The yelp dataset

The yelp dataset is a treasure trove of data for which all kinds of interesting questions can be formulated.

The review dataset is particularly suitable for typical sentiment analysis questions such as:

- Which words or ngrams are most useful in predicting how a company will ultimately be rated? Can you use these words to draw conclusions about what is most important to customers?
- Does the length of a review correlate with whether other users rate it as useful?
- How does a text rated as "funny" differ from a text rated as "cool" or "useful"?

Using parts of the photo dataset, convolutional neural networks could be trained to answer the questions like:

- Is there food or drink in the photo?
- Is it a menu?
- Does the photo show the interior or exterior view of a building?

Regarding the business dataset, SVC, RFC, etc. could be trained to answer questions like:

- What is the best location in LA to open a restaurant?
- What kind of restaurants are most successful in my region?
- Do longer opening hours or a delivery service affect the average rating of my restaurant?

Recommender System

In this challenge I took inspiration from Sparks machine learning library and tried to implement an explicit feedback collaborative filtering recommender that tries to predict how users rate businesses.

The recommender learns on the basis of a user-item-rating matrix, which contains data about which users gave which businesses what ratings in the past. This information is read from the review dataset.

The number of latent factors and the strength of regularization of the ALS model were optimized by cross validation. Due to limited computing capacities, only a few parameter combinations with 50 training iterations were tested within a 1-fold cross validation. For the hyperparameter "rank" the values [20, 30, 50] were tested in combination with each of the values [0.1, 0.3, 1] for "regParam".[1] Training was performed on 60% of the review data set, validation and out-of-sample testing took place on 20% of it each. The best performing model used 30 latent factors, a value of 0.3 for "regParam" and resulted in an RMSE of 1.476 on the test set.

recommend_for_user.py is a small program that can be started and configured via the console. It uses the best recommender model to make business recommendations to a new user based on the name of his favorite business. For this purpose, the user is appended to the user-item-rating matrix, and the model is refitted to the updated matrix. The fitted recommender is then used to predict for the given user for all businesses (except his favorite business) how much he would like them. The businesses with the highest ratings are recommended to the user.

Using the script's options, the user can specify the name of his favorite business and determine how many business recommendations he wants to receive. Furthermore, the directory in which the data is

[1] When tune_recommender is set to True in *recommend_for_user.py,* the script tunes the model using hardcoded parameter combinations.

stored can be specified, as well as the names of the business and review json-file of the yelp dataset. All arguments to the script are optional. If the script is run without arguments, it assumes that the user's favorite restaurant is named "Sugar Bowl". If the specified name is not listed in the business file, the program will throw a KeyError.

```
$ python recommend_for_user.py --help
```

```
optional arguments:
  -h, --help            show this help message and exit
  --fav_business [FAV_BUSINESS]
                        name of favorite business (5 star rating will be
                        assumed)
  --dir [DIR]           directory that contains input data
  --business_filename [BUSINESS_FILENAME]
                        name of the file that contains information about
                        businesses
  --ratings_filename [RATINGS_FILENAME]
                        name of the file that contains each users business
                        ratings
  --checkpoint_directory [CHECKPOINT_DIRECTORY]
                        checkpoint directory used by the ALS model
  --n_rec N_REC         number of business recommendations that are outputted
```

Note: The class defined in *create_debug_data.py*, was used to create smaller versions of the review and business data set that were used for debugging.

Known Issues

The script throws an EOFError, but keeps running. The error message ends like this:

> File"D:\Programme\ApacheSpark\spark-2.4.5-binhadoop2.7\python\lib\pyspark.zip\pyspark\serializers.py", line 724, in read_int EOFError

The error is somehow connected to *the _add_numeric_ids* method of the recommender.

How to run the code on Docker (at least in theory)

1. Clone the git repository https://github.com/RalfSchmidtner/new_yorker_challenge to your machine.
2. Download the dataset from https://www.yelp.com/dataset to *new_yorker_challenge/data* and extract it there. The extracted json-files should end up in the directory `new_yorker_challenge/data/yelp_dataset .`
3. Build the the image from the dockerfile, which is located in the root directory of the repository.

   ```
   $ docker build --tag ralf_schmidtner:1.0 .
   ```

4. Run a container on the image.

   ```
   docker run -d --name container_schmidtner ralf_schmidtner:1.0
   ```

5. Enter a interactive bash console and change to sparks *bin* directory.

   ```
   $ docker exec -it container_schmidtner /bin/bash
   bash-5.0# cd ./spark/conf
   ```

6. Run the following command to get ten recommendations for a user that loves the restaurant Sugar Bowl.

   ```
   bash-5.0# spark-submit --master local[*] \
   /app/recommender/recommend_for_user.py \
   --fav_business "Sugar Bowl" \
   --n_rec 10 \
   --dir /app/data/yelp_dataset \
   ```

```
--ratings_filename yelp_academic_dataset_review.json \
--business_filename yelp_academic_dataset_business.json \
--checkpoint_directory /app/data/checkpoints
```

Sentiment Analysis

I also played around a bit with pipelining & sentiment analysis tools provided by the Spark ML library. In the *script review_sentiment_analysis.py* within the *sentiment_analysis_playground* directory an SVC is trained to distinguish between good (4 or more stars) and bad reviews based on the review texts.

In a first pipeline firstly the review texts are tokenized. The second stage uses the tokens to create bigrams, the third and fourth stages create a tf-idf matrix based on that bigrams. Then an SVC is trained to differentiate between positive and negative reviews using this matrix.

The bigrams that were considered important by the SVC (high coefficients) are then integrated into the review texts. The review texts, which now contain the key bigrams, are subsequently fed into a second pipeline. This pipeline tokenizes the updated texts in its first stage. In the second stage stop words are removed. In the third and fourth stage, a tf-idf matrix is created based on the remaining tokens. Based on this matrix, a second SVC is trained to classify the review texts.

With this approach, a solid out-of-sample ROC AUC score (>0.9) could be reached, despite the fact that only 8K review texts were used for training.

Final notes

Many thanks for this challenge. I know there are still improvements to be made on all fronts, but that's the status quo after one week. I learned a lot about Spark and Docker and even though it was very challenging overall, I enjoyed it.