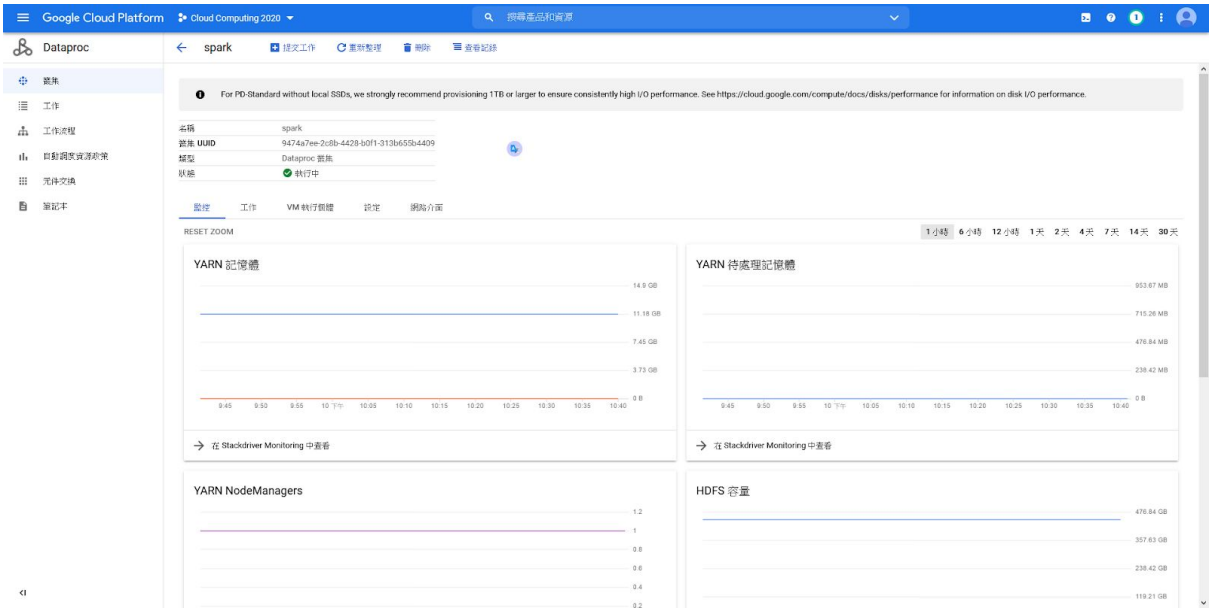
[HW4] Spark - Logistic Regression

r08942088 李政旻

(1) Reproduce the results using your own spark cluster (can be in standalone (maintained by yourself using docker containers, or in Google/AWS/Azure Cloud).

Same as HW3, I create a standalone cluster with 4 CPU and 15GB memory by google cloud dataproc. It will install Spark in the initial machine setup.



The original Spark logistic regression example is already out of date. So, I have made some changes in the code.

```python
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.mllib.regression import LabeledPoint
from pyspark import SparkConf, SparkContext

def getSparkContext():
    """
    Gets the Spark Context
    """
    conf = (SparkConf()
         .setMaster("local") # run on local
         .setAppName("Logistic Regression") # Name of App
         .set("spark.executor.memory", "1g")) # Set 1 gig of memory
    sc = SparkContext(conf = conf)
    return sc

sc = getSparkContext()

# Load and parse the data"
data = sc.textFile("data_banknote_authentication.txt")

def mapper(line):
    """
    Mapper that converts an input line to a feature vector
    """
    feats = line.strip().split(",")
    # labels must be at the beginning for LRSGD, it's in the end in our data, so
    # putting it in the right place
    label = feats[len(feats) - 1]
    feats = feats[: len(feats) - 1]
    #feats.insert(0,label)
    #features = [ float(feature) for feature in feats ] # need floats
    return LabeledPoint(label, feats)

parsedData = data.map(mapper)

# Train model
model = LogisticRegressionWithSGD.train(parsedData)

# Predict the first elem will be actual data and the second
# item will be the prediction of the model
labelsAndPreds = parsedData.map(lambda point: (int(point.label),
        model.predict(point.features)))

# Evaluating the model on training data
trainErr = labelsAndPreds.filter(lambda vp: vp[0] != vp[1]).count() / float(parsedData.count())

# Print some stuff
print("Training Error = " + str(trainErr))
```

1. np array is an unsupported data type, changed to LabeledPoint.
2. According to the change of data type, getters also change.

```
A520M-4750G@spark-m:~/spark-example$ python spark-ex.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
20/11/22 14:05:54 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
20/11/22 14:05:54 WARN com.github.fommil.netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
Training Error = 0.04446064139941691
A520M-4750G@spark-m:~/spark-example$
```

The result is different to the original example, maybe there is some update in the library, because all the versions of Spark environment are different.

(2) Write your own SGD (stochastic gradient descent or simple gradient descent) function of logistic regression. And compare the results.

The result is different from (1), I am not sure if it is caused by overflow or different batch size.

```
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:41: RuntimeWarning: overflow encountered in exp
<string>:52: RuntimeWarning: overflow encountered in exp
Training Error = 0.13702623906705538
A520M-4750G@spark-m:~/spark-example$
```

And the source code(only the part changed):

```
parsedData = data.map(mapper).cache()

def gradient(row, w):
    Y = row[0]
    X = row[1:]
    return ((1.0 / (1.0 + np.exp(-Y * X.dot(w))) - 1.0) * Y * X.T).sum()

def add(x, y):
    x += y
    return x

w = 2 * np.random.ranf(size=4) -1
for i in range(100):
    #print("Iteration: {}".format(i+1))
    w -= parsedData.map(lambda r: gradient(r,w)).reduce(add)

labelsAndPreds = parsedData.map(lambda point: (int(point[0]), 1.0 / (1.0 + np.exp(-1 * w.dot(point[1:])))))

# Train model
#model = LogisticRegressionWithSGD.train(parsedData)

# Predict the first elem will be actual data and the second
# item will be the prediction of the model
#labelsAndPreds = parsedData.map(lambda point: (int(point.label),
#        model.predict(point.features)))

# Evaluating the model on training data
trainErr = labelsAndPreds.filter(lambda vp: vp[0] != vp[1]).count() / float(parsedData.count())

# Print some stuff
print("Training Error = " + str(trainErr))
```

reference:

[logistic-regression-in-apache-spark]

[linear_regression_with_sgd_example.py]

[logistic_regression.py]

github: https://github.com/a2134666/CloudComputingHW4