Decomposing Fit Semantics for Product Size Recommendation in Metric Spaces

Rishabh Misra
UC San Diego
r1misra@eng.ucsd.edu

Mengting Wan
UC San Diego
m5wan@eng.ucsd.edu

Julian McAuley
UC San Diego
jmcauley@eng.ucsd.edu

ABSTRACT

Product size recommendation and fit prediction are critical in order to improve customers' shopping experiences and to reduce product return rates. Modeling customers' fit feedback is challenging due to its subtle semantics, arising from the subjective evaluation of products, and imbalanced label distribution. In this paper, we propose a new predictive framework to tackle the product fit problem, which captures the semantics behind customers' fit feedback, and employs a metric learning technique to resolve label imbalance issues. We also contribute two public datasets collected from online clothing retailers.

1 INTRODUCTION

With the growth of the online fashion industry and the wide size variations across different clothing products, automatically providing accurate and personalized fit guidance is worthy of interest. As retailers often allow customers to provide fit feedback (e.g. "small", "fit", "large") during the product return process or when leaving reviews, predictive models have been recently developed based on this kind of data [1, 8, 9]. A few recent approaches to this problem use two sets of latent variables to recover products' and customers' "true" sizes, and model the fit as a function of the difference between the two variables [8, 9].

However, we notice that customers' fit feedback reflects not only the objective match/mismatch between a product's true size and a customer's measurements, but also depends on other subjective characteristics of a product's style and properties. For example, in Fig. 1 we see two customers expressing concerns that a jacket seems 'baggy,' but both have different feedback regarding this fit shortcoming. Additionally, such fit feedback is ordinal in nature and is unevenly distributed (most transactions are reported as "fit"), which differs from general item recommendation tasks and requires domain-specific techniques.

In this paper, we pose product size recommendation problem as fit prediction problem and tackle the aforementioned challenges in the following ways: First, unlike previous work which focuses on recovering "true" sizes, we develop a new model to factorize the semantics of customers' fit feedback, so that representations can capture customers' fit preferences on various product aspects (like shoulders, waist etc.). We apply an ordinal regression procedure to learn these representations such that the order of labels is preserved (Step 1 in Fig. 1). Second, using a heuristic we sample good representations from each class and project them to a metric space to address label imbalance issues (Step 2 in Fig. 1).

We collect customers' fit feedback from two different clothing websites and contribute two public datasets. Through experiments on these datasets, we show the effectiveness of uncovering fine-grained aspects of fit feedback and highlight the ability of metric learning approaches with prototyping in handling label imbalance issues.

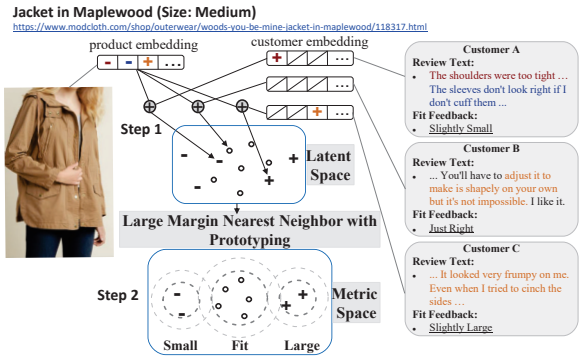


Figure 1: Workflow of the proposed framework. In Step 1, we learn customers' and products' embeddings from transactions containing fit feedback. Using these latent representations, in Step 2 we learn good representations for each class by applying prototyping and metric learning techniques.

2 RELATED WORK

The product size recommendation problem is fairly recent with a only few studies proposed so far [1, 8, 9]. One approach recovers products' and customers' "true" sizes and uses these as features in a standard classifier for fit prediction [8]. In parallel to our work, another approach extends the above method and proposes Bayesian logit and probit regression models with ordinal categories to model fit [9]. Our approach differs from these studies in that we focus on capturing fit semantics and handle label imbalance issues using metric learning approaches with prototyping. Another recent approach uses skip-gram models to learn customers' and products' latent features [1]. However, this approach assumes the availability of platform-specific features whereas our model works on more limited (and thus more readily available) transaction data.

RecSys '18, October 2–7, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-5901-6/18/10…$15.00
https://doi.org/10.1145/3240323.3240398

Metric learning has previously been applied to several recommendation problems. For music recommendation, one study encodes songs by their implicit feedback and employs metric learning to retrieve songs similar to a query song [6]. Another study uses users' meta-data to build representations and employs metric learning to retrieve suitable partners for online dating recommendation [7]. In contrast, we learn representations of transactions that capture the ordinal nature of fit. Recently, collaborative filtering was combined with metric learning and its effectiveness was shown on various recommendation tasks [3]. However, the approach (that relies on binary implicit feedback data) does not translate directly to the ordinal nature of the product size recommendation problem.

Many studies have used prototyping techniques with Nearest Neighbor based classification methods. Köstinger et al. [5] propose to jointly identify good prototypes and learn a distance metric, and also show that this leads to better generalization compared to k-Nearest Neighbor (k-NN) classification. Another study proposes a novel algorithm for deriving optimal prototypes, specifically for the 1-NN setting [11]. Following these lines, we develop a simple, fast and effective heuristic to choose relevant prototypes for Large Margin Nearest Neighbor (LMNN).

3 METHODOLOGY

3.1 Learning Fit Semantics

To model fit semantics, we adopt a latent factor formulation. We assign a score to each transaction which is indicative of the fitness of a corresponding product on the corresponding customer. In particular, if a customer c buys a product p (e.g. a medium jacket) which is a specific size of the parent product pp (the corresponding jacket), then the fitness score of this transaction t is modeled as

$$f_w(t) = \Big\langle \underbrace{\mathbf{w}}_{\text{weight vector}} , \overbrace{\alpha \oplus b_{t_c} \oplus b_{t_{pp}}}^{\text{fit bias terms}} \oplus \underbrace{(\mathbf{u_{t_c}} \odot \mathbf{v_{t_p}})}_{\text{fit compatibility}} \Big\rangle \quad (1)$$

where $\mathbf{u_{t_c}}$ and $\mathbf{v_{t_p}}$ are K-dimensional latent features, α is a global bias term, $\oplus$ denotes concatenation and $\odot$ denotes element-wise product. The bias term $b_{t_{pp}}$ captures the notion that certain products tend to be reported more 'unfit' because of their inherent features/build, while b_{t_c} captures the notion that certain customers are highly sensitive to fit while others are more accommodating (as shown in Fig. 1).

In order to recommend appropriate sizes to customers, our model should preserve fit ordering. That is, if a product size is small (resp. large) for a customer, all smaller (larger) sizes of the corresponding parent product should also be small (large). We achieve this behavior by enforcing an order in fitness scores of different size variants of a parent product. To that end, we require that for each product p, all its latent factors are strictly larger (smaller) than the next smaller (larger) catalog product p^- (p^+), if a smaller (larger) size exists. This works since for a given customer and parent product, fitness scores vary only based on p's parameters.

We tune the parameters of the scoring function such that for each transaction t, the Hinge loss, defined as follows, of an ordinal regression problem is minimized subject to monotonicity constraints:

$$\min L(t) = \begin{cases} \max\{0, 1 - f_w(t) + b_2\} & \text{if } Y_t = \text{Large} \\ \max\{0, 1 + f_w(t) - b_2\} & \\ + \max\{0, 1 - f_w(t) + b_1\} & \text{if } Y_t = \text{Fit} \\ \max\{0, 1 + f_w(t) - b_1\} & \text{if } Y_t = \text{Small} \end{cases} \quad (2)$$

$$\textbf{s.t. } \mathbf{v_{t_{p^-}}} < \mathbf{v_{t_p}} < \mathbf{v_{t_{p^+}}}.$$

Here b_1 and b_2 are the thresholds of an ordinal regression problem (with $b_2 > b_1$). The overall loss is simply the sum of losses for each transaction and is minimized when, for any transaction t with fit outcome Y_t, $f_w(t) > b_2$ when Y_t = Large, $f_w(t) < b_1$ when Y_t = Small and $b_1 < f_w(t) < b_2$ when Y_t = Fit, subject to monotonicity constraints on t_p's latent factors. We use Projected Gradient Descent for optimization.

3.2 Metric Learning Approach

We propose the use of metric learning with prototyping to handle the issue of label imbalance. To that end, our prototyping technique first alters the training data distribution by re-sampling from different classes, which is shown to be effective in handling label imbalance issues [4]. Secondly, LMNN [10] improves the local data neighborhood by moving transactions having same fit feedback closer and having different fit feedback farther (see Fig. 1), which is shown to improve the overall k-NN classification [10]. We describe our approach in detail in the following subsections.

3.2.1 Metric Learning. The goal of metric learning is to learn a distance metric $\mathbf{D}$ such that $\mathbf{D}(k, l) > \mathbf{D}(k, m)$ for any training instance (k, l, m) where transactions k and l are in the same class and k and m are in different classes. In this work, we use an LMNN metric learning approach which, apart from bringing transactions of the same class closer, also aims at maintaining a margin between transactions of different classes. LMNN does this by (1) identifying the target neighbors for each transaction, where target neighbors are those transactions that are *desired* to be closest to the transaction under consideration (that is, few transactions of the same class) (2) learning a linear transformation of the input space such that the resulting nearest neighbors of a point are indeed its target neighbors. The final classification in LMNN is given by k-NN in a metric space. The distance measure $\mathbf{D}$ used by LMNN is the Mahalanobis distance which is parameterized by matrix $\mathbf{L}$.

If the vector representation of a transaction $\vec{x}$ is a function of corresponding latent factors and biases (details in Algorithm 1), then mathematically LMNN aims to optimize the following objective for all (k, l, m) triplets in the training data:

$$\textbf{Minimize } (1-\mu) \sum_{k, l \rightsquigarrow k} ||\mathbf{L}(\vec{x}_k - \vec{x}_l)||_2^2 + \mu \sum_{k, l \rightsquigarrow k, m} (1 - y_{km}) \xi_{klm}$$

$$\textbf{s.t. } ||\mathbf{L}(\vec{x}_k - \vec{x}_m)||_2^2 - ||\mathbf{L}(\vec{x}_k - \vec{x}_l)||_2^2 \geq 1 - \xi_{klm},\ \xi_{klm} \geq 0 \quad (3)$$

where $\mathbf{L}$ is a real-valued transformation matrix, $l \rightsquigarrow k$ denotes the target neighbors of k, y_{km} is 1 if k and m belong to the same class (otherwise 0), $\xi_{(.)}$ are slack variables for each (k, l, m) triplet and μ is a trade-off parameter. The first part of the objective tries to minimize the distance between transactions and their target neighbors whereas the second part tries to minimize the margin

Algorithm 1: FitPredictionAlgorithm

Initialization:
for *each customer c* **do**
  $\mathbf{u_c} \sim \mathbf{N}(0, 0.1), b_c \sim N(0, 0.1)$
for *each parent product pp* **do**
  Set $r = 1$ and L = list of sizes in pp in increasing order
  $b_{pp} \sim N(0, 0.1)$
  for *each product p in L* **do**
    $\mathbf{v_p} \sim \mathbf{N}(r, 0.1)$
    Set $r = r + 1$
Set $\alpha = 1$, $\mathbf{w} = \mathbf{1}_{K+3}$, $b_1 = -5$ and $b_2 = 5$

Optimization 1:
while *NOT CONVERGED* **do**
  Tune parameters $\mathbf{u_c}$s, $\mathbf{v_p}$s, α, b_cs, b_{pp}s, $\mathbf{w}$, b_1 and b_2 using Projected Gradient Descent such that the sum of Hinge losses (eq. (2)) over all transactions is minimized.

Prototyping:
(1) For each transaction t, create representation of t as $[\mathbf{w}[1].b_{t_c} \oplus \mathbf{w}[2].b_{t_{pp}} \oplus (\mathbf{w}[3:] \odot \mathbf{u_{t_c}} \odot \mathbf{v_{t_p}})]$
(2) Find the centroid for each class *cls*
(3) Sort distances of all points from the respective class's centroid in increasing order and store them in X_{cls}
(4) For each class *cls*, set $\text{interval}_{cls} = \frac{\#\text{ transactions in } cls}{p_{cls}}$
(5) **for** *each class cls* **do**
  Set $X_{cls} = X_{cls}[r_{cls}:]$
  for *i = 1 to n* **do**
    (a) Select $X_{cls}[i * \text{interval}_{cls}]$ as a prototype
    (b) $\text{interval}_{cls} += q_{cls}$

Optimization 2:
while *NOT CONVERGED* **do**
  Tune $\mathbf{L}$ on the selected prototypes using L-BFGS to optimize the objective given in eq. (3).

Predict:
for *each test transaction t* **do**
  (1) Project $\vec{x}_t$ in the metric space using $\mathbf{L}$
  (2) Apply k-NN in the metric space to classify t

violations from the transactions of different classes and the hyper-parameter μ trades-off between the two. The constraints enforce a margin between transactions of different classes while allowing for a non-negative slack of $\xi_{(.)}$. We use the PyLMNN package[1] for the implementation of LMNN which tunes the parameters using L-BFGS.

3.2.2 Prototyping. LMNN fixes the k target neighbors for each transaction *before* it runs, which allows constraints to be defined locally. However, this also makes the method very sensitive to the ability of the Euclidean distance to select relevant target neighbors [2]. Prototyping techniques, which aim to select a few representative examples from the data, have been shown to increase processing speed while providing generalizability [5]. Additionally, re-sampling methods are shown to tackle label imbalance issues [4]. Thus, to mitigate the aforementioned limitation of Euclidean distances and tackle label imbalance issues, we develop a heuristic that provides a good representation for each class by reducing noise from outliers and other non-contributing transactions (like the ones which are too close to the centroid of their respective class or to already selected prototypes; details in Algorithm 1) by carefully sampling prototypes. Furthermore, since the time-complexity of LMNN is linear in the size of the data [5], training LMNN on selected prototypes incurs only constant time if the number of selected prototypes ($\sim$ 300 in our experiments) is much less than the size of the original training data.

[1] https://pypi.org/project/PyLMNN/1.5.2/

4 DATASETS

We introduce[2] two datasets obtained from the websites *ModCloth*[3] and *RentTheRunWay*.[4] *ModCloth* sells women's vintage clothing and accessories, from which we collected data from three categories: dresses, tops, and bottoms. *RentTheRunWay* is a unique platform that allows women to rent clothes for various occasions; we collected data from several categories. These datasets contain self-reported fit feedback from customers as well as other side information like reviews, ratings, product categories, catalog sizes, customers' measurements (etc.). In both datasets, fit feedback belongs to one of three classes: 'Small,' 'Fit,' and 'Large.'

Statistic/Dataset	ModCloth	RentTheRunWay
# Transactions	82,790	192,544
# Customers	47,958	105,571
# Products	5,012	30,815
Fraction Small	0.157	0.134
Fraction Large	0.158	0.128
# Customers with 1 Transaction	31,858	71,824
# Products with 1 Transaction	2,034	8,023

Table 1: General dataset statistics.

General statistics of the datasets are provided in Table 1. Note that a 'product' refers to a specific size of a product, as our goal is to predict fitness for associated catalog sizes. Also, since different items use different sizing conventions, we standardize sizes into a single numerical scale preserving the order. Note that these datasets are highly sparse, with most products and customers having only a single transaction.

5 EXPERIMENTS

5.1 Comparison Methods

We compare the performance of the following five methods:

- **1-LV-LR**: This method assumes a single latent variable for each customer and product and uses $f_w(t) = w(u_{t_c} - v_{t_p})$ as the scoring function. The learned features are then used in Logistic Regression (LR) for final classification, as is done in [8].
- **K-LV-LR**: This method assumes K latent variables and one bias term for each customer and product and has a scoring function of

[2] Code and datasets are available at https://rishabhmisra.github.io/
[3] https://modcloth.com
[4] https://renttherunway.com

Dataset/Method	(a) 1-LV-LR	(b) K-LV-LR	(c) K-LF-LR	(d) K-LV-ML	(e) K-LF-ML	improvement (e) vs. (a)	improvement (e) vs. (b)	improvement (e) vs (c)
ModCloth	0.615	0.617	0.626	0.621	0.657	**6.8%**	6.5%	4.9%
RentTheRunWay	0.61	0.676	0.672	0.681	0.719	**17.9%**	6.4%	7%

Table 2: Performance of various methods in terms of average AUC.

the form $f_w(t) = \left\langle \mathbf{w}, (b_{t_c} - b_{t_{pp}}) \oplus (\mathbf{u}_{\mathbf{t_c}} - \mathbf{v}_{\mathbf{t_p}}) \right\rangle$. This is a simple extension of 1-LV.

- **K-LF-LR**: The method proposed in Section 3.1. This method also uses LR for final classification.
- **K-LV-ML**: Uses the metric learning approach, instead of LR, with K-LV to produce the final classification.
- **K-LF-ML (proposed method)**: Uses the metric learning approach, instead of LR, with K-LF to produce the final classification.

These methods are designed to evaluate (a) the effectiveness of capturing fit semantics over "true" sizes; (b) the importance of learning good latent representations; and (c) the effectiveness of the proposed metric learning approach in handling label imbalance issues. The AUC is used to evaluate the performance of these methods, following the protocol from [8].

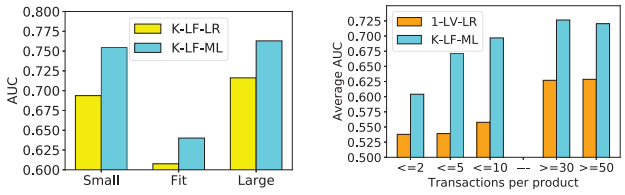
5.2 Experimental Setup

Training, validation and test sets are created using an 80:10:10 random split of the data. For ordinal regression optimization, we apply ℓ_2 regularization. The hyper-parameters (learning rate and regularization constant in ordinal regression; n, $\{r_{cls}\}$, $\{p_{cls}\}$ and $\{q_{cls}\}$ in prototyping; and k of k-NN and output dimensionality of $\mathbf{L}$ in LMNN) are tuned on the validation set using grid search. The PyLMNN package uses Bayesian optimization to find hyper-parameters internal to LMNN optimization. For the 1-LV method, we used the setting described in [8].

6 RESULTS AND ANALYSIS

Our results are summarized in Table 2. We find that models with K-dimensional latent variables outperform the method with one latent variable. Furthermore, we observe that improvements on *ModCloth* are relatively smaller than improvements on *RentTheRunWay*. This could be due to *ModCloth* having relatively more cold products and customers (products and customers with very few transactions) compared to *RentTheRunWay* (Table 1). Of note is that metric learning approaches do not significantly improve performance when using representations from the K-LV method. The reason for this could be that K-LV does not capture biases from data, as bias terms merely act as an extra latent dimension, and learns representations which are not easily separable in the metric space. This underscores the importance of learning good representations for metric learning. Finally, we see that K-LF-ML substantially outperforms all other methods on both datasets. Besides learning good representations, this could be ascribed to the ability of the proposed metric learning approach in handling label imbalance issues (Fig. 2a).

Next we analyze how our method performs in cold-start and warm-start scenarios (Fig. 2b). For cold products, we notice that K-LF-ML consistently performs better than 1-LV-LR, although their performances are slightly worse overall. As we consider products



(a) Effectiveness of metric learning approach in classifying infrequent labels. **(b) Average test AUC for products with given number of transactions in training data.**

Figure 2: Effectiveness of the proposed method in handling label imbalance and cold-start issues on *RentTheRunWay*.

with more transactions, K-LF-ML improves quickly. The performance of 1-LV-LR improves significantly given sufficiently many samples. We observe the same pattern for cold customers.

Finally we perform a neighborhood analysis of transactions in our learned metric space to see if there are any pattens in terms of fit semantics. By looking at reviews, we found that neighboring transactions generally capture fitness along similar body regions (Fig. 3). We also report cosine similarities to give a sense of how similar vector representations are.

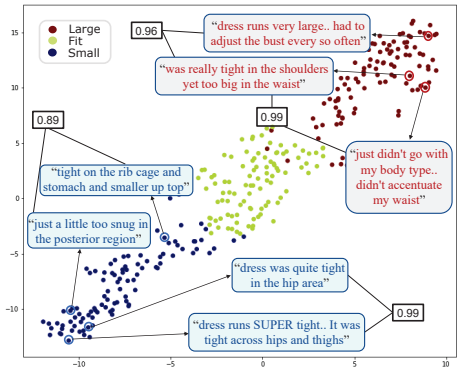


Figure 3: Neighborhood analysis of t-SNE embeddings of a selection of transactions from *RentTheRunWay*.

7 CONCLUSION

In this study, we propose a new predictive framework for the product size recommendation problem. Specifically, we adopt a latent factor formulation to decompose the semantics of customers' fit feedback and employ a metric learning approach to address label imbalance issues. Furthermore, we contribute two public datasets which contain customers' self-reported fit feedback. Quantitative and qualitative results on these datasets show the effectiveness of the proposed framework.

REFERENCES

[1] G Mohammed Abdulla and Sumit Borar. 2017. Size Recommendation System for Fashion E-commerce. *KDD Workshop on Machine Learning Meets Fashion* (2017).

[2] Aurélien Bellet, Amaury Habrard, and Marc Sebban. 2013. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709* (2013).

[3] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *Proceedings of the 26th International Conference on World Wide Web*.

[4] Chen Huang, Yining Li, Chen Change Loy, and Xiaoou Tang. 2016. Learning deep representation for imbalanced classification. In *CVPR*.

[5] Martin Köstinger, Paul Wohlhart, Peter M Roth, and Horst Bischof. 2013. Joint learning of discriminative prototypes and large margin nearest neighbor classifiers. In *ICCV*.

[6] Brian McFee, Luke Barrington, and Gert Lanckriet. 2012. Learning content similarity for music recommendation. *IEEE Transactions on Audio, Speech, and Language Processing* (2012).

[7] Brian McFee and Gert R Lanckriet. 2010. Metric learning to rank. In *ICML*.

[8] Vivek Sembium, Rajeev Rastogi, Atul Saroop, and Srujana Merugu. 2017. Recommending Product Sizes to Customers. In *RecSys*.

[9] Vivek Sembium, Rajeev Rastogi, Lavanya Tekumalla, and Atul Saroop. 2018. Bayesian Models for Product Size Recommendations. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*.

[10] Kilian Q Weinberger and Lawrence K Saul. 2009. Distance metric learning for large margin nearest neighbor classification. *JMLR* (2009).

[11] Paul Wohlhart, Martin Köstinger, Michael Donoser, Peter M Roth, and Horst Bischof. 2013. Optimizing 1-nearest prototype classifiers. In *CVPR*.