Multimodal and Multitask Classification Using Artificial Neural Networks

Lucy Wang and **Charlie Guthrie**
New York University
Professor: Kyunghyun Cho

Abstract

We used brand name, product description, and image to classify fashion inventory (clothing and accessories) into 3 levels of nested categories. To find a model with optimal classification accuracy, we experimented with the following: varying input data sources from text to images to both; embedding text with a bag-of-words representation vs. long-short-term memory RNN; varying classification model architecture. In our experiments, we achieved the best result ($>$ 99% accuracy for the top category) by training the multitask feedforward neural network on both text and image data.

1 Introduction

1.1 Business Problem

Spring is a fashion marketplace with a focus on personalized user experience where users can easily search and discover products from over 800 brands. Because Spring is a marketplace where information is gathered from brands directly, there is no standardized product catalog across the platform. Some brands categorize products differently from others and some brands don't provide complete information about products.

To ensure that all information is complete and consistent across the platform, the Spring team proposed a solution to introduce an industry standard, multilevel taxonomy to catalog the millions of products on the platform. The proposed solution involves hiring a team of workers that manually categorizes the products; however, this solution is not capital-efficient and does not scale well when there is a large number of products to process every day. To address this issue, we propose an automatic classifier to categorize as many products accurately as possible and only manually fill in the information for an item when the classifier is not sufficiently confident.

Because the team has only recently started manually tagging the data for Spring, we decided to use a similar, longer dataset to train the models. Given Nordstrom has the best product taxonomy in the industry, we obtained a complete dataset of more than 750,000 products from the retailer's website.

1.2 Experiments/Hypotheses

We propose a series of experiments using a combination of feature sets, models, and data size to select the best model for identifying product taxonomy. The goal is to achieve the best accuracy with the least complexity.

We have available two types of data for each product: texts and images. We experimented with building the model using only text data, image data, and both. We hypothesized that using both types of data together would produce the best accuracy; however, it might be sufficient to only use one type of data given the simple nature of the problem. We also hoped to learn the relative value of text and image data.

Additionally, we tested text representation using a long-short term memory (LSTM) recurrent neural network (Hochreiter and Schmidhuber, 1997) and a feedforward neural network (MLP) on a bag-of-words representation. We believed that product categories would depend on the vocabulary of the description more than word sequence. Therefore,

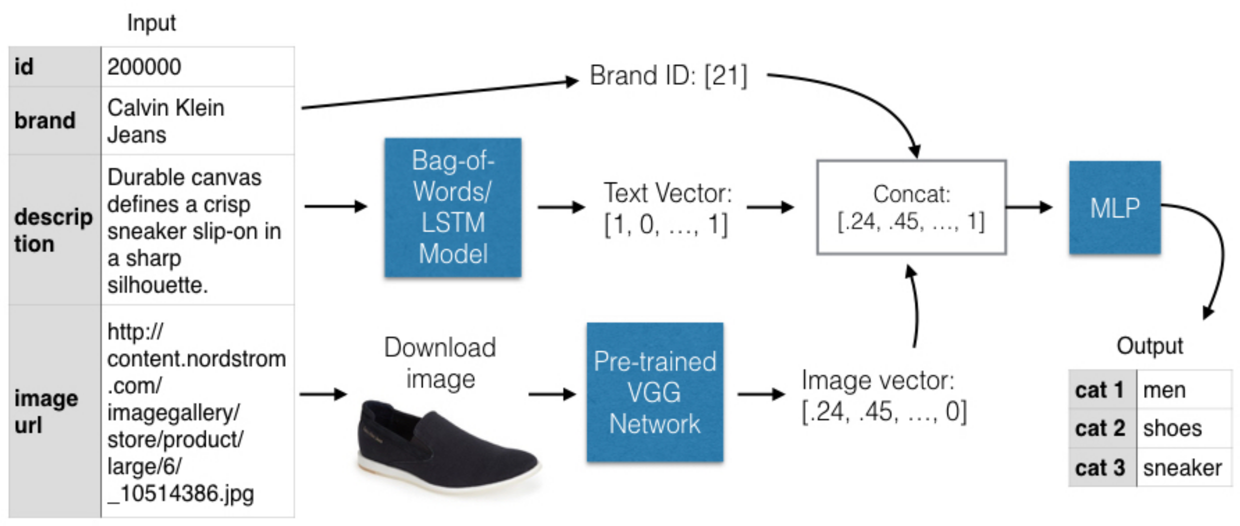
Input
id
200000
brand
Calvin Klein Jeans
description
Durable canvas defines a crisp sneaker slip-on in a sharp silhouette.
image url
http://content.nordstrom.com/imagegallery/store/product/large/6/_10514386.jpg
Brand ID: [21]
Bag-of-Words/ LSTM Model
Text Vector: [1, 0, ..., 1]
Concat: [.24, .45, ..., 1]
MLP
Download image
Pre-trained VGG Network
Image vector: [.24, .45, ..., 0]
Output
cat 1
men
cat 2
shoes
cat 3
sneaker

Figure 1: Complete model architecture. For each data point, brand name is recorded as an index and one-hot encoded. Description gets cleaned and recorded as bag-of-words vector, or run through an LSTM model. We download the image and run it through a pre-trained image recognition network to extract a feature vector. The vectors are concatenated and fed into a multi-task multilayer perceptron with three outputs: one for each level of category. Variations omitted description text, images, or both.

we hypothesized that a feedforward neural network would be sufficient for this task. Within the feedforward neural network, we also experimented with two architectures to test for accuracy and computing efficiency. Within the recurrent neural network, we experimented with sequenced predictions of each category level using either the top prediction of the previous category or the top 5 (beam search).

Finally, we experimented with different sizes of training data to figure out the optimal amount of data to use for building the model. This is an important question for us to answer. Since we are using a proxy dataset to build the model right now, it is best to use labeled data at Spring when available. Our goal here is to manually label the fewest products necessary to train a model and then apply the model to the rest of the products at Spring.

2 Data and Methodology

2.1 Training Data

The dataset of 750,000 Nordstrom products includes information such as the product name, product brand, product description, and the corresponding categories. Because this dataset was created using a scraper, a significant amount of cleaning needed to be done before it could be used for modeling. For example, the product description often contained information about the product category as well. There were also a considerable number of duplicates in the form of small variations on a single product, such as different colors or size.

In addition to product description and images, we used the brand of each item as a predictor. There were 2700 brands, which we one-hot encoded using *scikit-learn* (Pedregosa et al., 2011). Starting with brands as a baseline, we added text data, image data, or both to complete the data set, depending on the experiment being run. Again, see figure 1 for an illustration of model architecture. See figure 1 for a sample data point from the Nordstrom data set.

2.2 Target Labels

There are three levels of categories in the Nordstrom product catalog. There are 19 unique values in the first category, 40 unique values in the second, and 270 unique values in the third. In total, there are roughly 1,200 unique combinations of categories. This indicates that not all subcategories are possible for all higher level categories. For example, 'Sweaters' is not a viable subcategory of 'Hair Products'. For each inventory item, the goal was to predict each level of category from the training data. See table 1 for sample categories.

Category 1	Category 2	Category 3
men	shoes	sneakers boots oxfords ...
	bottoms	jeans pant short ...
	...	...
women	shoes	sneakers ...
	dresses	dress gown
	...	...
toddler	...	...
...	...	...

Table 1: Sample categorizations of inventory items.

2.3 Train and Test Split

We randomly split the entire data set into training and test set based on the product id. We reserved 25% of the data as the test set. In training, we left 10% of randomly sampled training data for validation.

3 Models and Architecture

We extracted image and text features separately, then concatenated the resulting feature vectors and fed into a multi-layer perceptron network. The process is summarized in figure 1 and detailed below.

3.1 Extracting Features from Images

Here we made use of the Visual Geometry Group's (VGG) Convolutional Neural Net, which was pretrained on ImageNet for classification (Chatfield et al., 2014), (Simonyan and Zisserman, 2014). . The VGG CNN is a pretrained neural network that takes in images and returns labels for those images. Show it a picture of a bicycle, for example, and it will return a ranked list of likely labels, "bicycle" being one of them. Rather than use the labels themselves as inputs for our classifier, we chose to use the last hidden layer, which contains much more information. Taking advantage of this pretrained network was much quicker and more accurate than it would have been to develop an image-recognition model ourselves.

Each item in our data set included an image url. For each item, we downloaded the corresponding image, resized and center-cropped it to fit the format of the VGG network. Once each image was preprocessed, we fed it into the VGG network to extract its feature vector to be input later into our model.

3.2 Extracting Features from Product Descriptions

3.2.1 Bag of words for MLP

We employed a simple bag of words representation using the top 5,000 words and no term weights. First, we built a dictionary of all unigrams in the training set after tokenization and removing stop words. The words in the dictionary are ordered by overall term frequency. Each product description is then turned into a sequence of word indices and we only took the top 5,000 words. This cut off eliminates all words that appeared only one time throughout the training set. To train the model, each sequence is transformed into a vector of binary values with length 5,000 using the tokenizer from the Keras Python package (keras.io).

3.2.2 Sequencing for LSTM

We performed the same tokenization step here as for bag of words. Again, we created a vocabulary dictionary ordered by term frequency from the highest to lowest. We limited the vocabulary size to 50,000 and cut out the rest. Each sentence is padded to length 100. Most product descriptions are well within 100 words.

3.3 Hyperparameters and regularization

- **LSTM Hyperparameters**: For all LSTM models we trained: 128 word embedding dimensions, 0.0001 learning rate, sigmoid nonlinear activation function. The LSTM models are implemented using Theano (Bastien et al., 2012; Bergstra et al., 2010).

- **MLP Hyperparameters**: For all MLP models we trained: 256 embedding dimensions, 3 hidden layers, 0.01 learning rate, rectified linear units. The MLP models are implemented using Lasagne (Dieleman et al., 2015).

- **Regularization**: We used 50% dropout on all hidden layers for both LSTM and MLP models, and 20% dropout on the input layer to MLP.

- We did not tune any other hyperparameters other than early stopping for LSTM training. All models were trained through SGD over shuffled minibatches with adadelta updates (Zeiler, 2012).

3.4 Classification Model from Extracted Features

3.4.1 LSTM model architecture

We experimented with two variations of the LSTM model. For each sub-category, the previous category was concatenated with the product description and brand information and fed into the neural network. In training, the actual category was used. In testing, we tested using a hard prediction (taking the top prediction for the previous category) and a soft prediction (beam search through top 5 predictions and taking the top 5 again for the next prediction).

3.4.2 MLP model architecture

We tested two variations of the MLP model as well. In the first, we used only brand and description data and trained the outputs for the three categories in parallel. The three predictions shared the same hidden layers and cost function. Each had a separate softmax layer mapping to corresponding number of categories. In the second variation, we created three models in sequence and incorporated the previous category into the softmax layer for each subsequent category. These models shared the same description and brand embedding, but were heavily biased in the softmax layer by the previous category information.

4 Results and Insights

There were several experiments being run to compare. We tried varying the data to be used, varying text embedding method, and varying the design of our neural net classifier.

4.1 Text Data vs. Image Data

The first experiment was to examine the relative effectiveness of text data vs. image data for classification. For this we compared accuracy of models trained on the following data sets:

1. Brand Index only

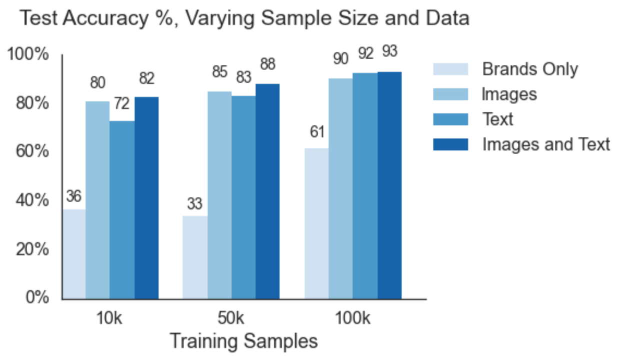

Figure 2: Comparison of models using different data inputs. The "Brands Only" model did not use text or image data. The "Text" model used Bag-of-Words representation of product descriptions.

2. Product Descriptions and Brand Index

3. Extracted Image Features and Brand Index

4. All of the above

Test accuracy results are summarized in figure 2. To arrive at a single number for comparing models, we took the average of classification accuracy across all three category levels.

Brands alone (with no image or text data) had predictably poor classification accuracy. Images and text were comparable given sufficient data, but 10,000 training samples the text-only model had lower accuracy. We suspect the image-only model had an advantage over text given low amounts of data, because the image-only model had the benefit of getting informative features from the pre-trained VGG network. Models using image and text data together consistently had the best performance.

More data, predictably, yielded better results. However, there were diminishing marginal returns from increasing the amount of data, and increasing training time and memory issues. Given that the models did not improve substantially from 50k to 100k data points, and given time constraints, we opted to stop at 100,000 training samples.

In figure 3 we examine performance of the Text-plus-Image model at individual category levels. As a benchmark, we also included a naive model that would simply classify according to whichever category had a plurality of the training data; for example, 56% of the training data was labeled as 'women'

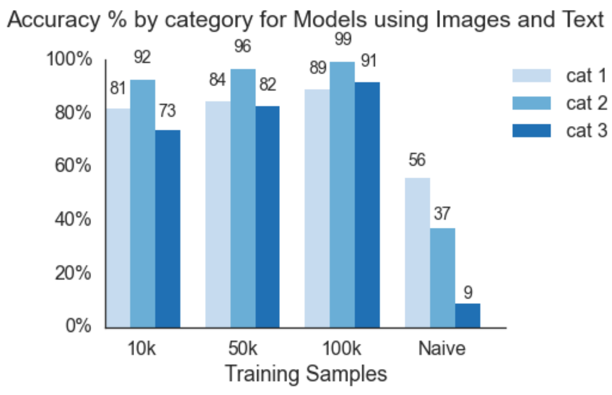

Figure 3: Looking at accuracy of the best-performing model for individual category levels. As a benchmark, we also included a naive model that would simply classify according to whichever category had a plurality of the training data.

for category 1, so this Naive model always guessed 'women' for category 1. Our classifier had highest accuracy - 99% - for category 2. Accuracy for categories 1 and 3 were close, except when fed little data. Category 1 may have had ambiguities that would not be obvious from the image or text alone: for example a pair of pants could belong in the boys', men's, women's or girls' sections. However, Category 1 was the easiest to guess at random: it had the fewest number of unique values (19) and 56% of inventory was categorized as 'women'. On the other hand, category 3 had very specific labels, but had 268 possible values and the plurality - dresses - was only 9% of the training data.

4.2 Text Models: LSTM vs. Bag-of-Words

Comparing models that used text data only: the results here indicate that word ordering is helpful for classification, at least for a low number of samples (see fig. 4).

4.3 Neural Net Design

Finally we compared performance on text data with MLP using two approaches to this multi-tier classification problem. We tested the model performances using 50,000 training data points. The multitask model significantly outperformed the sequential MLP model, even though the latter model incorporated more information for each prediction, the previous category (see fig. 5). However, this result makes a lot of sense. We built the sequential

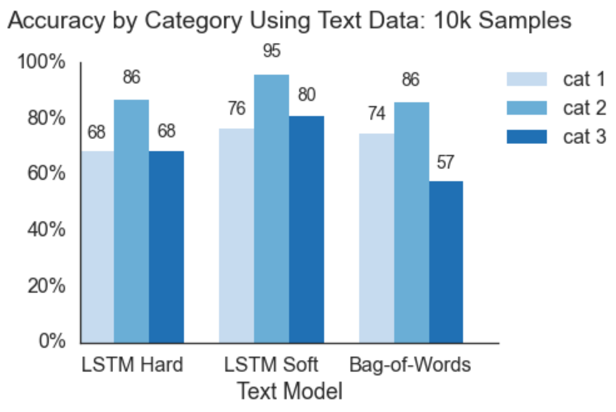

Figure 4: Text-only model performance comparing three methods for embedding text. The results indicate that word ordering is useful for classification, at least for a low number of samples.

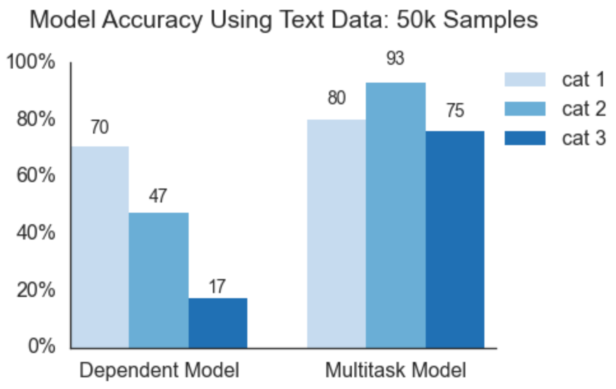

Figure 5: The dependent model used the prediction from category 1 to predict category 2, and both predictions to predict category 3. But errors in predicting category 1 compounded, resulting in worse accuracy for categories 2 and 3. The multitask model with shared parameters performed consistently better.

model so that each model shares the same word embedding layers. At the softmax layer, the previous category is concatenated to the word vector. This means the model is heavily biased by the previous category, which would result in problems similar to that of a greedy search. If the prediction of the previous category is wrong, it would lead to even more wrong predictions on subsequent tasks.

5 Next Steps

There are several next steps we have in mind. First, given the steady increase in accuracy by using more data points, more experiments can be done on bigger training sets. We only tested up to 100,000 training data. Second, more analysis can be done on misclassifications. We did not analyze the accuracy across various categories. Given some categories are more common and easier to identify than others, it would be interesting to do more analysis on which categories the model works best on. This would also help Spring to make decisions on which categories to keep a close eye on when using the model to automatically categorize products. Finally, we built the LSTM models on 10,000 training data and sequentially. It would be interesting to see how a multitasking LSTM model would perform. Given the excellent performance of the multitasking MLP model, we believe the LSTM model would produce equal if not better results.

References

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. 2012. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June. Oral Presentation.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Return of the devil in the details: Delving deep into convolutional nets. *CoRR*, abs/1405.3531.

Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, Sren Kaae Snderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, diogo149, Brian McFee, Hendrik Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. 2015. Lasagne: First release., August.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Acknowledgments

We would like to thank Kyunghyun Cho and Kelvin Xu for their guidance and support. In particular, we could not have done this without endless hours of debugging with Kelvin.