Data Mining: Learning from Large Data Sets - Spring Semester 2014 Projects

nivo@student.ethz.ch
ganzm@student.ethz.ch
rohrp@student.ethz.ch

April 7, 2014

1 Approximate near-duplicate search using Locality Sensitive Hashing

This section explains the implementation of project 1. In this first project we were supposed to find near-duplicate videos using Locality Sensitive Hashing. The videos were given as a number of shingles that could be compared. The implementation was done in Python to be used in combination with the hadoop infrastructure. This means we implemented a mapper and a reducer that read from stdin and write to stdout. In the following two sections we explain the functionality of the mapper and the reducer each.

mapper.py

The mapper creates a signature vector of a video. This signature vector is afterwards subdivided into b bands having r elements that get hashed. Each band and its hash are then emitted.

These are the steps in more detail:

1. As Figure 1 shows, choosing $b = 32$ and $r = 8$ does not miss any true positives.
2. We create $k = b \cdot r$ hashfunctions of the form $a_p x + b_p \mod c_p$ where $a_p = rand(0, 999), b_p = rand(0, 99999)$, $c_p = 10000$ and $rand(x, y)$ is random variable drawn from a uniform distribution

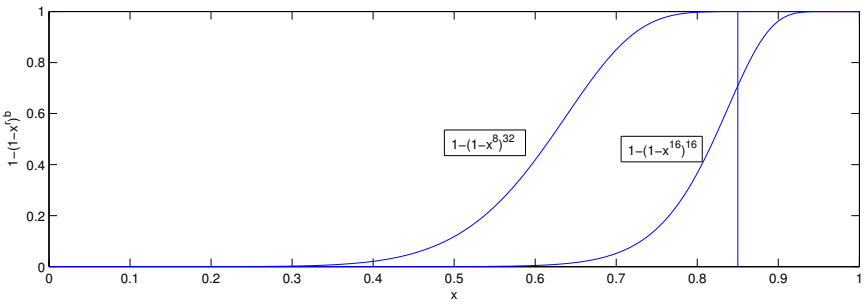


Figure 1: $1 - (1 - x^r)^b$

between x and y. These hashfunctions are used to calculate the permutations used for min-hashing.

3. We create r hashfunctions of the form $a_b x + b_b \mod c_b$ where $a_b = rand(0, 999), b_b = rand(0, 999)$, $c_b = 1000$ and $rand(x, y)$ is random variable drawn from a uniform distribution between x and y. These hashfunctions are used to calculate the band hashes before emitting.
4. The signature is then created according to 1

Algorithm 1 Create signature

$signature = \infty$
for all shingle in shingles **do**
  for i in range(k) **do**
    $signature[i] = min(a_{p_i} \cdot shingle + b_{p_i} \mod c_p, signature[i])$
  end for
end for

5. Calculate the band hash and emit the band hash and the band as key, and the video with its corresponding shingles as value according to 2

Algorithm 2 Emit keys and values

for band in range(b) **do**
  $vector = signature[band \cdot r : band \cdot r + r]$
  $bandhash = \sum_{i=1}^{len(vector)} a_{bi} \cdot vector[i] + b_{bi} \mod c$
  emit $key = [bandhash, band]$ $value = [video\_id, shingles]$
end for

reducer.py

The main task of the reducer is to get rid of the false positives by comparing the reported similar videos using the jaccard distance:

1. gather all videos with the same key in a collection $duplicates$
2. emit similar videos like shown in 3

2 Large-Scale Image Classification

This section describes the implementation of project 2. The task was to implement binary classification on a large number of images.

2.1 mapper.py: Stochastic Gradient Descent

For the mapper functionality we chose to implement the PEGASOS [1] algorithm discussed in the lecture. The trained normal vector w is emitted out of each mapper.

Algorithm 3 Emit similar videos

```
for i=0 to len(duplicates) do
  for j=i+1 to len(duplicates) do
    if duplicates[i].video_id < duplicates[j].video_id then
      shingles_left = duplicates[i].shingles
      shingles_right = duplicates[j].shingles
      distance = |shingles_left ∩ shingles_right| / |shingles_left ∪ shingles_right|
      if distance > 0.85 then
        emit duplicates[i].video_id duplicates[j].video_id
      end if
    end if
  end for
end for
```

2.2 Feature transformation

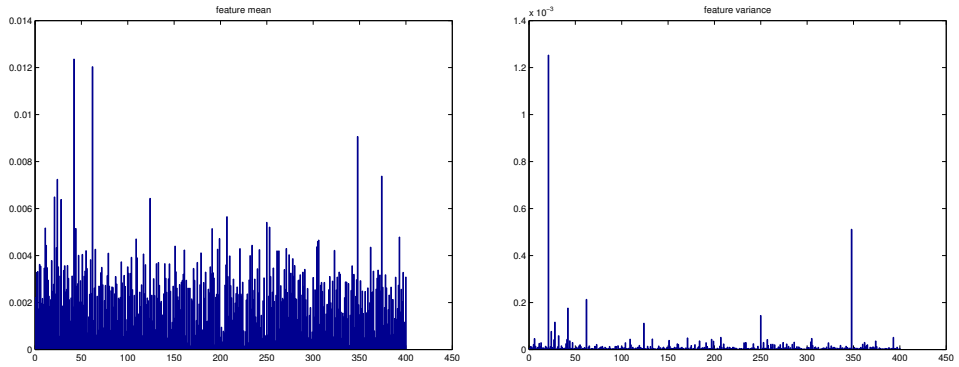
To reduce the classification error, meaning having less falsely classified images we implemented following transformations.

2.2.1 Normalisation

As a feature transformation we chose to normalize our data first. Based on the training data both mean and variance was calculated. These two calculated vectors were hard coded into the mapper function to apply the following transformation to the data:

$$x' = \phi(x) = \frac{x - \mu}{\sigma}$$

This transformation indeed seems to make sense when looking at figure 2. Both mean and variance are somewhat scattered.



(a) Feature Mean (b) Feature Variance

Figure 2: Mean and Variance of Feature Vectors

2.2.2 Additional features created

In addition to the already existing set of 400 features we added the set of 400 square roots and the set of 400 logarithms to the transformation. Upon each of the sets mentioned before following features were added:

- mean
- variance
- standard deviation
- number of zero elements
- minimum element
- maximum element
- median element

2.3 reducer.py

Inside the reducer all trained vectors w are gathered and averaged. The averaged vector is then emitted.

2.4 Cross validation

To verify the success of the feature transformation and to find good λs for the PEGASOS algorithm we implemented cross validation on the training data: We trained the vector w against a subset of the training set and calculated the error with the remaining data.

References

[1] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.