Entity Matching Stage 4 Report

Sidharth Mudgal	Dongning Wang	Jie Feng
sidharth@cs.wisc.edu	dw@cs.wisc.edu	jie.feng@wisc.edu

1. Abstract

Determining whether two given product descriptions refer to the same real world entity is a complex but important task. While traditional feature engineering based techniques may give us reasonably good accuracies in some domains, some level of natural language understanding is necessary to be able to achieve human level accuracy that is generalizable over several domains. We investigate a neural network model based on dynamic memory networks (Kumar et al., 2015) that forms a memory based on an initial "glance" and uses an attention mechanism to determine which words in the product description are relevant for the purposes of comparison. We also try traditional machine learning models for comparison. Using a training set of 15k samples, our precision on a 3k test set was ~96.8% with a recall of ~97.1 %.

2. Neural Network model

The main steps in our model are as follows:

1. Preprocess product descriptions to produce a single string of words for each product in each pair.
2. Train a character level neural language model that produces a vector for each word. We used this to convert the string of words into a variable length feature vector for each product pair.
3. Train a neural network model to classify each product pair as a match or mismatch.

2.1 Preprocessing

One of the biggest challenges in product matching is the large number of missing values. Most attributes are almost always empty. In some cases, an attribute may be listed in the text of another attribute. Since there could be several corner cases while trying to handle these problems, it may be the best to leave this task to machine learning models. Hence we simply merge together all attributes to produce a single long string of text for each product.

Next we tokenize each string using the tokenizer in Stanford's CoreNLP toolkit to produce a sequence of tokens. One of the benefits of using this tokenizer is that it incorporates several heuristics that helps it decide when single quotes are part of words, when periods don't mean end of sentences, etc..

The following two preprocessing steps are some heuristics we use to improve performance and training time:

- We remove most special characters from the tokenized strings since many of them don't add substantial information to the product description. This helps reduce our model size significantly.
- We remove long words from all product descriptions. All words longer than 15 characters are removed. This significantly reduces our vocabulary size as well as our model size.

2.2 Neural Language model for word embeddings

Unlike some other text sources such as newspapers, product descriptions tend to have a large number of typographical errors. Word level vector representations are inherently unable to deal with such frequent mistakes. One common approach to deal with rare words such as spelling errors in word level models is to reserve a representation for rarely occurring words. However, we cannot expect this to yield great results in our application since we end up assigning the same vector to most spelling errors.

To deal with these problems, we use a character based language model (Kim et al., 2015) that makes predictions at the word level and use this to create word representations. This enables us to use information about the characters in a word to help predict a more reasonable vector representation for words with typographical errors. This also helps us get reasonable representations for words that have never been seen before, such as serial numbers.

The word vector model we used (Kim et al., 2015) first performs a temporal convolution over the characters using several kernel (filter) widths. A max pooling over time operation is then performed to get a fixed size representation for each word. A 3-layer highway network (Srivastava et al, 2015) is then used to produce a final representation for each word. Using highway networks provides the model a way to avoid the attenuation of input signal in higher layers, an inherent problem in deep neural networks. For the purpose of training the language model, this is fed into a recurrent neural network with long short term memory units (Hochreiter et al, 1997).

Once the neural language model is trained, the recurrent layers are removed and the outputs of the highway layers are used to generate word vector representations for each word in the string associated with each product. This forms the input to the entity matching model.

2.3 Entity matching neural network model

The goal of this model is to predict whether a given product pair refer to the same real world entity. To do this we first need to create a summary that contains relevant information on what each product is, and then compare the summaries to determine whether or not the products match.

To generate the summaries, we use an episodic memory module (Kumar et al., 2015). The idea is to go through the product description once to take a first glance at it, and then make

the model read it a second time so that we have more context to decide what is relevant and deserves "attention". The second pass produces the final summary that is used for comparing the two products. For both passes we use gated recurrent units or GRUs (Chung et al., 2015). GRUs consist of two internal gates, an update gate and a reset gate. The update gate determines how much of the previous state should be carried over to the next state. This gate has the ability to completely ignore an input and just use the previous hidden state, if it feels that the input (word) at a certain time step is worthless. The reset gate determines whether or not the previous hidden state should be "forgotten" and reset. These gates work together to try and capture both long and short range dependencies in the input sequence. The attention mechanism in the second pass is implemented using an additional gate that determines whether or not to ignore an input word given the summary from the first pass and the hidden state from the previous word.

On a more philosophical note, a common theme in most of these ideas is that knowing when to ignore inputs is as important as (or, some may argue, more important than) knowing when to remember or utilize inputs.

Practical considerations and implementation details:

- We use the same model to generate summaries for both products. This is done by creating a clone of the model and sharing the weights.
- We use batch normalization (Ioffe et al., 2015) in certain steps to handle internal covariate shifts or in other words, the problem of shifting input distribution to each layer of the model during training.
- We use leaky ReLUs (He et al., 2015a) and ReLUs with outputs clamped between 0 and 1 instead of sigmoid units in the attention gate for the second pass. This produced better results than the tanh and sigmoid units used in the original dynamic memory network paper.
- Adam optimization (Kingma et al., 2014) was used for training since this converged to a much better optimum than stochastic gradient descent. This technique utilizes per parameter learning rates that depend on the gradients for the parameter in the past few iterations.
- We use Caffe - Xavier initialization (Glorot et al., 2010) to set initial weights in most of our layers. This technique takes into account the inverse of the number of input neurons feeding into each neuron and uses this to randomly initialize the weights.

2.4 Boosting accuracy further

We noticed that our model had trouble extracting crucial attributes from the description such as product name and category. The model was not able to place the necessary emphasis on these attributes and extract them correctly. To solve this, we "endowed" the model with this prior knowledge. We extracted these attributes and made the model "read" these separately, by creating dedicated networks for these attributes. These networks communicated with each other at the end of each pass. This was done by concatenating the

summary produced by each of the networks and using this as the memory input to the attention gates.

2.5 Boosting accuracy even further

Since our word vectors were trained on the same training data that we used as inputs for our entity matching model, we let the model fine tune the word vector model while training the product matching model. This was done by backpropagating the gradients at the input layer of the product matching model through the character based word vector model, all the way until the character level embeddings.

2.6 Stage 3 Results

The results of our 5 fold cross validation can be found here. Our precision on the set Y (3k), when trained on a 15k training set, was 95.9% with a recall of 96.3%. This model was used for our stage 3 submission.

2.7 Stage 4 Improvements and Results

Stage 3 contained all optimizations until section 2.4. For our stage 4 submission, we made our product matching model backpropagate through the word vector model as well.

Our precision on the set Y for stage 4 is 96.8% with a recall of 97.1%.

2.8 Results on Home dataset

We applied our model to a dataset containing descriptions of products in the home and garden section. However, in this case our model was not able to achieve as good results as in the case with the electronics dataset. Our precision for this dataset was 84.7% and the recall was 88.5%. Furthermore, our model did not show any sign of converging when we tried backpropagating through the word vector model as mentioned in section 2.5, even though this had helped us achieve a high accuracy in the electronics products dataset. This may be due to the excessive overfitting.

3. Traditional Machine Learning Models

In order to apply traditional ML methods such as decision tree, random forest, SVM, naive bayes, and logistic regression, we need to create features from the data manually. Then the extracted features will be feeded into each ML model in ski-learn for training and prediction.

3.1 Feature Extraction

Features are constructed for each pair of products.

First we filtered out the following categorical attributes by ranking the frequency they appear in the product pairs:

List of Attributes
"Product Type", "Product Segment", "Product Name", "Brand", "Country of Origin: Components", "Category", "Manufacturer", "Color", "Actual Color", "Warranty Length", "Condition", "Type", "Operating System",

"Multipack Indicator", "Battery Type", "Screen Size", "Features", "Number of Batteries", "Hard Drive Capacity", "Processor Type", "RAM Memory", "Memory Capacity", "Processor Speed", "Connector Type", "Processor Core Type", "Release Date", "Material", "Digital Video Formats", "Operating System Required", "RAM Memory Type", "Solid-State Drive", "Wireless", "Video Game Platform", "Monitor Type", "Computer Mouse Included", "Computer Mouse Type", "Keyboard Included", "Desktop Computer Type", "Display Resolution"

Then for each attribute in the list, we computed all the similarity functions and distance functions appeared in the py-stringmatching package of the attribute of the two products in a pair. For distance functions, we divided the result with the sum of the lengths of the attribute of the two products. This is to eliminate the potential bias from the variation of string lengths.

The results of similarity functions and rescaled distance functions are added as features for each product pair. Note that we did not include long description and short description at this step. The default value for similarity function is 1 and the default value for rescaled distance function is 0. This means that in the case of missing attribute, we assumed they are the same.

Next, we took out the numerical attributes: "Assembled Product Length", "Assembled Product Width", "Assembled Product Height" of two products, computed their difference and then rescaled it with the max of the two. The default value here is 0.

We will refer the features computed above as base features.

Then we compared the extracted brands of the two products with the method developed in stage 2. Here we also used the synonym list from stage 2. So if the predicted brands are synonyms, then they are considered the same. This predicted brand feature is 0 if they are the same for the two products, otherwise is 1.

In addition, we constructed the following attributes:

- Numerical Feature in Product Name: Extract all numbers from two product names, and compute their set-based distance. This feature can capture the subtle difference in the products like iPhone 6 vs iPhone 5.
- Refurb Feature: If the word “refurb” appears in any attribute of the product, then the product is considered as refurbished. If one product is refurbished and the other is not, this feature will be 1, otherwise 0.
- Bullet Feature from Long Description: We visualized the data in html tables and found that there are bulleted lists in many long descriptions. The bulleted lists fit well into dictionaries. So we extracted them from the long description and compare the dictionary. If there is difference in values of the same key, then this feature is 1, otherwise 0. The default value of this feature is 0.

Since some of the information extracted from the long description appeared in the attributes as well, so we put these information back into their corresponding attributes, and recomputed the base features. We will refer them as enriched base features.

3.2 Results of ML Methods

We run five methods (decision tree, random forest, SVM, naive bayes, and logistic regression) over different set of features. Parameters of the methods are determined by experiments. If a precision of 96% can be achieved, we chose the parameters which maximize the recall. If 96% precision can not be achieve for the model, we will maximize F1. The best parameter for decision tree is max_depth=2. For random forest, the best choice is n_estimators=1000, max_depth=4. For SVM we found C=100 works best.

The result of Naive Bayes is very bad. With any set of features, its precision won't go above 60%, so we drop it out of comparison. For logistic regression, we could not make precision higher than 90% with any choice of feature set.

With the base features and predicted brands, we have the following results:

Method	Precision	Recall	F1
random forest	0.911	0.832	0.868
decision tree	0.812	0.832	0.820
svm	0.912	0.649	0.751

With enriched base features + Numerical Feature in Product Name + Refurb Feature + Bullet Feature, we have:

Method	Precision	Recall	F1
random forest	0.945	0.836	0.885
decision tree	0.933	0.831	0.877
svm	0.860	0.822	0.841

3.3 Rules

We used three rules to improve the precision of the traditional ML methods:

- If refurb feature=1, predict MISMATCH
- If bullet feature=1, predict MISMATCH
- If numerical features differ a lot (any of the three is greater than 0.6), predict MISMATCH.

These rules are crucial to improve the precision of the random forest and decision tree upto 96%. We dropped SVM since it cannot achieve 96% precision even with the rules.

Method	Precision	Recall	F1
random forest	0.966	0.748	0.836

decision tree	0.980	0.644	0.777

3.4 Discussion about the Result

We did not push this approach further since we already got very good result from the deep learning approach discussed in section 2. One key difficulty of this approach is to extract as much information from the long description as possible. With deep learning, this is handled automatically. Our current approach only looks at bulleted lists. Also we did not apply advanced distance measure to the dictionaries we get from the bulleted lists. This is another potential improve.

3.5 Discussion about py-stringmatching Package

In order to avoid bias coming from length of different attributes, we rescaled the distances computed with py-stringmatching into a number between 0 and 1. It will be convenient to have a similarity function for each distance function. Even though for some distance functions it is not clear what is the max possible value, scaled it down with some upper bound will be still helpful.

4. Future work

One of the drawbacks of using a neural network model is the lack of interpretability. It is difficult to visualize and interpret what rules the model learns to make predictions. More work is needed in this area to understand how to improve models further and also to be able to gain more trust in the model's predictions.

One reason why our model could have done so poorly in the home products dataset is that it seems to be a fundamentally more difficult even for humans. It is more easy to distinguish between two electronics products than it is to distinguish between say two rugs.

Another reason is that there could have been some signals in the electronics dataset (perhaps due to an issue with the way the dataset was created) that made it very easy for the model to figure out which models match. In our brief local analysis of the predictions made by our model (Ribeiro et al., 2016), it seems as though our model places a lot of importance on the "Country of Origin: Components" attribute for many products. Under certain circumstances it seems this attribute gives the model an easy way to determine whether the products match. The best solution to this problem might be to simply use another dataset that does not have such a flaw.

References

Kumar, Ankit, et al. "Ask me anything: Dynamic memory networks for natural language processing." *arXiv preprint arXiv:1506.07285* (2015).

Kim, Yoon, et al. "Character-aware neural language models." *arXiv preprint arXiv:1508.06615* (2015).

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE International Conference on Computer Vision*. 2015a.

Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).

Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin. "" Why Should I Trust You?": Explaining the Predictions of Any Classifier." *arXiv preprint arXiv:1602.04938* (2016).

Chung, Junyoung, et al. "Gated feedback recurrent neural networks." *arXiv preprint arXiv:1502.02367* (2015).

Collobert, Ronan, et al. "Natural language processing (almost) from scratch."*The Journal of Machine Learning Research* 12 (2011): 2493-2537.

Léonard, Nicholas, Sagar Waghmare, and Yang Wang. "rnn: Recurrent Library for Torch." *arXiv preprint arXiv:1511.07889* (2015).

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory."*Neural computation* 9.8 (1997): 1735-1780.

Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *International conference on artificial intelligence and statistics*. 2010.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. "Highway networks." *arXiv preprint arXiv:1505.00387* (2015).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

He, Kaiming, et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 37.9 (2015b): 1904-1916.

He, Kaiming, et al. "Deep Residual Learning for Image Recognition." *arXiv preprint arXiv:1512.03385* (2015c).

Bengio, Yoshua. "Practical recommendations for gradient-based training of deep architectures." *Neural Networks: Tricks of the Trade*. Springer Berlin Heidelberg, 2012. 437-478.

Nielsen, Michael A. "Neural Networks and Deep Learning." *URL: http://neuralnetworksanddeeplearning. com/.(visited: 01.11. 2014)* (2015).

Appleyard, Jeremy, Tomas Kocisky, and Phil Blunsom. "Optimizing Performance of Recurrent Neural Networks on GPUs." *arXiv preprint arXiv:1604.01946* (2016).

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.