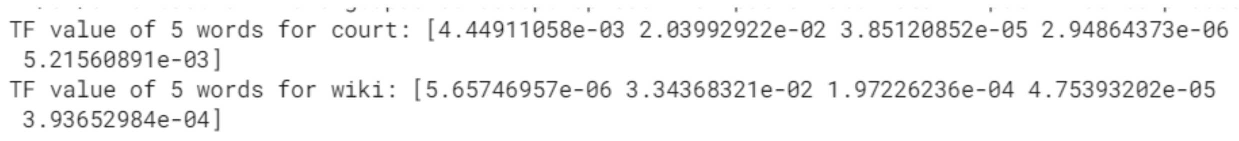
Introduction

This project aims to apply Batch Gradient Descent via PySpark to achieve logistic regression to classify text documents. The whole data set consists of two categories which are Wikipedia pages and Australian court cases.

Task 1 – Data Preparation and Model Training

Firstly, the total IDF array is calculated based on the training set, which represents the idf value of 20000 top words. Then, the TF arrays of both training set and testing set are calculated as well. In terms of the following words: “applicant”, “and”, “attack”, “protein”, and “court”, their average tf value for court documents are 4.45e-03, 2.04e-02, 3.85e-05, 2.95e-06, 5.22e-03, and the average tf value for Wikipedia documents are 5.66e-06, 3.34e-02, 1.97e-04, 4.75e-05, 3.94e-04 (See Figure 1).

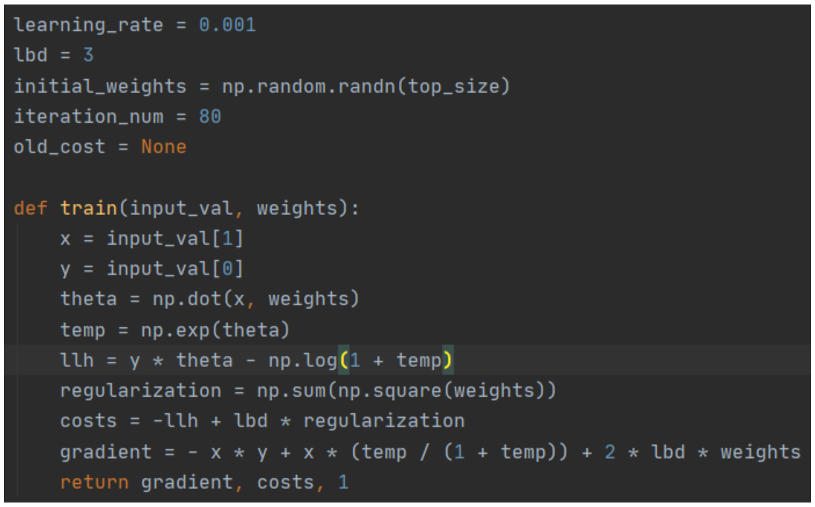
Figure 1



```
TF value of 5 words for court: [4.44911058e-03 2.03992922e-02 3.85120852e-05 2.94864373e-06
 5.21560891e-03]
TF value of 5 words for wiki: [5.65746957e-06 3.34368321e-02 1.97226236e-04 4.75393202e-05
 3.93652984e-04]
```

Then, a logistic regression model is built for the training (See Figure 2). The learning rate is set to be 0.001, the lambda parameter of L2 regularization is set to be 3. The initial weights is generated by the random function of numpy in which the value follows the normal distribution. The model is trained for 100 interation.

Figure 2

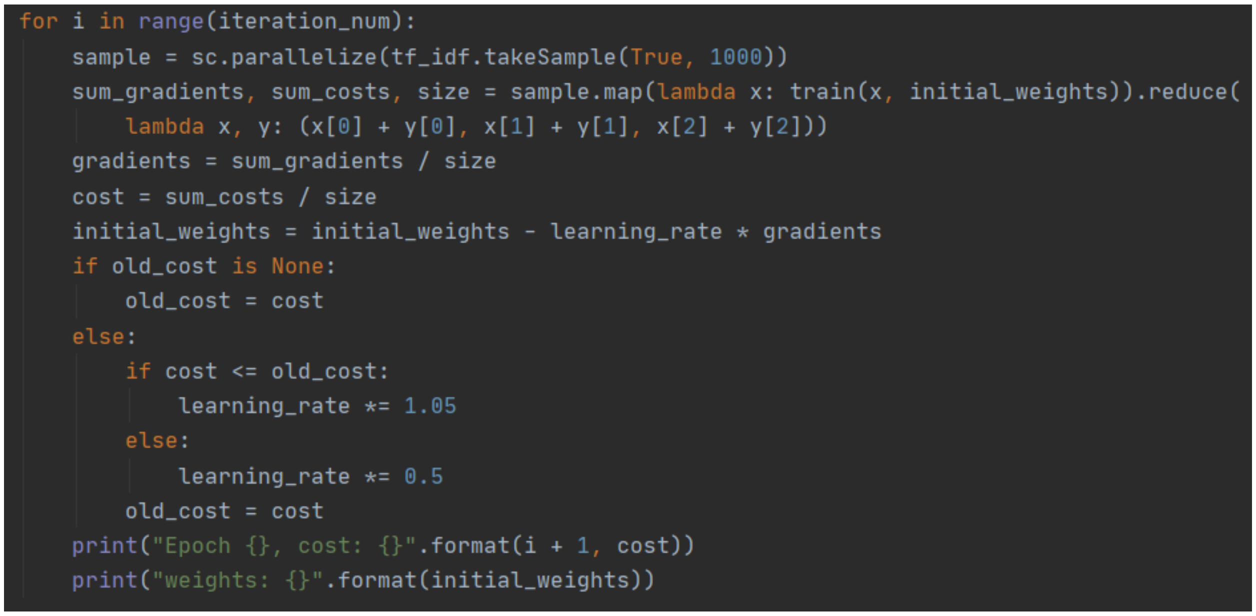


```
learning_rate = 0.001
lbd = 3
initial_weights = np.random.randn(top_size)
iteration_num = 80
old_cost = None

def train(input_val, weights):
    x = input_val[1]
    y = input_val[0]
    theta = np.dot(x, weights)
    temp = np.exp(theta)
    llh = y * theta - np.log(1 + temp)
    regularization = np.sum(np.square(weights))
    costs = -llh + lbd * regularization
    gradient = - x * y + x * (temp / (1 + temp)) + 2 * lbd * weights
    return gradient, costs, 1
```

Since this is a batch gradient descent model, for each iteration, 1000 samples are taken from the whole dataset for the training. In addition, the training of this model applys "Bold Driver" technique to dynamically change the learning rate (See Figure 3).

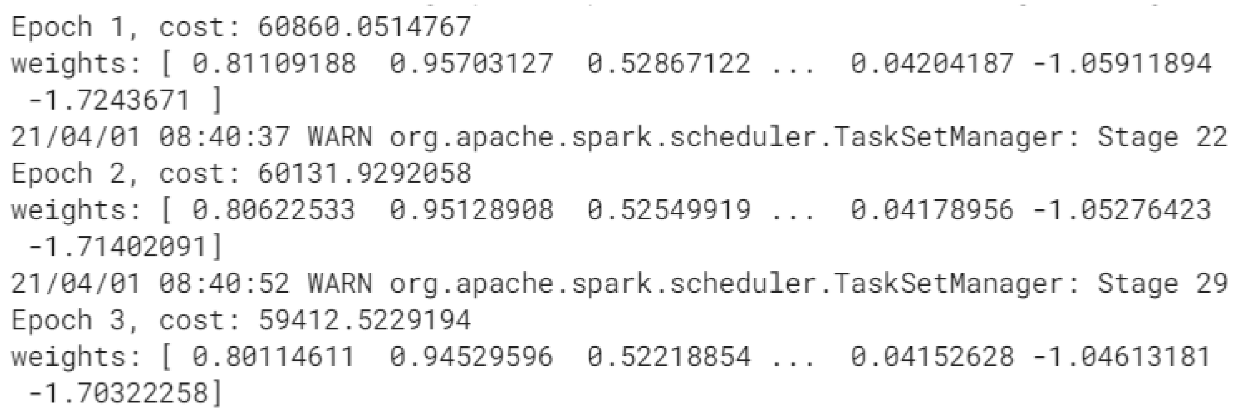
Figure 3



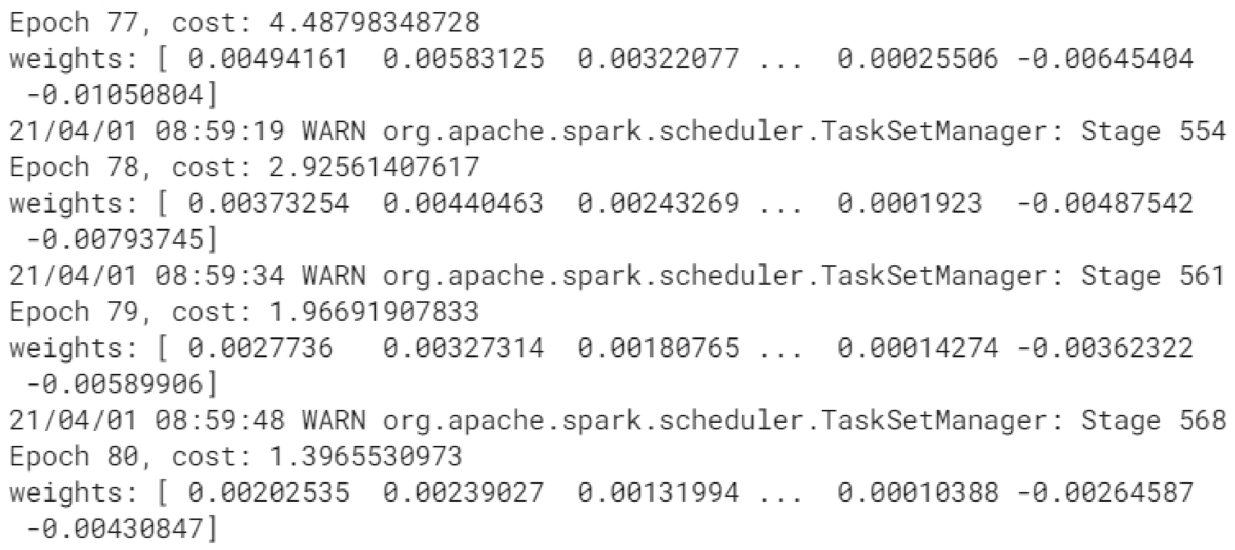
```
for i in range(iteration_num):
    sample = sc.parallelize(tf_idf.takeSample(True, 1000))
    sum_gradients, sum_costs, size = sample.map(lambda x: train(x, initial_weights)).reduce(
        lambda x, y: (x[0] + y[0], x[1] + y[1], x[2] + y[2]))
    gradients = sum_gradients / size
    cost = sum_costs / size
    initial_weights = initial_weights - learning_rate * gradients
    if old_cost is None:
        old_cost = cost
    else:
        if cost <= old_cost:
            learning_rate *= 1.05
        else:
            learning_rate *= 0.5
        old_cost = cost
    print("Epoch {}, cost: {}".format(i + 1, cost))
    print("weights: {}".format(initial_weights))
```

The costs and model parameters in each iteration is shown in Figure 4. It can be seen that after training 80 iterations, the cost decreased from 60860 to 1.397.

Figure 4



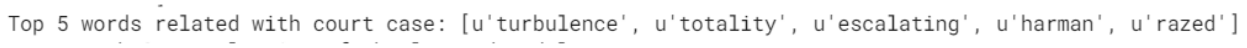
```
Epoch 1, cost: 60860.0514767
weights: [ 0.81109188  0.95703127  0.52867122 ...  0.04204187 -1.05911894
 -1.7243671 ]
21/04/01 08:40:37 WARN org.apache.spark.scheduler.TaskSetManager: Stage 22
Epoch 2, cost: 60131.9292058
weights: [ 0.80622533  0.95128908  0.52549919 ...  0.04178956 -1.05276423
 -1.71402091]
21/04/01 08:40:52 WARN org.apache.spark.scheduler.TaskSetManager: Stage 29
Epoch 3, cost: 59412.5229194
weights: [ 0.80114611  0.94529596  0.52218854 ...  0.04152628 -1.04613181
 -1.70322258]
```



```
Epoch 77, cost: 4.48798348728
weights: [ 0.00494161  0.00583125  0.00322077 ...  0.00025506 -0.00645404
 -0.01050804]
21/04/01 08:59:19 WARN org.apache.spark.scheduler.TaskSetManager: Stage 554
Epoch 78, cost: 2.92561407617
weights: [ 0.00373254  0.00440463  0.00243269 ...  0.0001923  -0.00487542
 -0.00793745]
21/04/01 08:59:34 WARN org.apache.spark.scheduler.TaskSetManager: Stage 561
Epoch 79, cost: 1.96691907833
weights: [ 0.0027736   0.00327314  0.00180765 ...  0.00014274 -0.00362322
 -0.00589906]
21/04/01 08:59:48 WARN org.apache.spark.scheduler.TaskSetManager: Stage 568
Epoch 80, cost: 1.3965530973
weights: [ 0.00202535  0.00239027  0.00131994 ...  0.00010388 -0.00264587
 -0.00430847]
```

After the training finished, the five words with the largest regression coefficients are shown in Figure 5, they are “turbulence”, “totality”, “escalating”, “harman”, “razed”.

Figure 5

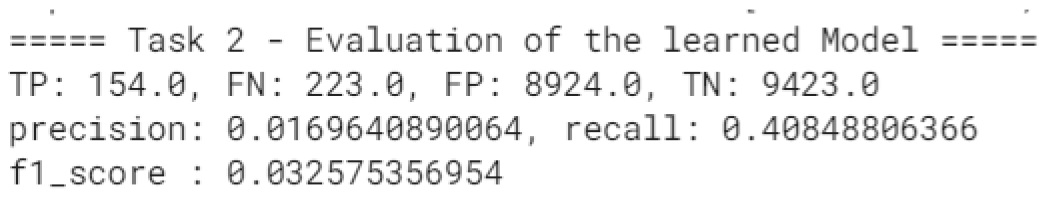


```
Top 5 words related with court case: [u'turbulence', u'totality', u'escalating', u'harman', u'razed']
```

Task 2 – Evaluating of the learned Model

Accurding to Figure 6, in order to evaluate the learned model, the four categories of predicted results, including “True Positive”, “False Negative”, “False Positive”, and “True Negative” are counted, in which their counted numbers are 154, 223, 8924, and 9423 respectivly. Based on it, the precision rate is calculated to be 1.7%, the recall rate is calculated to be 40.9%. The final F1 score is calculated to be 3.26%.

Figure 6

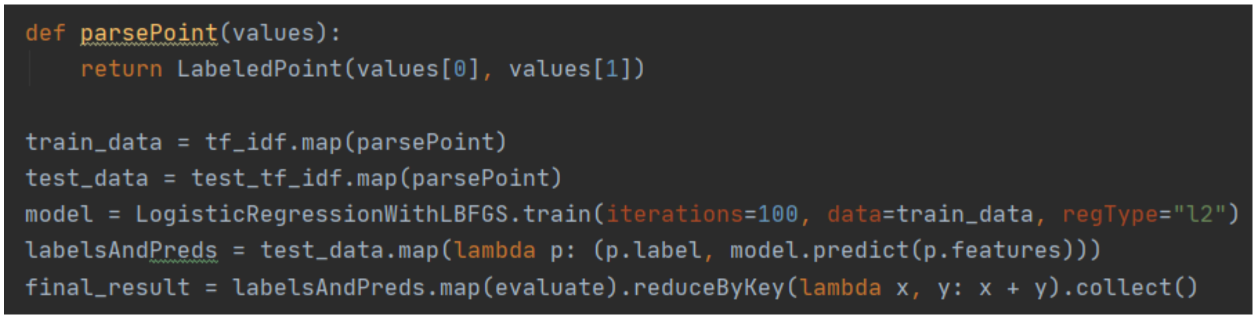


```
===== Task 2 - Evaluation of the learned Model =====
TP: 154.0, FN: 223.0, FP: 8924.0, TN: 9423.0
precision: 0.0169640890064, recall: 0.40848806366
f1_score : 0.032575356954
```

Task 13 – Using Spark Machine Learning Library

As shown in Figure 7, the LogisticRegressionWithBFGS function from the mllib library is applied for the training. The L2 regularization method is set and the model is trained for 100 iterations.

Figure 7

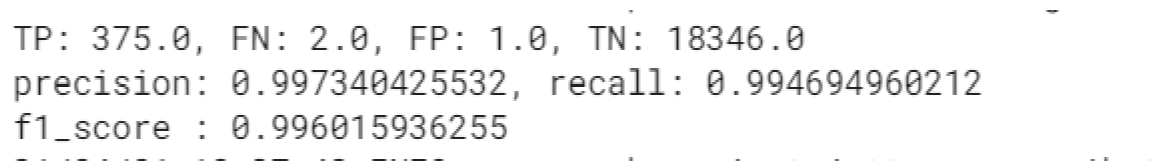


```
def parsePoint(values):
    return LabeledPoint(values[0], values[1])

train_data = tf_idf.map(parsePoint)
test_data = test_tf_idf.map(parsePoint)
model = LogisticRegressionWithLBFGS.train(iterations=100, data=train_data, regType="l2")
labelsAndPreds = test_data.map(lambda p: (p.label, model.predict(p.features)))
final_result = labelsAndPreds.map(evaluate).reduceByKey(lambda x, y: x + y).collect()
```

According to Figure 8, when this model is applied on small training data set, the four categories of predicted results are 375, 2, 1, and 18346 respectivly. Based on it, the precision rate is calculated to be 99.73%, the recall rate is calculated to be 99.47%. The final F1 score is calculated to be 99.6%.

Figure 8



```
TP: 375.0, FN: 2.0, FP: 1.0, TN: 18346.0
precision: 0.997340425532, recall: 0.994694960212
f1_score : 0.996015936255
```