Data Intensive Computing – Handout 5

Training Cluster

You can login to the training cluster *dlrc* via machine `ufallab.ms.mff.cuni.cz` and port 11422, i.e., using `ssh -p 11422 ufallab.ms.mff.cuni.cz`.

All machines in the cluster use same architecture and share `/dlrc_share` directory.

The cluster consists of master `dlrc` and 5 nodes `dlrc-node1` to `dlrc-node5`. Each node has 2 cores and 4GB ram, but four SGE jobs are allowed to run simultaneously (only because otherwise there would be too little slots; the slot number may be even increased later).

Running Browser with Proxy to the Cluster

To run a browser, which can access master *and* the cluster nodes, run:

```
(chromium --proxy-server=socks://localhost:2020& \
ssh -ND 2020 -p 11422 ufallab.ms.mff.cuni.cz)
```

Wikipedia Links Data

The links between Wikipedia pages are available in the following directories:

- `/dlrc_share/data/wiki-links/cs-txt`: Czech Wikipedia pages links, 9.6M links, 500k pages, 328MB.
- `/dlrc_share/data/wiki-links/en-txt`: English Wikipedia pages links, 143.8M links, 11.8M pages, 5.6GB.

The files are encoded in UTF-8 and every line contain two space separated page names – the source page and the target page.

English Tagging Data

The well-known WSJ English tagging dataset is available here:

- `/dlrc_share/data/tagging/en/train.json`: The training portion, 912k words.
- `/dlrc_share/data/tagging/en/dev.json`: The development portion, 131k words.
- `/dlrc_share/data/tagging/en/test.json`: The training portion, 129k words.

The files are in the JSON format (readable by sqlContext.read.json), with the following fields per row:

- `word`: current word
- `tag`: current tag
- `left`: array of the left neighbouring word and the second left neighbouring word
- `right`: array of the right neighbouring word and the second right neighbouring word

Tasks

Task	*Points*	*Description*
`link_path`	4	For two given Wikipedia pages, find out the shortest path of page links between them, if it exists.
`transitive_closure`	4	Compute transitive closure of the Wikipedia link graph. In other words, compute for each page all pages reachable from it.
`page_rank`	5	Compute PageRank of Wikipedia pages and output the pages sorted by the rank. Use given number of iterations of the iterative algorithm `http://en.wikipedia.org/wiki/PageRank#Iterative` with damping factor 0.85.
`english_tagging`	6	Implement a machine learning pipeline which trains to perform POS tagging using `train.json` file and computes accuracy on `test.json` dataset. Try using a `MultilayerPerceptronClassifier`, with either none or one hidden layer. Note that the current state-of-the-art accuracy on the test set is about 97.7.

The pyspark.ml API

ml_simple_text_classification.py

```
from pyspark import SparkContext
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import SQLContext

if __name__ == "__main__":
    sc = SparkContext(appName="SimpleTextClassificationPipeline")
    sqlContext = SQLContext(sc)

    # Prepare training documents, which are labeled.
    # Prepare training documents, which are labeled.
    training = spark.createDataFrame([
        (0, "a b c d e spark", 1.0),
        (1, "b d", 0.0),
        (2, "spark f g h", 1.0),
        (3, "hadoop mapreduce", 0.0)
    ], ["id", "text", "label"])

    # Configure an ML pipeline, which consists of tokenizer, hashingTF, and lr.
    tokenizer = Tokenizer(inputCol="text", outputCol="words")
    hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
    lr = LogisticRegression(maxIter=10, regParam=0.001)
    pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

    # Fit the pipeline to training documents.
    model = pipeline.fit(training)

    # Prepare test documents, which are unlabeled.
    test = spark.createDataFrame([
        (4, "spark i j k"),
        (5, "l m n"),
        (6, "spark hadoop spark"),
        (7, "apache hadoop")
    ], ["id", "text"])

    # Make predictions on test documents and print columns of interest.
    prediction = model.transform(test)
    selected = prediction.select("id", "text", "prediction")
    for row in selected.collect():
        print(row)

    sc.stop()
```

The example is available in `/home/straka/examples`.

ml_string_indexer.py

```
df = spark.createDataFrame(
    [(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],
    ["id", "category"])
indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)
indexed.show()
```

The example is available in `/home/straka/examples`.

ml_multilayer_perceptron.py

```
data = spark.read.format("libsvm")\
    .load("data/mllib/sample_multiclass_classification_data.txt")
# Split the data into train and test
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]
# specify layers for the neural network:
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [4, 5, 4, 3]
# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers,
                                         blockSize=128, seed=1234)
# train the model
model = trainer.fit(train)
# compute precision on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="precision")
print("Precision:" + str(evaluator.evaluate(predictionAndLabels)))
```

The example is available in `/home/straka/examples`.

ml_crossvalidation.py

```
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

paramGrid = ParamGridBuilder() \
    .addGrid(hashingTF.numFeatures, [10, 100, 1000]) \
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=5)

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(training)
```

The example is available in `/home/straka/examples`.