MiBOOTS: Multi-armed-Bandit Online Two-phased System

An Ensemble Method for Real-Time Image-Feature Based Clothing Recommendations

David Greenfield[1], Erin McMahon[1], Josh Plotkin[1], and Priya Venkat[1]

[1]Columbia University, Computer Science, New York, 10027, USA

***Abstract*— We propose a method, MiBOOTS, for making user-specific clothing recommendations using high-dimensional feature representations extracted from a Convolutional Neural Network. We implement an ensemble of batch and online learning methods (multi-armed bandits, gradient boosted trees, and online logistic regression using stochastic gradient descent) to optimize for speed, scalability, and performance. See the model in action on our website.**

I. Introduction

To date, recommendation systems are divided into two categories: collaborative or content-based filtering. Collaborative filtering is based on sparse user-item matrices. The method is especially useful in identifying "styles" that might not be realized in the images themselves.[8] However, it can muddy the difference between substitutes and complements. Content-based filtering can identify object similarity and addresses the "cold start" problem, i.e. when few users have viewed an item.[9], [11] This approach has been more widely utilized as CNN and deep learning techniques have grown more robust.[10] However, content-based methods are primarily used for substitute selection.

Whether collaborative or content-based filtering, most recommendation systems are trained offline, and most online website only make recommendations visible on product pages.[18] Furthermore, substitute and complementary models have to be trained separately.[12], [13] In both our research and experience, we have encountered few systems that optimize the user experience in real-time for both substitutes and complements.

Therefore, our goal with MiBOOTS was to create a real-time hybrid recommendation system. Our approach combines content metadata by clustering data based on CNN features as well as user data through a multi-armed bandit + classification prediction. Multi-armed bandits prevent bias in our model by introducing entropy into recommendations. In addition, multi-armed bandits addresses both complimentary and substitute selection by allowing for independent β posteriors across clusters. Image-based clusters decrease our search space at each iteration while online classification allows us to operate in real-time.

II. Amazon Data Set

We leveraged a research data set of product listings from Amazon.com review data set that included both product-level data and 142.8 million reviews spanning May 1996 - July 2014. Given the flexible design of the MiBOOTS framework, our system could work within specific product groups as well as a more general set of products.

To limit the scope in our initial model, we focused only on a deduped set of "boots" in the product descriptions of the product category Clothing, Shoes and Jewelry. The resultant data set includes 3,697 women's boot products. For each product a unique identifier (asin) as well as product metadata listed below.

- price - prices as of data extraction date
- imUrl - Url to image of product
- related products - List of asins for also bought, also viewed, bought together, buy after viewing
- salesRank - sales rank information
- brand - brand name
- categories - list of categories the product belongs to

We focused exclusively on image data; although, in this paper we will suggest some areas where additional data could be used to augment or validate the success of the system. The images themselves were used for display within the system. However, a vector of Convolutional Neural Net for each image was used as the feature set. The creation process for this data is described in more detail below.[15]

In addition, we completed user testing based on our application. We served random images from the Amazon dataset and got independent users to generate likes/dislikes.

III. MiBOOTS Framework and Methodology

We built a hybrid model combining image-based clustering with real-time user feedback model.(Figure 1) Recommendations require a two-step process for suggesting items to a user. First, we draw from a Beta-Bernoulli posterior to pick an image-based cluster. This is the foundation of the multi-armed bandits model, which is updated according to "likes." Second, once a cluster has been chosen, users are shown a specific item based on either a distance metric (< 10 likes) or an online classification. The two-step process prevents bias in the model and speeds up the selection process.

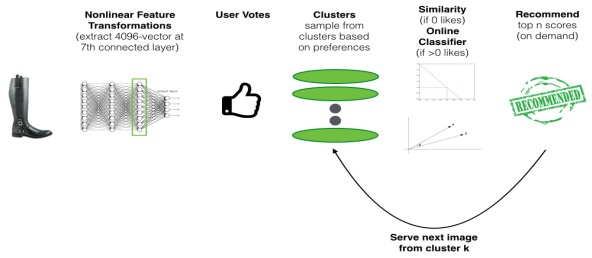

Fig. 1. A visualization of the application flow

A. CNN Feature Creation Process

In order to create features for use in image comparison, the CAFFE package [2] presents a standard open-source framework for deep learning. A CAFFE reference model with 5 convolutional layers followed by 3 fully-connected layers was used. This generalized model was trained using ImageNet [3] (1.2mln broadly focused images). For our feature set we follow the standard methodology and use the second fully-connected layer, which results in a 4096 item long feature vector.[1], [5]

B. Clustering Methods

For all clustering methods we used only the set of 4096 CNN derived features for each images and refrained from augmenting with any network, product metadata or behavioral data. These clusters formed the basis of our style spaces.

1) K-Means: K-Means was our initial choice for clustering due to its relative computational simplicity compared to other clustering methods and the large size of the dataset. K-means has a computational complexity of $O(n^{dk+1} \log n)$ where d is the dimension of the data and k is the number of clusters and n is the number of observations. K-means solves to minimize $\Sigma\Sigma||x - \mu_i||^2$. K-means is implemented using the Sklearn toolkits implementation of Lloyd's Algorithm. Below are examples of clusters created using K-means.

2) GMM Clustering: We also used Gaussian Mixture Model clustering (GMM) as an alternative to k-means. We implemented both Variational Inference (VI) and Dirichlet Process (DP) variations of GMM. Both methods are capable of determining the optimal number of clusters, and, therefore, provided less rigid structure for our clusters. This was especially important when we implemented a cluster-optimized version as shown in Algorithm 2.

3) Evaluating Clustering Results: While an initial visual check gave some confirmation that the clustering was effective, it did not provide any feedback on optimal values of K. In order to determine the optimal values of K, we looked at the behavioral data provided by amazon.

In order to evaluate clustering success and find the optimal cluster size for a style space, we looked at the ratio of in-cluster also viewed x_i to out of cluster x_o normalized by cluster size and averaged across clusters.

$$\text{Style space} = \text{argmin}_c\left(\frac{\Sigma_c \frac{\Sigma_i x_i}{n}}{c}\right)$$

where n is the number of items in the cluster and c is the number of clusters.

C. Recommendation System

The full algorithm for our recommendation system is outlined in Algorithm 1: MiBoots.

1) Multi-Armed Bandits: The introduction of Bayesian multi-armed bandits addresses two components of our recommendation systems: 1) "likes" to identify users' preferred style space (or cluster) and 2) faster classification based on image similarity. Typically, recommendation systems focus exclusively on metadata feature predictions or collaborative filtering algorithms. By using multi-armed bandits, we combined the user-specific choices with image-based features to create a hybrid method.

By using clusters of similar CNN features, we identified separate styles that users like without restricting our model to a binary substitute-or-complement segmentation (one that seems unrealistic in practice). For example, if a user has a high posterior for cluster 2 and cluster 5, the user will be shown these two clusters disproportionately higher than other clusters, regardless of the image similarity between

Algorithm 1: MiBoots

Input : $C_1, ...C_k$:clusters no, $X_1, ..., X_n$:items, L:likes, I:items seen

Output: X_{new}:newest item recommended for user

Initialize clusters $(C_1, ...C_k)$
repeat
  for *each new item* **do**
    $\max(C_i \sim \beta_i(1 + L_i, 1 + I_i))$
    if $I < 10$ *and* $L < 1$ **then**
      $X_{new} = \max(cosine(X^{\setminus I}_{1,c}, ..., X^{\setminus I}_{p,c}))$
    else
      w:= $\arg\min_w \max(0, (y - \text{logit}(w^T x^I))^2 + \alpha(\lambda|w|_1) + (1 - \alpha)(\lambda|w|_2)$
      $X_{new} = \max(logit(w^T X)$ for $X^{\setminus I} \in X_c)$
    end
    User input, $y_{new} \in L$ (if like)
    User input, $y_{new} \in I$
    Update $\beta_i \leftarrow \beta_i(1 + L_i, 1 + I_i)$
  end
until *user finished shopping*;

the two groups. We used a Bayesian approach to multi-armed bandits, also referred to as Thompson Sampling [4], [14]. We updated our posterior based on likes according to a Beta-Bernoulli model [6] and set our prior as Beta(1,1). However, as a next step, we should update our initial priors based on all user data to create a more realistic distribution for style-space preferences.

In addition, our image-based clusters decreased the search space for predictions at each iteration but still allowed exploration through the bandits approach. Over time, the algorithm hones in on specific clusters, but it allows for variation in prediction.

2) Within-Cluster Prediction: Multi-armed bandits allows us to hone in on specific image clusters. However, we then need to create predictions within clusters. We assume that we start with no user data. To initialize, we choose randomly within the cluster.

With fewer than 10 boots and no likes, we take a first pass based simply on distance metrics. We used cosine similarity to display items in the chosen cluster that are closest to previous liked items and random if the user has not currently liked any items.

After the initial sample size requirement (rated at least 10 boots and liked at least one) is reached, we switch to a classification model. We trained using 5-fold cross validation and oversampling the minority class until the proportions were roughly equal. We used user-generated test set and gauged the success of each model by using the area under the ROC curve. After exploring various models including Naive Bayes, SVMS, and Gradient Boosted Trees, we landed on regularized logistic model with optimal parameters of penalty of $alpha = 0.01$ and and elastic net L1-ratio of 0.7.

These models were trained in batch, and were built to be used for the batch recommendations. To handle the serving of images, we leverage the Stochastic Gradient Descent optimization algorithm. We chose scikit-learn's implementation, called SGDClassifier. Our trials yielded different results when training on one sample at a time, compared to training in batch. To optimize for predictive accuracy, we re-train in batch every 10 reviews. For the subsequent 9 reviews, we update the model with that 1 sample.

D. Cluster Optimization

In the framework laid out above, the clusters are static. In this section, we propose a method for using the classification model weights as feature subset selection to re-cluster each iteration. $y = logit(w^T X)$ where $X_{new} = w^T X$. We then change the posterior for each cluster in multi-armed bandits to also reflect this change. Because updating clusters is slow in practice, we only did this process for a user's next session. Over time, the model would learn the features after each session that the user cares about and adjust the style space accordingly.

This updated model is outlines in Algorithm 2.

Algorithm 2: MiBoots

Input : $C_1, ...C_k$:clusters no, $X_1, ..., X_n$:items, L:likes, I:items seen

Output: X_{new}:newest item recommended for user

Initialize clusters $(C_1, ...C_k)$
for *multiple user sessions* **do**
  repeat
    for *each new item* **do**
      $\max(C_i \sim \beta_i(1 + L_i, 1 + I_i))$
      if $I < 10$ *and* $L < 1$ **then**
        $X_{new} = \max(cosine(X^{\setminus I}_{1,c}, ..., X^{\setminus I}_{p,c}))$
      else
        w:= $\arg\min_w \max(0, (y - \text{logit}(w^T x^I))^2 + \alpha(\lambda|w|_1) + (1 - \alpha)(\lambda|w|_2)$
        $X_{new} = \max(logit(w^T X)$ for $X^{\setminus I} \in X_c)$
      end
      User input, $y_{new} \in L$ (if like)
      User input, $y_{new} \in I$
      Update $\beta_i \leftarrow \beta_i(1 + L_i, 1 + I_i)$
    end
  until *user finished shopping*;
  $X' = X \not< w>0$
  Update $C_1, ..., C_k \leftarrow$ Cluster(X')
end

IV. Discussion

The inspiration behind MiBOOTS is centered on the idea that the online shopping experience largely remains disconnected from real-time recommendations based on visual data. While many sites integrate compliments ("Customers Who Bought this Item Also Bought...") and substitutes ("Customers Who Viewed this Item Also Viewed...") on product pages, these sites do not learn or serve up items that are similar to viewed items. There is good reason for that: sites do not want to bias customers to one type of item and limit their potential revenue.

However our user testing and research has shown that sites should indeed learn the preferences of users.[17] For example, if a user consistently dislikes tall boots, the application should be less inclined to serve images of tall boots. Embedding intelligence in the serving of images has the added benefit of increasing sample size in areas of the search space where the user seems interested. This results in more information gain per vote than random selection.

This spawned a need for a fast and scalable algorithm to make recommendations. A user may be satisfied to wait two seconds to receive a batch list of recommended boots on demand, but the serving of images should happen in real-time.

Another unique aspect of our problem is the shape. Our user-generated datasets have 4096 features and very few samples. This necessitated an algorithm capable of performing feature selection and generating a sparse model.

Furthermore, by using multi-armed bandits, we do not add significant bias toward predictions since each cluster is chosen at random from a beta posterior. Our model can generate both substitutes and complements depending on the needs of the user at each shopping session.

One final challenge we learned from user testing is the problem of class imbalance. One particular user rated 150 boots and liked only 4.

With all of these problems in mind, we considered options for both similarity measures and classification models. Each has its advantages and disadvantages. Our first approach to classification was simple and interpretable. Given the boots that a user has liked, we find the unviewed boots with the highest cosine similarity. This proved to be effective on very few training examples based on user testing. Conversely, classification models are completely ineffective with only a handful of training samples. However, this similarity approach has two major pitfalls. One is that this method effectively throws away the negative samples. A similarity model using both negative and positive examples becomes much more complicated and less interpretable. Computing pairwise similarity at every iteration can also prove to be a computationally expensive task, and our research did not yield any methods for online updating of cosine similarity.

The results can be seen in the following section.

V. Results

A. Batch Model

We tested six different classification models on two different user-generated datasets. One of the datasets contains roughly 20% likes over 160 samples, and the other contains 148 samples with 3% likes.

We used scikit-learn's GridSearchCV function to run a grid search using 10-fold cross validation for each of the six models, excluding XGBoost.

We also attempted to oversample the positive samples to balance the training set. We found this to complicate the process without improving the predictive accuracy.

The results below are based on training 80 samples from the training set, and AUC calculated from the remaining samples.

XGBoost performs significantly better than all others, and is the only one of the six to have an AUC significantly greater than 0.50 for the heavily skewed dataset 2.

However, XGBoost is also the slowest of the models to train and score all of the unseen data. Given these results, we decided to use XGBoost for making on-demand recommendations. The time is longer than we'd like for running after every user vote, but it is not too long to use for generating and visualizing the best recommendations for the user.

B. Training Set 1 Results: 127 dislikes, 33 likes

URLs:

- Training Set 1: Balanced Batch Classification Notebook
- Training Set 2: Unbalanced Batch Classification Notebook

Model	Train/Score Time(ms)	AUC	AUC (Balanced)**
XGBoost	2370	0.83	0.75
SVM*	1070	0.67	0.50
Naive Bayes	542	0.63	0.59
Random Forest	1110	0.50	0.59
Log. Reg. (Elastic Net)	1030	0.50	0.52
SGD Classifier*	1300	0.74	0.63

C. Training 2 Results: 142 dislikes, 4 likes

URLs

- Training Set 2: Balanced Batch Classification Notebook
- Training Set 2: Unbalanced Batch Classification Notebook

Model	Train/Score Time(ms)	AUC	AUC (Balanced)**
XGBoost	2120	0.71	0.50
SVM*	879	0.50	0.50
Naive Bayes	887	0.50	0.50
Random Forest	563	0.50	0.50
Log. Reg. (Elastic Net)	1640	0.58	0.54
SGD Classifier*	1370	0.50	0.50

* SVM and SGDClassifier can take class weight (form of balancing) as a tuning parameter

** Datasets balanced using oversampling

Below are the ROC curve, and a visualization of the features XGBoost treats as most important.

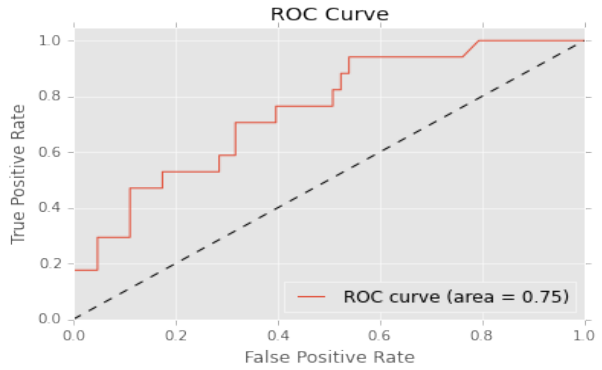

Fig. 2. ROC Curve for Dataset1, XGBoost

Fig. 3. Visualization of boots with highest positive and negative values, the feature with highest importance per our XGBoost model

Fig. 4. Visualization of boots with highest positive and negative values, the feature with second highest importance per our XGBoost model

D. Online Model

The remaining algorithms are comparable in performance. One major benefit of SGD Classifier is the option to update an existing model with a new dataset. Considering the comparable time and performance, this benefit made it a strong candidate for the algorithm to select which images to show next.

`06b_unbalanced_batch_classification.ipynb` contains the testing of the online aspect of the model.

The AUC for training in batch is 0.745. The AUC for training incrementally (updating the model one-at-a-time n times) is 0.737. Batch training takes 40.5ms. Incremental training takes 558ms. One incremental update (update using one sample) took 28ms.

Even though in aggregate, batch training is much faster, we are able to shave 12.5ms off the user's wait time between images. This comes with comparable performance to the batch method.

The following parameters from the most successful run:

```
Parameters: {'alpha': 1.6e-07,
'l1_ratio': 0.20000000000000001,
'loss': 'squared_hinge',
'class_weight': {0: 0.5, 1: 0.5}}
```

We use class weight as a proxy for over- or undersampling. It should be noted that there is significant noise in the grid search settings, but we frequently see the algorithm choose even class weight.

The squared hinge loss makes this like a linear SVM, however the features were put in very high dimensions by the ConvNet. This method is analogous to kernelizing the data in training a Support Vector Machine.

E. Current Model Implementation

In our implementation as of this writing, we use the following:

- SGDClassifier (scikit-learn with roughly the above parameter settings) to quickly update the model coefficents and serve the next image. In this case, the critical feature is the speed with which the model gets updated.
- XGBoost for making recommendations on demand. In this situation, we're willing to sacrifice speed for performance.

VI. Application

To test our model, we used a web application to test speed and efficacy. The application can be found on our website. Click here.

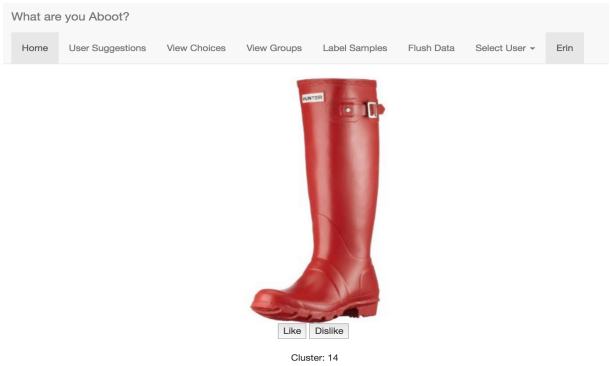

Fig. 5. A boot being shown to a user

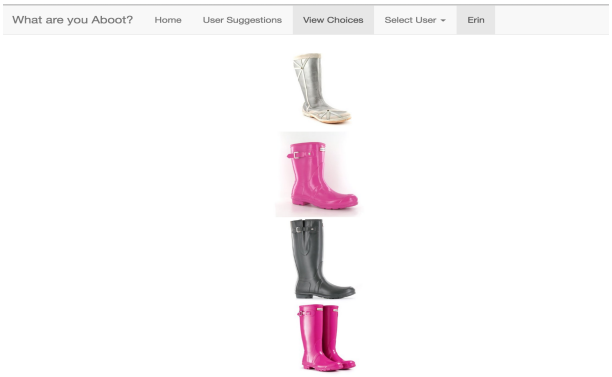

Fig. 6. This user has "liked" mostly pink rain boots

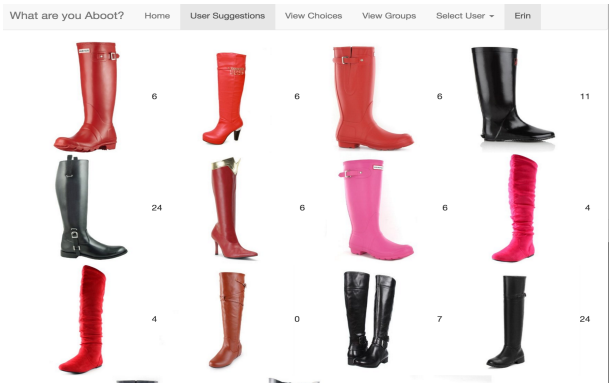

Fig. 7. The algorithm recommends boots with similar features to those liked (and dissimilar from those disliked)

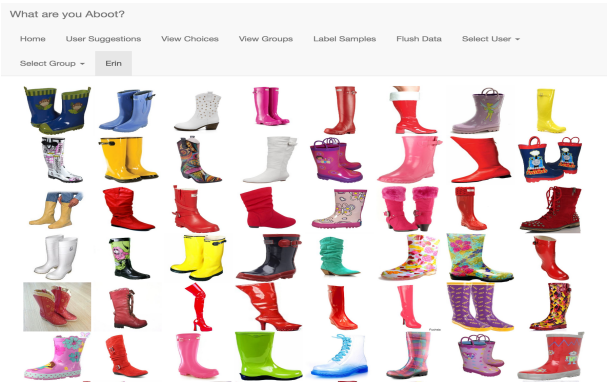

Fig. 8. A look at one of the clusters

VII. Conclusions

MiBOOTS achieves the dual goal of real-time recommendations that incorporates image features and user likes/views. By incorporating clustering, multi-armed bandits, and online classification models, we created a recommendation system flexible enough to incorporate complementary and substitute items, fast enough to work in real-time, and scalable enough to expand outside of product categories. To truly expand on this system, we would like to include more user data, different online brands, and more diverse product set as a next step.

Appendix

All additional graphs and code can be viewed on our github page:

- ImageRec Github Link
- Training Set 1: Balanced Batch Classification Notebook
  Training Set 2: Unbalanced Batch Classification Notebook
- Training Set 2: Balanced Batch Classification Notebook
- Training Set 2: Unbalanced Batch Classification Notebook

References

[1] J. McAuley, C. Targett, J. Shi, A. van den Hengel. Image-based recommendations on styles and substitutes SIGIR, 2015
[2] Y. Gia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional Architecture for Fast Feature Embedding. ACM, 2014.
[3] http://www.image-net.org/papers/imagenet_cvpr09.pdf
[4] M. Strens. A Bayesian framework for reinforcement learning, In Proceedings of the Seventeenth International Conference on Machine Learning, 2000.
[5] A. Karpay. What a Deep Neural Network thinks about your selfie. http://karpathy.github.io/2015/10/25/selfie/, 2015.
[6] C. Stucchio. Bayesian Bandits - optimizing click throughs with statistics. *www.chrisstucchio.com/blog/2013/bayesian_bandit.html*, 2013.
[7] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. ACM TOMCCAP, 2006.
[8] G. Linden, B. Smith, and J. York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. IEEE Computer Society, 2003.
[9] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. In SIGIR, 2002.

[10] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. IEEE Trans. on PAMI, 2000.
[11] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. TKDD, 2005.
[12] J. McAuley, R. Pandey, J. Leskovec. Inferring Networks of Substitutable and Complementary Products. ACM, 2015.
[13] X. Jin, J. Luo, J. Yu, G. Wang, D. Joshi, and J. Han. Reinforced similarity integration in image-rich information networks. IEEE Trans. on KDE, 2013.
[14] E. Brochu, M. Hoffman, N. Frietas, Portfolio Allocation for Bayesian Optimization, Proceedings of International Joint Conferences on Artificial Intelligence, 2015.
[15] Amazon Product Data. http://jmcauley.ucsd.edu/data/amazon/
[16] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, T. Darrell, DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition.
[17] J. Grahl, F. Rothlauf, O. HinzHow. Do Social Recommendations Influence Shopping Behavior: A Field Experiment.
[18] J.B. Schafer, J. A. Konstan, J. Riedl. E-Commerce Recommendation Applications. Data Mining and Knowledge Discovery, 2001.