Advanced Programming Miniproject 2
Natural Language Processing
Spring 2018

Thor Olesen (tvao@itu.dk)
Dennis Nguyen (dttn@itu.dk)
Daniel Hansen (daro@itu.dk)

May 9, 2018

Contents

1 Data Sets

We have used the following files to train our artificial neural network[1]:

- Musical Instruments - Size: 10,261 reviews (200 seconds)
- Video Games - Size: 231,780 reviews (2000 seconds)
- Kindle Store - Size: 982,619 reviews (Infeasible)
- Electronics - Size: 1,689,188 reviews (Infeasible)
- Books - Size: 8,898,041 reviews (Infeasible)

The GLoVe model with 50 feature dimensions (i.e. 'glove.6B.50d.txt') is used to reduce training time.

Test machine specifications:

- **OS:** macOS - High Sierra
- **CPU:** Intel Core i7 - 2.3 GHz
- **Memory:** 16 GB 1600 Mhz DDR3

The machine was able to run Musical Instruments and Video Games with an average runtime duration of 250s̃econds . However, when running the solution with Kindle Store, Electronic and Books , which are on the larger end of the given files, the average running time is... Books never completed due to the long computation time. The likely reason for the slow computation is because the solution was running on a single laptop. Thus to increase the computation speed, the optimal solutions is to run it on a cluster of machines or on the cloud.

Also, another factor for the slow computation is due to the preprocessing that ensures the data is compatible with the provided GloVE model. The preprocessing utilizes expensive functions such as: exploding/flatten the whole text into words, joining the words with the GLoVE vectors and finally collapse the vectors into a single average word embedded vector. One may optimize these operations, yet this was not the main focal point.

2 Accuracy

We have achieved an accuracy around 90% on most tested data sets using a layer configuration with an input layer of 50 input features, two hidden layers of size 5 and 4, and an output layer with three class labels (i.e. Negative 0, Neutral 1, Positive 2). We tried to use a glove with more than 50 dimensions assuming that this would increase the accuracy of our review text classification at the cost of increased training time. However, changing the glove did not increase accuracy but only the training time as expected. Also, we tried to change the size of hidden layers (4-40), max iterations (15-100), and convergence tolerance (1E-4 to 1E-6) of iterations without improving results.

- Musical Instrument Reviews
  - Accuracy: 0.8831041257367387
  - Recall: 0.8831041257367387
  - Precision: 0.7798728968932496
  - F1: 0.8282844121411874
  - False Negative: 45, False Neutral: 74, False Positive: 0, Total: 1018
- Video Game Reviews
  - Accuracy: 0.7578759054018676
  - Recall: 0.7578759054018676
  - Precision: 0.5743758879887004
  - F1: 0.653488549702139
  - False Negative: 2802, False Neutral: 2747, False Positive: 0, Total: 22918

[1]All files are found at: http://jmcauley.ucsd.edu/data/amazon/links.html

Cross Validation Results

We used cross validation, which yielded the following results:

- Variance: 0.045595123767712686
- minAccuracy: 0.861003861003861
- maxAccuracy: 0.9065989847715736

3 Parallelization

In terms of parallelization, we use the Spark dataset that provides a type safe abstraction working on top of the RDD (i.e. resilient distributed dataset) abstraction in Spark. This is an immutable distributed collection that is stored in memory, is lazy evaluated, and may be processed in parallel. The level of parallelism is configured in the Spark configuration to be local[*], meaning it runs locally with N threads being the number of cores on the test machine (4 in our case). We compared the degree of parallelization by running the smallest test instance (i.e. Amazon music reviews) on a single core vs 4 cores, which resultated in a lower wall clock time but increased CPU time. In terms of performance, the biggest challlenge on bigger files (i.e. Amazon book reviews) is caused by the fact that Apache spark runs in-memory but has to do hot swapping between memory and disk due to a memory limit on the test machine being set to 12GB ram. Ideally, one could dramatically improve the training time by using Spark with YARN on a cloud cluster (e.g. Hortonworks) using more CPU cores and memory.

4 Extensions

We have tried to use linear regression and naive bayes classification on the data set using an ML pipeline that tokenizes data, converts tokenized words into feature vectors, and scales the vectors before feeding them to the ML algorithm. Surprisingly, using the Spark ML library without using the pre-trained Glove word-embedded vectors runs significantly faster. Presumably, this is because it takes a long time to preprocess the review data into word embedded vectors in the NLP pipeline that tokenizes text into words (i.e. flatten), embed words into vectors (i.e. word embedding), and finally reduces (i.e. collapeses) word embedded vectors into average review vectors.

The project problem is a supervised text classification problem so we tried to use linear regression to identify which review class a review belongs to. The linear classifier achieves this by making a classification decision based on the value of a linear combination of the feature vectors. For this purpose, we used the Spark ML VectorAssembler to convert vectors from words. Based on the results, the Spark lbirary implementation was way faster than using the pretrained Glove model that required expensive flatten, join and reduce transformations. Presumably, Spark is also able to optimize the pipeline shown below:

Source Code 1: Spark ML Pipeline

```
def main(args: Array[String]) = {
val logisticPipeline = new Pipeline().setStages(Array(
        regexTokenizer, // Convert text into words
    remover, // Remove special symbols
    hashingTFUnigram, // Convert words into vectors (word embedding)
    idf, // Scale vectors
    logisticRegression // Perform logistic regression
))
}
```