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

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

May 31, 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

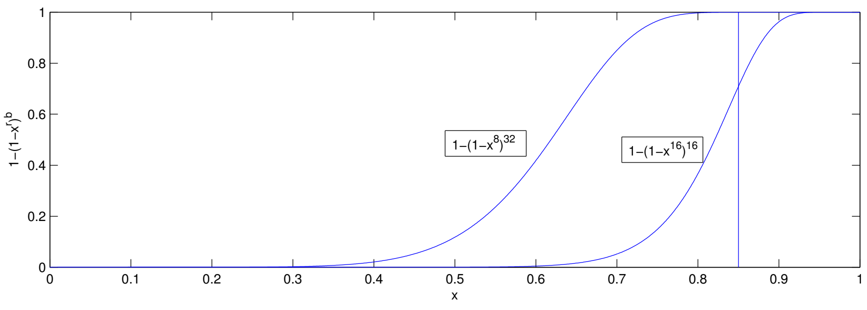


Figure 1: $1 - (1 - x^r)^b$

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 [2] 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
      $shingles\_left = duplicates[i].shingles$
      $shingles\_right = duplicates[j].shingles$
      $distance = \frac{|shingles\_left \cap shingles\_right|}{|shingles\_left \cup shingles\_right|}$
      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

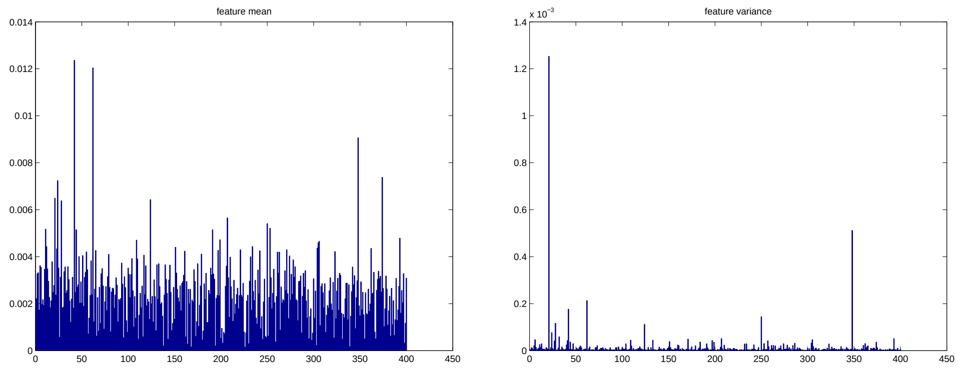
To reduce the classification error, meaning having less falsely classified images we implemented following transformations.

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. To each of these sets the following features were added:

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

This results in a new feature vector of about 1220 elements after transformation.

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.

3 Extracting Representative Elements From Large Datasets

This section describes the implementation of project 3. The goal of this task was to extract representative elements from a large image dataset.

Generally speaking we implemented online K-Means for both the mapper and the reducer. In doing so we nearly reached the hard baseline (score 741.96).

3.1 Mapper

The 300 mappers which independently run over a their subset of data perform online K-Means.

We chose to run K-Means with k=200 centroids. A minor modification of the algorithm is that we track the number of times we touch (move) a centroid while performing the algorithm. We refer to this as the "touch count" of a centroid. Each mapper in the end emits all of its 200 centroids with their corresponding touch counts tc_i.

3.1.1 Centroid Initialization

What we've learnt from this project is that initialisation really matters (in terms of score). We optimised initialization of the centroids in a way that the touch count is evenly spread. If a mappers K-Means algorithm trains a centroid which is never touched this specific one is most probably badly initialized.

By playing around with a few parameters we came up with a **normally distributed** set of **initial centroids** with $\mu = 0$ and $\sigma = \frac{1}{100}$.

3.1.2 Learning Rate η

The next modification of the default online K-Means algorithm presented in the lecture is the learning rate. For centroid i we keep track of a learning parameter $\eta_i = min\left(0.05, \frac{1}{tc_i}\right)$. This way we decrease the parameters value down from 0.05 which worked best for us.

3.1.3 Emitted centroids with corresponding touch counts

Figures 3 shows a histogram with touch counts and number of centroids after the mapping step of 10 mappers (2000 centroids). If there is a bar at x-axis 50 with height 320 this means that there are 320 centroids which each have between 49 and 52 closest points assigned. We can see from that plot that there are no unused centroids. The minimum number points assigned to a centroid is roughly 20. There are some outliers with much higher counter that are cut out of the set.

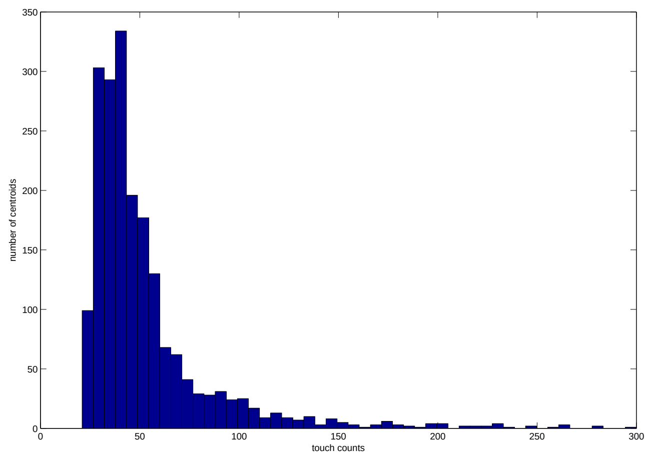


Figure 3: Distribution of point assignments of mapper output

3.2 Reducer

The reducer receives for each of the 300 mappers 200 centroids-touch count pairs. We perform online K-Means on the centroids which are now viewed as weighted data in the context of the reducer.

The touch counts are considered as follows: When K-Means iterates over one point with touch count tc_i it applies the centroid update step tc_i times.

4 Explore-Exploit Tradeoffs in Recommender Systems

For project 4 we implemented different algorithms. Namely the two algorithms LinUCB and Hybrid LinUCB described in paper [1]. The highest score was reached with a simple version of LinUCB which only considers user context data. As soon as we added article data the test score got worse

4.1 Features

Depending on what time of the day a user wants to access different articles. One could imagine that at work a typical user prefers articles about news while in the evening a normal user wants articles about entertainment, night life etc.

We constructed the following feature

$min\,(hh/24, (24 - hh)/24)$

where hh is the current hour of the day. This basically describes a pyramid function. The intuition behind is that similar hours of day should yield similar feature values.

However adding this feature decreased our score. We assume that timestamps are maybe not local timestamps. If log is collected from different time zones this may distort our user feature vector.

4.2 Clustered UCB

As a first implementation we tried clustering the users to user groups using a simple online k-Means algorithm with a $k = 12$. Then we applied the simple plain UCB algorithm. For each user group we memorized the mean reward and the corresponding upper confidence bound. The performance of this algorithm however was quite poor.

4.3 Hybrid LinUCB

Implementing the significantly more complicated hybrid yielded significantly worse score than the plain simple LinUCB algorithm. We could not figure out how to improve our score with the hybrid model. Despite of the paper stating that performance should be quite good.

4.4 LinUCB

The best performance was reached with the LinUCB algorithm from paper [1]. We ignored article information and considered only user context data. With an α value of 0.2 we reached the best performance.

Every attempt to use the article features resulted in worse score than omitting them.

4.5 Local Testing

To locally evaluate our code we downloaded the yahoo dataset from `http://webscope.sandbox.yahoo.com/catalog.php?datatype=r`. We changed the local evaluator.py code to be able to handle the data.

References

[1] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[2] 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.