Data Mining 2013: Project Report

lbeyeler@student.ethz.ch
dmelidis@student.ethz.ch

July 1, 2013

1 Approximate retrieval - Locality Sensitive Hashing

We implemented the straight forward 3-step version of the LSH algorithm for the map-reduce framework. Later on we added a false positive (FP) rejection to the last step. FP rejection was implemented by computing the Jaccard similarity between all the documents hashed to the same bucket. This is not the most efficient way of rejecting the false positives (and confirm the true positives) since for the same document the last step is performed once per band and thus the same document is validated multiple times. Ideally the FP rejection is done in an additional map-reduce step.

1. How was your choice of rows (r) and bands (b) motivated? How did you search for the best parameters?

   Answer: In our initial straight forward implementation without the FP rejection we optimised r and b such that the false positives and false negatives in the theoretical s-curve are balanced as well as minimised. To this end we used both the approximation from the slides and a numerically exact computation of the threshold. Figure 1 illustrates this parameter search. After adding the FP rejection we optimised our parameters for high true positive rate by minimising the false negatives over the similarity threshold from the assignment while still keeping a steep s-curve to reduce the number of false positives. Figure 2 shows the combinations we tested for this second approach.

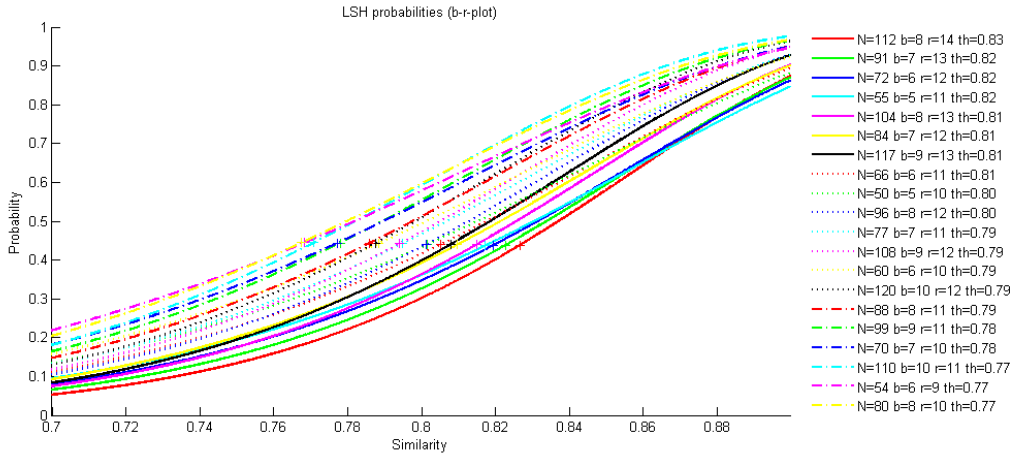

Figure 1: The s-curves of some interesting r and b combinations we found and tested using our initial approach without the FP rejection.

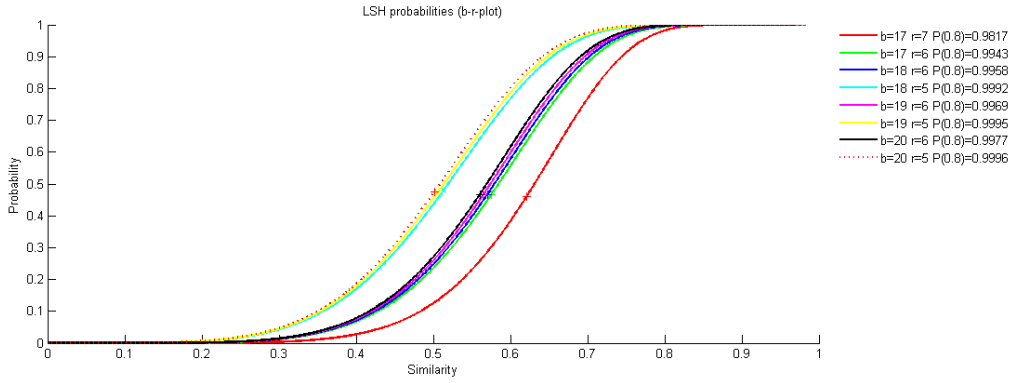

Figure 2: The eight s-curves of r and b combinations we used to test our final approach including the FP rejection of which most got an f1-score of 1.0.

2. Conceptually, what would you have to change if you were asked to design an image retrieval system that you can query for similar images given a large image dataset?

   Answer: The basic algorithm only needs adjustment of the hashing to the new dictionary size since we may expect the image features to be in a bag-of-words representation[1]. As such the algorithm works in the same way as with the shingled documents from assignment 1. To return only the similar images to a query image a naive solution is to add another map-reduce step filtering for pairs where one of the two is the query image. A smarter solution is to run the basic algorithm on the database in an offline step registering the images to their buckets. Then the algorithm only needs to hash the query image(s) and grab the respective buckets and the registered images associated with these buckets for the (optional) FP rejection step.

[1] To get a bag-of-words representation of an image in a similar fashion as we got one by shingling our documents there are three steps: extract your features of choice from the image, choose a dictionary of said features from the images in the database and assign the matching set to this image.

2 Large-scale Supervised Learning

1. Which algorithms did you consider? Which one did you choose for the final submission and why?

   Answer: We consider the parallelised stochastic gradient descent [Zinkevich M., et al.] with a few modifications for better performance in the imbalanced data set. The main reason for selecting this algorithm is the straight forward implementation using online stochastic gradient descent. Also its properties to decrease the bias by averaging the results of the individual SVMs and to decrease the average solution error as the problem complexity goes up and the number of used machines is increased.
   To ensure repeatable but still valid test results, our first modification was to eliminate the randomness the shuffling step in the map-reduce framework introduces using hashing for the bucket assignment, sorting and then re-shuffling (with our own controllable *java.lang.Random* instance). This enabled us to use grid-search more efficiently (one trial per parameter configuration) instead of running each test multiple times.
   We tried different approaches related to the learning rate η:

   Diminishing Learning Rate η is scaled by a factor inversely proportional to the number of samples processed t to avoid over fitting on the last few data points. We tried modelling the factor as $\frac{1}{\sqrt{t}}$ and as $\frac{1}{ln(t)}$ both with moderate success.

   Windowed Approach This approach is described by [Kuncheva L., et al.] as *variable learning rate*. The idea is to get a sense of the change in the error rate over a number of past events and modify η according to these changes. The results were below our expectations, worse than some of our tuned fixed learning rates.

   Fixed for Both Classes Our final approach achieving our top score was to use two separate learning rates η_p and η_n. Our parameter optimisation then was based purely on the profound class imbalance in our data set and the different misclassification penalty for each class.

2. How did you select the parameters for your model? Which are the most important parameters of your model?

   Answer: The parameters of our model are the learning rate (η), the regularisation parameter (λ) (indicating the difficulty of the classification problem) and the number of machines (κ) used for parallelisation. From theory the parallelised stochastic gradient descent has $\mathbf{E}[error] \leq O(\varepsilon(\frac{1}{\sqrt{\kappa\lambda}} + 1))$ so we tuned κ to the maximum number of available processors[2] and λ to a small positive value as to minimise the upper bound for the expected classification error. From previous discussion we ended up using individually fixed learning rates for the two classes: we choose to weigh the negative class more than the positive because the penalty of false negative is 4 times higher than the penalty of false positive and the data set contains 15 positive instances every 100 instances.
   For the parameter tuning the learning rate(s) impacted the score the most. By intuition η describes how fast we descend in the gradient direction to find the global minimum (of the hinge loss function). There is a trade-off which comes in via the reprojection which pulls the step in gradient direction back onto the feasible solution space correcting more as the step size increases. After an appropriate tuning of λ and κ which results in a base classification, we used a grid-search to select the learning rate(s) appropriately.

[2] For a fixed η we performed a grid search on λ and κ for which we achieved better scores for small λ and large κ.

3 Recommender Systems

1. Which algorithm did you implement? What was your motivation?

   Answer: We implemented and iterated over several algorithms. Here are our short conclusions in the chronological order of implementation. Our bandit model uses an arm per article in all approaches.

   UCB1 The algorithm is designed to learn the mean return and confidence width for independent arms. The algorithm already performed significantly better than choosing uniformly at random even though it neither considers the context (user features) nor the age of the articles and requires an initialisation for each newly encountered article. To initialise an arm we simply counted the number of no-clicks until and including the first click.
   We later extended this approach by collecting feedback in buckets to smooth out the binary reward of click versus no-click improving slightly on the standard algorithm.

   Clustering UCB1 We implemented a clustering variant of UCB1. Using sequential k-means we clustered the contexts into groups keeping a separate UCB1 instance per group. The intuition behind this approach is learning different group preferences for distinctly different users (contexts). This approach requires a huge amount of training data for even small numbers of clusters (groups) and we got the best scores using only $k = 4$. Even so the score was only a marginal improvement to the unclustered UCB1 though we would expect better performance for a larger set of data depending in our case also on the duration an article stays interesting (that is how fast the article ages / its relevance deteriorates). We reran the tests later using the bucket approach in UCB1 to no avail.

   LinUCB disjoint linear model Our implementation is learning a hindsight optimal linear mapping on the user features (contexts) alone. This approach improved significantly over the UCB1 and clustering UCB1 implementations. We extended this approach both by using buckets and including an exponentially decaying age coefficient for the articles.

   LinUCB hybrid linear model This model extends the disjoint one, with an additional linear mapping on a combined user-article space. This implementation required a lazy evaluation and a buffering strategy for re-occurring computations and some numerical optimisations to avoid the time limit on the submission system. Therefore we spent a lot of time on these code optimisations.
   We tried using buckets in a similar fashion as with UCB1 to smooth multiple updates by averaging their combined rewards. This led to no improvements and was not utilised in the final submission. After implementing an exponentially decaying age coefficient for the articles this approach led to our final top score improving slightly on the disjoint linear model.

   The hybrid linear model is the most successful approach for the given data set. The other approaches neglect to account for any shared information between users and the users-article space. Only the Clustering UCB1 approach tries to distinguish the reward for different groups of users but with the cost of splitting the dataset and thus only with moderate success.

2. How did you select the parameters for your model?

   Answer: As described the best approaches are the disjoint and hybrid linear UCB with the addition of an age coefficient for the articles. Consequently we needed to tune the model parameter α and the speed at which the exponentially decaying age coefficient would change (called β). As a starting point in figure 2 of section 5.3 in [Lihong Li, et al.] we examined similar α values. With fixed values for α we found that the speed for the decay of the age coefficient β for this dataset is in the order of seven days (we assume UNIX timestamps).

3. Does the performance measured in CTR increase monotonically during the execution of your algorithm? Why?

   Answer: No. The CTR is fluctuating because of several possible reasons:

   - LinUCB is a hindsight optimal algorithm. While LinUCB learns coefficients from feedback from recent decisions, prediction is based on past experience. For an unusual situation the past experience initially impedes an optimal prediction.
   - The linear UCB model falls short to recommend the best article when the reward is non-linearly related in the feature space.
   - Articles may have different lifespans such that a sensational article might be very interesting on the first day, but already old news after that. All that accumulated positive feedback is now slowly deteriorating while LinUCB still predicts high importance relative to other less sensational articles.
   - The selection of articles may contain several similar items, amplifying noise in the learned coefficients and leading to worse predictions until some of the similar items have left the selection or are getting irrelevant.
   - Certain topics might be overrepresented relative to niche topics. Most users will share interests in those prominent topics while still be interested in some niche topics. Thus the most mainstream topics will get a broad range of strong coefficients while the niche topics are less often suggested, leading to fewer feedback and a slower learning rate.

References

[Zinkevich M., et al.] Martin Zinkevich, Markus Weimer, Alex Smola, Lihong Li (2010) Parallelized Stohastic Gradient Descent Advances in Neural Information Processing Systems 23, 2010

[Kuncheva L., et al.] Ludmila Kuncheva and Cartin Plumpton (2008) Adaptive Learning Rate for Online Linear Discriminant Classifiers SSPR & SPR '08 Proceedings

[Lihong Li, et al.] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire (2010) A contextual-bandit approach to personalized news article recommendation WWW-10, Raleigh, 2010