Data Mining: Learning from Large Data Sets - Fall Semester 2015

mwurm@student.ethz.ch
merkim@student.ethz.ch
lwoodtli@student.ethz.ch

October 13, 2015

Approximate near-duplicate search using Locality Sensitive Hashing

Problem Description

The goal of the project was to develop a program that finds duplicate videos. The videos were provided as shingles in a text file. There was also a training set for the video data. The program needed to run on a Map Reduce environment and should be developed with a Locality Sensitive Hashing (LSH) algorithm.

Approach of the Team

The team decided that each member will develop the algorithm independently in a first step. Afterwards the implementations were compared. So details could be discussed and errors were eliminated. At the end one implementation was chosen and the final code was developed together.

Environment

The running environment with a Map Reduce framework was provided. So the Mapper and Reducer were developed independently and then submitted to the running environment.

Map

The Mapper implements the LSH algorithm. In a first step the provided shingles of each video are min-hashed.

```
# each line contains data for a video
for line in sys.stdin:
        # ...
        # iterate over the shingles of a video
        for shingle in shingles:
                # iterate over the hash functions
```

```
for i,hash in enumerate(minHashes):
        hashed = (hash[0]*shingle + hash[1]) % 20000
        # keep hashed value only if it is smaller
                if hashed < signature[i]:
                signature[i] = hashed
```

As a second step the min-hashes are divided in bands by a LSH algorithm.

The resulting min-hashes per band and video are sent to the Reducer together with the original shingles and sorted by band-id and min-hash value. The original shingles were provided in order to allow the Reducer a real comparison of the potential duplicates.

Reduce

The Reducer iterates over the sorted input and checks for duplicate min-hash values which indicates potential duplicates. To decrease the amount of false positives the Reducer compares in an additional step all shingles of the potential pairs for 90% equality. This real comparison of the videos allows an improvement of the precision. But it also introduces more work to the Reducer. Therefore it is very important to have an optimal Mapper step which makes sure that no duplicates are missed (leads to false negatives) and at the same time does not provide too many false positives which need to be sorted out by the Reducer (decrease of efficiency). This shows us that it is really important to choose the right values for the two main parameters of the Map step.

One of them is the number of hash function to use for calculating the Min-hashes. The other one is the number of rows in a band for the LSH algorithm.

Improvement

Missed real duplicates (false negatives: fn) and wrong potential duplicates (false positives: fp) can weaken the result, which should at best, contain as many correct duplicates (true positives: tp) and deny any non-duplicates (false negatives: fn) as good as possible. The main keywords in this context are Precision $= \frac{tp}{tp+fp}$ and Recall $= \frac{tp}{tp+fn}$. These two quality charactersitics shall be as close to 1 as possible. To improve the result the two main parameters described above can be tuned. The calculated probability for a hit, gives an first hint, where to search for the best configuration. The probability for a hit of two signatures within any of the bands is given as:

$$P(\text{Colission in some band}) = 1 - (1 - s^r)^b$$

Since we are after a colission rate of arround 99%, we decided to use 10 bands with 10 rows each. Given a similarity of 0.9, this leads to a probability of 98.6% for a de facto collision between similar files.

To reduce the risk of outputting a video pair more than once there could be a final step that compares the output of all Reducers and eliminates video pairs that occur more than once. This comparison could be performed in an additional Reduce step. This final step was not implemented in this project.

Conclusion

This project shows that LSH is a nice approach to find near duplicates in a large set of videos. It can be simply developed in a Map Reduce environment.