Sentiment Analysis with Spark

Detecting emotion in language is one of most central and accessible tasks in Natural Language Processing (NLP). The task is important for e-commerce and organisations in order to assess public and consumer opinion and assess strategy accordingly. In this project you will use Amazon's product reviews to construct a simple sentiment analyser.

We will follow a simple method, in three stages:

1. Translate the review text into a vector of numbers (a so called *word embedding*)
2. Use the embeddings with known ratings to train a network (a *multilayer perceptron classifier*)
3. Use the perceptron to predict ratings for another set of reviews (validation).

1 Computing a word embedding model

A *word embedding* is an NLP model that attempts to capture the meaning of a word as a vector of real numbers. Word embeddings are learned from large amounts of raw text data, such as a Wikipedia dump or corpora scraped from the web. They are meant to represent distributional information of the word in question, a representation of the word given its context.

We will use a precomputed set of vectors for 400 thousand English words made available by the GLoVe project at Stanford. Get it at: https://nlp.stanford.edu/projects/glove/ (pre-trained vectors).

Each line in a GLoVe file consists of a word followed by vector of numbers. There are several files with vectors of various sizes (50, 100, 200 and 300 numbers). Longer vectors can, in principle, capture more information about the word, so they should give better precision of analysis. However, it is not obviously clear, that this benefit is perceptible for our problem at hand, so feel free to experiment with various files. Shorter embeddings will be faster to process (and it will be faster to train a neural network using them).

We will perform the sentiment analysis for dumps of product reviews from Amazon.com. You can get the data files here: http://jmcauley.ucsd.edu/data/amazon/links.html. Use the "small" subsets for the experimentation, the files in the 5-core column. The reviews are stored in JSON format. Check the first record to get an idea. We recommend to work with one of the smaller files in the early stages. Once the project is working, try to push the limits, and run on the larger files.

We are interested in three fields from the JSON records: summary, reviewText, and overall. A 'summary' is the title given to the product review. 'Overall' is the rating given by a user. We will be analyzing the summary and the text, trying to predict the overall rating. The ratings are on a 5 step scale (from 1 to 5).

A simple way to capture meaning of a review is to take the average vector of its words. We first tokenize the texts (turn them into words). Then we translate all the words in a reviewText and the summary to vectors from the GLoVe dictionary. We sum all vectors for given review and divide the result by the number of vectors summed (compute the average vector). Ignore the words that are not found in the GLoVe corpus.

You do not need to know more about word embeddings than described above (and in the lecture) to complete the task.

2 Training a multilayer perceptron classifier

A *perceptron* is a simple layered neural network. For our purposes it is sufficient to know that a perceptron is a *classifier*: a 'circuit' that takes as the input features of the classified object, and computes a class to which this object belongs.

In our case the features are the average word embeddings for the reviews, and the classes are the

’overall’ user ratings (the sentiment value). We will only consider three classes: negative for overall rating of 1 or 2, neutral for rating 3, and positive for rating 4 or 5. You will need to remap the ’overall’ rating column in the data from five classes to three classes before training.

We will train a perceptron with 50 inputs/features (if using a 50-dimensioned GLoVe file) and with three outputs (0/1/2, standing for negative/neutral/positive). The training will be performed by an existing algorithm from a machine learning library.

3 10-fold cross validation

Split the review file into 10 parts randomly. For every 9 parts of these, we perform the training of the perceptron. Then we use the last part for the validation. In the validation stage, we compute the text representation of average embedding vectors for each review, and ask the perceptron to predict the sentiment (this again is done using an existing algorithm taken from a library). Then we compare the predicted sentiment class, with the class originally stored in the Amazon files.

Background: https://en.wikipedia.org/wiki/Cross-validation_(statistics) (scroll to k-fold cross-validation)

4 Implementation requirements

1. Use scala, Apache Spark, and its machine learning library MLlib.
2. While Spark provides a cool SQL interface to DataFrames, please avoid using it. We want to train the functional style interface. So don’t use the `sql` method.
3. Use map, flatMap or a for-comprehension.
4. Implement the entire program in a pure way (except for the spark API calls that are impure). Make your functions referentially transparent.
5. Write at least five tests (not necessary property tests, unit tests are fine)

5 Implementation remarks

1. An SBT project with skeleton files is provided in the course git repository. If you run spark from this project (using sbt run), you do not need to install spark on your machine. SBT takes care of that. You need to be online, at least the first time you compile. (Andrzej tested this only on Ubuntu Linux. Rumours are that Windows machines pose problems. This might be the week to use our docker container.)
2. The provided code already shows how to load the JSON file with reviews and the space separated file with word embeddings. We are not using RDDs, but the newer DataSet interface of Spark, which is supposedly more performant on data that is structured uniformly in tables.
3. You will need to tokenize the review text. Use the MLlib tokenizer for that.[1]
4. [**important**] A computation on one Dataset cannot use a computation involving another one as a parameter. *You cannot access other Datasets in the closure of a function value passed to a HOF operating on a Dataset.* This means that we cannot use map to translate reviews to embedding vectors. This would require accessing the GLoVe Dataset in the closure of the map’s argument. You get an inexplicable exception when you attempt that.
   We need to bring the data and the computation to a format, which Spark can distribute. One workaround is to flatten the records containing lists by creating a separate entry for each word (as is typically done when normalizing data to 1st normal form in databases). Once the data is in this format, you can `join` the Dataset of GLoVe with the reviews over the word column (as you would do it, if you implemented this in a relational database).
5. Use the perceptron implementation from MLLib.[2] Study the example in the referred page of

[1] https://spark.apache.org/docs/2.4.4/ml-features.html#tokenizer
[2] https://spark.apache.org/docs/2.4.4/ml-classification-regression.html#multilayer-perceptron-classifier

documentation (it also includes validation with prediction and accuracy computation).
The API requires that all features are Double values in an instance of `org.apache. spark. mllib.linalg.Vector`. Use the `Vectors.dense` factory to convert from arrays to this type. These vectors should be found in a Dataset column named `"features"`. The value we want to predict (0/1/2) should be placed in the column named `"label"`.
The perceptron needs to be configured. Use four layers, like in the documentation example. We need to modify the size of the input (first) and the output (last) layer in the configuration array. The input layer must have the same size the feature vectors (so 50 if you are using the smallest of the GLoVe files). The last layer has a unit for each of the classes in the label's column.

6. You will need some small files for testing (extract or create them).
   Sharing small test files is allowed between groups. If you make them available to teachers, we will share them with the entire class.
7. On an RDD call `_.toDS` to convert to a DataSet. On a DataFrame call `_.as[T]` to convert to a `Dataset[T]`. To convert a Dataset/DataFrame to an RDD access its `rdd` property.
8. It is often recommend to compile and run in separate JVM sessions. So first run 'sbt package' and then 'sbt run'. SBT's heap often gets broken after several spark reboots and you need to restart it anyways.
   Sbt is not really made for large memory programs so you are likely to experience all sorts of weird crashes.
9. You will quickly realize that most of the stuff you are writing is lazy and returns immediately. Use `take` and `count` to force computations on spark data objects for inspection and manual testing in the REPL. Also `show` can be used on the Dataset objects to visualize a prefix of a table.
10. For processing large files, you *might* find it useful to install spark anyways. This is how Andrzej runs this project outside sbt after compiling with '`sbt package`':

```
spark-submit --driver-memory 10G --class "Main" --master "local[8]"
  target/scala-2.11/sentiment_11-1.0.jar
```

    (If someone creates a good docker file with the installation, please let the teachers know, and we can share it with other students.)
11. I found the REPL much more useful than usual. It allows you to explore queries (completion is useful) and test step by step, visualizing parts of the data (with show and take)

6 Supporting resources

The main spark documentation is available online.[3] There is an O'Reilly book on Safari books that has some quick basic documentation, if you have access to Safari. There is also a free online ebook.[4]

7 Some extension ideas

If you find the subject interesting, here are some ideas on how to personalize your project by doing slightly more work. Other extensions are welcomed.

1. Push the size of the files to the limit (the book reviews are the largest collection). This can be done by optimizing the program, spark configuration to allow for heavy memory use, setting up a local cluster, and/or moving the computation to the cloud.
2. Experiment whether a model trained on one kind of products works on other kinds of products.
3. Investigate where do we have a monoid in the project (and why is this desirable for this project to have one)?

[3] https://spark.apache.org/docs/2.4.4/index.html
[4] https://mapr.com/ebooks/spark/

Perhaps not here: for us it is interesting to see several aspects: how laziness allows to split the compution and postpone it

8 Assessment

You hand-in a single Scala file containing your implementation (no .zip files accepted). Indicative size 200 lines.

In the bottom of the file answer the following questions in a comment:

1. What data sets have you managed to run (including size)?
2. What accuracy have you obtained? With how many iterations? And with what layer configuration? Show the variance of accuracy observed in cross validation.
3. What degree of parallelization was obtained? (for instance contrast the wall clock time with the CPU time). Note we are not asking you to optimize parallelization, just to report what you obtained.
4. What extensions (if any) you have implemented? Have you tried another classifier? Another network configuration? Running on a highly parallel cluster of machines, etc. ...

Please try to stay on point and do not write a lot.