Learning to Rank with Attentive Media Attributes

Baldo Faieta
Adobe
San Francisco, CA 94103
bfaieta@adobe.com

Yang (Allie) Yang
Adobe
San Francisco, CA. 94103
yangyan@adobe.com

Abstract

In the context of media search engines where assets have small textual data available, we explore several models that improve the learning to rank use cases. In particular, we propose a model with an attention mechanism that leverages phrase-based attributes to guide the importance of other keyword-based attributes. We train these models with clickthrough data from Adobe Stock search queries and we evaluate them using various ranking metrics. Preliminary results show that our proposed methods have the potential of significantly improve ranking results.

1 Introduction and Background

With the advent of massive proliferation of images and media in general, searching for images has become an important task for search engines. Users expect typing a few words and finding the documents they are looking for. In the case of media, this use case is harder than searching for documents of plain text. Indeed, images, and media in general, have very few attributes associated with them. There is usually some short text like a caption or a title and also other attributes like tags or language used.

By looking at the clickthrough data from the search results, relevance of results can be substantially improved [1]. By using a variety of signals such as image popularity, whether there is a title, the number of tags, etc. and, using techniques like SVM ranking [4] to assign weights to these signals, a search engine can use these signals at query time to boost the results for relevance. Because of the expectation for search engines to return results within 100ms, more expensive approaches to ranking are only possible within a two-stage approach where. First, an initial query search is performed that uses the offline ranking weights, and then a second, more expensive ranking, is done in a window on the top results of the first query using learning to rank methods.

An alternative approach to help users find images is to look into the image itself by identifying what is on the image and let the user query with an existing image to find similar images. Convolutional neural networks (CNN) have been trained to associate a feature vector with the images and use nearest neighbor (NN) search methods [2] to find the most similar images. That is, at query time a CNN feature vector is calculated for the query image and then using NN techniques such as product quantization compare the query feature vector efficiently with pre-trained codes associated with the feature vectors of the images in the corpus.

More recently, a combined approach for learning to rank using deep learned feature vectors (CNNs) has been applied in the domain of text search [3]. Using pre-trained embeddings associated with terms in the query and the documents, separate sentence matrices are constructed where convolutional feature maps are derived to train a network with question and answering data to derive ranking data for text retrieval. In principle, this method can be used in the second stage ranking and in an analogous manner to image similarity use NN methods to rank for the most relevant results.

In the context of media search, the question is whether deep learning learn-to-rank techniques can help for search relevance when using clickthrough data using the few keywords within the query and potentially the very short text associated with the media and additional textual features present such as tags. Here, we describe a set of models exploring this question where we learn representations of the query as well as for the media textual attributes so that when matched with each other we end up with a score that can be used in a second-stage learn-to-rank retrieval. The models we consider go from a simple model where we learn the embeddings of the query and the media attributes to a more complex model where we use the short text associated with the media to guide the representation of the media attributes and come up with a combined deep representation of the media for comparison with the query's.

Even with the two-phase query approach, there is a strict limitation on the amount of computation that can be devoted to evaluating each candidate result together with the query. The actual comparison between the query and the candidate representation has to be very simple and efficient and following the work of NN techniques, we assume that the actual computation per candidate at query time can be at most something equivalent to the cosine distance or a dot product. But, we can spend computationally some effort in deriving a query representation at query time given that it is done just once, and on the candidate result side, we assume that we can pre-calculate offline their representation, so we could build complex models.

2 Models

Figure 2 shows a diagram of the models considered. Our models follow the siamese architecture where the query flows in the left side, while the candidate results flows through the right side. Both sides produce a feature vector that are then compared with each other using the dot product to produce a score. There are 3 models we consider: a simple **baseline** model, the **GRU-Query** model where the query side is more complex and the **Attentive-Attributes** where an attention mechanism is used on the candidate results side.

2.1 Query-side

In the query side, for the simple **baseline** model, we just sum up the embeddings of the terms in the query while for the more complex models, **GRU-Query** and **Attentive-Attributes**, we use a recurrent neural network (RNN) and more specifically a GRU[6], which we feed with embeddings of the query terms. Indeed, we want the model to learn a representation of the query that takes into account the sequence of the terms. For example, we want the network to distinguish a query like "snow white" from that one of "white snow".

2.2 Candidate-results-side

On the candidate results side, for the **baseline** and **GRU-Query** models, we just add up the weighted embeddings associated with the tags. For the **Attentive-Attributes** model, we want to leverage the more targeted semantic coherence of the title (or potentially a caption) to guide (through an attention mechanism similar to [7]) the additional attributes like tags. Tags (as well as other attributes) are not sequential and but contain additional information where not all is very relevant. So, the attention mechanism focuses the importance of the tags based on the more coherent text of the tile.

2.3 Attentive-Attributes Model

In the **Attentive-Attributes** model, we use a GRU for the attributes that have phrases or sentences like the title in Figure 1: "Woman and little girl eating at kitchen". Titles tend to be short but there are still meaningful sequences and we want our model to represent them. Textual attributes like tags, they are usually narrow meaningful terms, but the order is less important. Also, the degree to which each tag is useful is not apparent. In Figure 1 for example, the tags “residential” or “ordinary” are peripherally important. the **Attentive-Attributes** model uses the output states of the title GRUs with an attention mechanism similar to [7] to weigh in each tag. Then, these weighted tags are max pooled to produce an aggregate representation of the tags. We concatenate the title and the tags and use this feature vector to compare with the query. In order to match both representations, we multiply the concatenated vector by an interaction matrix.

More in detail, similarly to [7] and [9], if H is the matrix of hidden states of the title, and J is an interaction matrix between tags and titles and T is the matrix that holds the tags embeddings, then we create a matrix $S = tanh(H\ J\ T^{\top})$ that holds the soft context of the hidden state h_i of the title up to term i to the tag t_j. The context $s_{i,j}$ is a score that says how much state h_i supports tag t_j. Then we apply a column-wise max-pool operation on S which essentially picks the most likely context for tag t_j among all the hidden states h_i in the title. In the example of Figure 1, effectively, tags like “vegetable” and “salad” might get boosted by the phrase “eating at kitchen”. With the max-pooled score s_j, as in [7], we normalize them using a softmax layer into a vector α_j which serves as a weight for tag t_j and then end up with an aggregate weighted vector $\sum_j \alpha_j\ t_j$ representing the tags. This vector is then concatenated with the output of the GRUs corresponding to the title and used as representation for the candidate result.

2.4 Positive-Negative Pathways

Our models are trained using a margin-loss where, as illustrated in Figure 3, the inputs to the model are on the one hand the query on the query side, and on the candidate results side, we have the positive (+) pathway with a stack corresponding to the inputs for the clicked image and the negative pathway (-) corresponding to the impression (i.e., an image that the user saw but did not click). In order for the error to back-propagate correctly, care has to be taken to make sure that the parameters for the stacks of the positive and negative pathways are tied (i.e., the same).

3 Related Work

As mentioned above, we follow the general approach described in [3]. That is, we have two separate flows, one for the query and one for the candidate results. However, in [3], they use a CNN on the sentence matrix induced by the embeddings of the terms in the query and in the document. In our case, that approach makes sense for the query, but not for the candidate result. Also, in [3], the comparison of the query and the document are more complex with a hidden layer between the score and the join layer. That adds computational complexity.

The DRMM model in [8] adds much more interaction between the query and the document and in addition, it has a fixed-length histogram mapping layer before feeding the results into a feed forward matching network and then to score aggregation with a term gating network to derive the final score. We do away from the local interaction between the query and the document and stick with the siamese architecture over the query and the candidate results. However, their term gating network is similar to our attention mechanism in that it helps focus the terms in the document using the terms in the query.

Our model is closer to [9] in that their model uses an RNN (a biLSTM) for both the question and answer and also their attention mechanism is almost the same as ours as well as that one in [7]. In their case, they use the intermediate states of the question RNN to provide context for the answer RNN. In our case, we use the intermediary states of the title RNN to provide context for the tags. We also do a column-wise max-pooling to come up with the tag weights while [9] uses both column-wise and row-wise pooling to come up with weights for both the question and the answer. However, the difference in approach between [9] and our approach is that we keep the interaction of the query side and the candidate result side very simple for query-time computational efficiency, while [9] do not focus on that.

The model in [10], their model LSTM-DSSM derived from earlier work on Deep Structured Semantic Model (DSSM), use LSTMs both for the query and for the document to derive a deeply learned representation that is sensitive to the sequence of the query and the document so that at query time, they can be compared much like our model. Also, like our model, it is trained with click data with a positive pathway (clicked image) and a negative pathway (impression) and the error has to be correctly propagated to the query, while the parameters of the RNN for both of the positive and the negative pathway are tied.

4 Approach

To evaluate the merit of the various models, we trained each model with a large dataset from search queries from Adobe Stock, a service used to license high quality images. For each model, we first conducted a hyper-parameter search on a dataset of 1M samples, and evaluated it using error rate to find the best parameter values. We define the *error rate*, as the ratio of the number of times the clicked image (+) score is lower than impression image score (-) and we use this metric for both the training and validation phase. After having found a good set of hyper-parameters, we use these to train on a larger dataset of 16M samples for each model. Finally, with these trained models, we perform our test evaluation with a different 1M dataset using a different metric: the relative weight calculated by an SVM ranker[4]. This metric is used to evaluate internally ranking features in production of the Adobe Stock service. More details on this is in section 5.1

Because we have a large training dataset, we thought it would be better to train our models together with the query, tags and titles embeddings as opposed to use pre-trained embeddings. We leave the last row of the embedding matrix to be all 0s, so that when summing up the row vectors doesn't affect the sum values. As we mentioned in previous sections, we use max margin for calculating the loss. For the **GRU-Query** and **Attentive-Attributes** models we use dropout and L-2 regularization to prevent overfitting. We also found that gradient norm stayed small, so there is no need for gradient clipping..

5 Experiments

5.1 Dataset

The Adobe Stock service, has over 70M+ images which contain titles, tags, and a myriad other attributes like category etc. The 100M data record collected correspond to user queries, produced over two years ago from the www.fotolia.com[1] service. Each data record is composed of a user query, the title and tags for the clicked image and the title and tags for one of the impressions of the corresponding query. Our dataset had already being pre-processed where terms had already being converted to indices using a 'pivot table', a mapping from about 1.7m words to 40k english terms. This is a limitation on the dataset given that the pivot table was curated rather than learnt. More on this issue will be discussed in section 6.

The 1M and 16M datasets were sampled from the 100M records. During batch training, we pad query, tags, and titles to maximum lengths of 6, 20 and 50 respectively. Their distribution charts can be seen below figure 4, 5 and 6. We believe The maximum pad lengths we choose will help to cover the cases for most training data while not wasting too much training time on tail rare cases. As mentioned in section 4, instead of applying mask at the end of the padded vectors, we just make the last row of embedding matrix to be zero and sum up row vectors to form vector representations of a query, tag or title.

[1] fotolia was later acquired by Adobe Corp.

For the final tests, we use an internally deployed SVM ranker for evaluation. The SVM ranker currently trains on 250 features to calculate an offline relevance score for each image. For each feature, it assigns a weight that effectively indicates the feature importance. We include the scores predicted by our model as an additional 251st feature and train the SVM ranker with the modified dataset to determine the weight assigned to our feature. The larger the weight, the more important the ranker considers it is. The dataset used during this test evaluation comes from search queries taken from www.stock.adobe.com during the month of Oct, 2016, and of 1M data records, roughly corresponds to about 26K queries, sampled from evaluation dataset of 70M with 270K queries.

5.2 Parameter Search

We first conduct hyper-parameter search on a small data set of 1M for training and 50k on dev, sampled from the total 100M as mentioned in 5.1. The final tuned hyper-parameters are listed in Table 1. For the **baseline** and **GRU-Query** models, we did a random parameter search with a fixed range. For the **Attentive-Attributes** model, we did a random search the parameters from a pre-defined lists of values chosen at different scales. All models are run for 10 epochs for each random set.

For the **Attentive-Attributes** model we run about 100+ random search sets. For a given hyper-parameter, we expect to have about 10+ sets runs with a particular value. In order to get a sense of the effect on each hyper-parameter on the model, we averaged across the observations the smallest error during training and dev phases and the epoch where the smallest dev error occurred.

	embed size	**hidden size**	**lr**	**adam beta1**	**dropout**	**margin**	**l2 beta**
baseline	391	n/a	0.00158	0.29	n/a	0.426	n/a
GRU Query	273	167	0.00102	0.811	0.438	0.144	0.961
Attentive-Attributes	512	512	0.00001	0.95	0.5	0.2	0.0001

Table 1: Hyperparameters. embed_size: learnt word embedding size; hidden_size: GRU hidden unit size; lr: learning rate; adam_beta1: the exponential decay rate for the 1st moment estimate; dropout: dropout keeping rate; margin: a constant in loss function; l2_beta: L2 normalization

5.3 Results

Figure 7 and 8 shows how model error rate changes depending on the different learning rate (lr) and margin in the **Attentive-Attributes** model. We conducted this analysis for all the various hyper-parameters though only the margin and lr are shown below. Learning rate we chose for **Attentive-Attributes** model was 0.00001, because at this point the dev error rate is relatively low and the epoch is high showing that it was probably not overfitting with these settings. We want small difference between train and dev error to prevent overfitting, however we also want high epoch showing that the model takes its time to keep on learning. Likewise, margin within 0.2 is a reasonable compromise where the epoch is high with a relatively low dev and training error. Another reason we focused on values that have a larger epoch is that because the **Attentive-Attributes** is a complex model, when being trained with the 16M dataset, we did not want it to get stuck in a local minima too early on. The hyper-parameters shown in Table 1 reflects this compromise of having on average larger epoch number with comparatively lower dev error.

Figure 9 shows the error rate by model and data size. Larger data shows better results. However, we obtain the unexpected results that baseline performs best, based on validation dataset. It looks like the more complex the model, the bigger error we get. We will further discuss the implications of this in section 6..

On SVM ranker production test, we achieve very promising results. Table 2 shows the top 5 ranks for the SVM calculated weight. Again, the larger dataset performs better than the small dataset. In 1M, our weights, 0.239 and 0.27, rank 3rd place out of 251 features; in 16M, our weights, 0.429 and 0.412, rank 1st place. With 16M, However, baseline receives the best weight again. Unfortunately, due to the time constraint and model complexity, we haven't finished SVM testing part for the final model at the submission time of the paper.

Weight Rank	Baseline		GRU	
	1M	16M	1M	16M
1	0.326	**0.429**	0.329	**0.412**
2	0.311	0.321	0.309	0.286
3	**0.239**	0.301	**0.27**	0.286
4	0.226	0.213	0.217	0.214
5	0.21	0.203	0.199	0.212

Table2: SVM top five weights out of 251 features

5.4 Demo

For the **baseline**, we loaded the data into a search service to visualize the effects of the rescoring results. The current system has a hard time to deal with queries that match the query terms but may have alternative partial meanings. Figure 10 shows a query where this is the case. The query "gold bowl" fails because titles like "Gold fish bowl", "Golden retriever with a bowl", "Gold colored wheat bowl" partially match the query terms. When we turn the rescoring resulting from our trained model, we can see that the results look much better.

6 Conclusion

All three of our models improve the ranking score of Adobe Stock, which already produces good results even though the training data was from an older (2 year old) version of the system. All of them reduce the error rate and produce large SVM weight and a good rank. A caveat though is that we only tested the SVM ranker with a smaller 1M dataset out of a larger 70M data set.

As mentioned in 5.1, our training data used a 'pivot table', a curated dictionary, to translate non-english words to english and then look up in vocabulary table. The pivot table does just a keyword lookup , which introduces errors in translation, thus creating noise in the data. Also, the training and test data are not quite the same quality.

Even though the baseline model performed better, we suspect that the more complicated models would fare much better with bigger datasets. Due to limitation on time and Azure resources, we only trained with the 16M dataset, but we believe that longer training times and using the bigger 100M dataset may prove that the more complicated models do perform best.

Going forward, we will use most update data for training so that train and test data are of same quality. We will enlarge the train and test data size to 100M+ and 70M respectively. 70M is the current production test size. We want also to mix the textual features with the visual features to create a multi-modal model. Lastly, we believe that using pre-trained embeddings that benefit from semantic similarities across languages will help us avoid the problems with the pivot table and potentially help with the more complex models where they don't have to focus on training the embeddings but more on the RNNs. We believe we should be able to significantly improve the ranking following some of these steps.

Acknowledgments

The work described here is based on earlier work by Matt Hoffman while he was working at

Adobe.

References

[1] Joachims, T. , M.C. (2002) Optimizing Search Engines using Clickthrough Data. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133-142. ACM.

[2] Jegou, H., Douze, M., Schmid, C. (2010) Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* (Volume: 33, Issue: 1, Jan. 2011). IEEE.

[3] Severyn, A., Moschitti, A. (2015) Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks. *SIGIR '15 Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 373-382. ACM.

[4] Joachims, T. (2009) Support Vector Machine for Ranking. *Cornell University*.
https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

[5] Kingma et al. (2014). Adam: A Method for Stochastic Optimization. *3rd International Conference for Learning Representations, San Diego.*

[6] Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y. (2014) On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111. Association for Computational Linguistics.

[7] B., Bahdanau, D., Cho, K., Bengio, Y. (2015) Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR 2015*.

[8] Guo, J., Fan, Y., Ai, Q., Croft, W. B., (2016) A Deep Relevance Matching Model for Ad-hoc Retrieval, *CIKM'16*, ACM

[9] dos Santos, C., Tan, M., Xiang, B., Zhou, B. (2016) Attentive Pooling Networks. *arXiv:1602.03609v1 [cs.CL] 11 Feb 2016*

[10] Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., Ward, R., (2015) Semantic Modelling with Long-Short-Term-Memory for Information Retrieval , *arXiv:1412.6629v3 [cs:IR] 27 Feb 2015*

Appendix

Figure 1: Image from Adobe Stock with title *"Woman and little girl eating at kitchen"*, and tags *"woman" "little" "girl" "eating" "vegetable" "salad" "residential" "kitchen" "happy" "together" "ordinary" "people"* ...

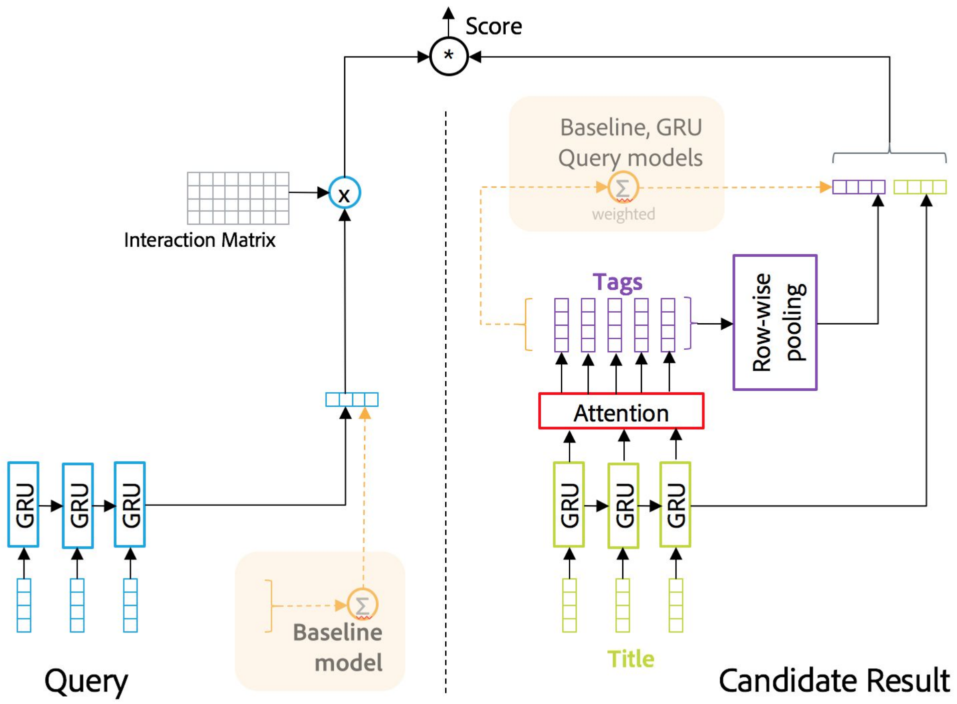


Figure 2: baseline, GRU Query and Attentive Attributes Models

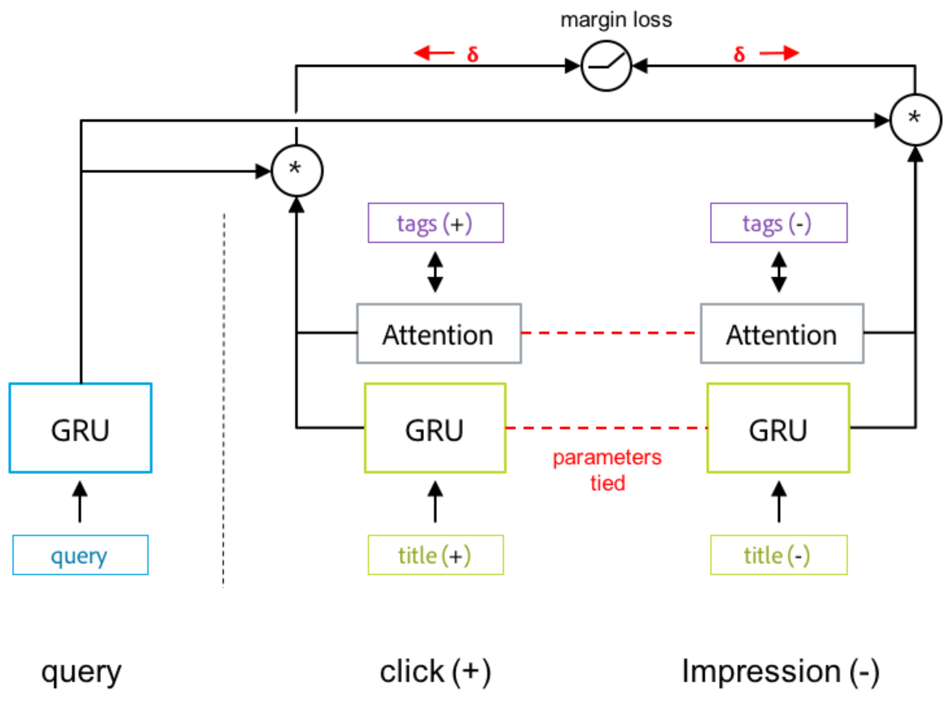


Figure 3: training with positive (+) and negative (-) samples

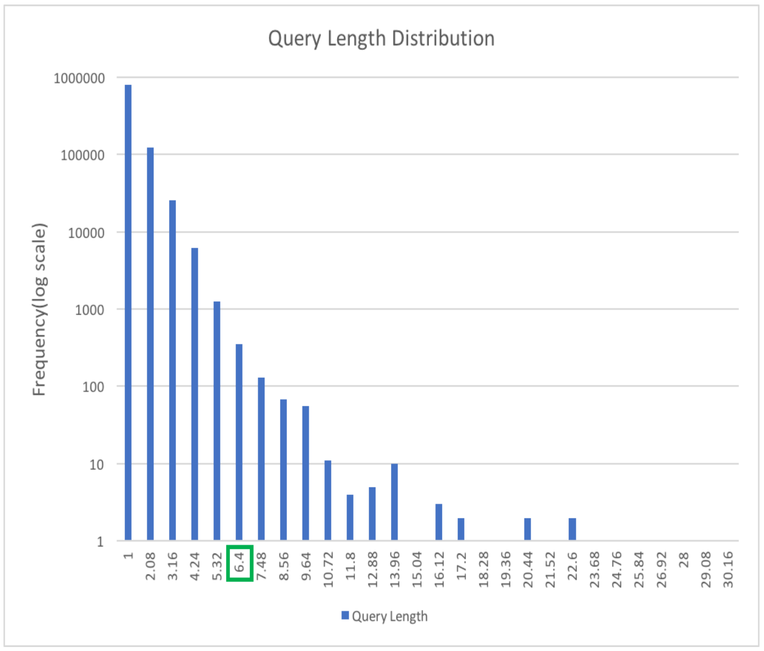


Figure 4: Query Length

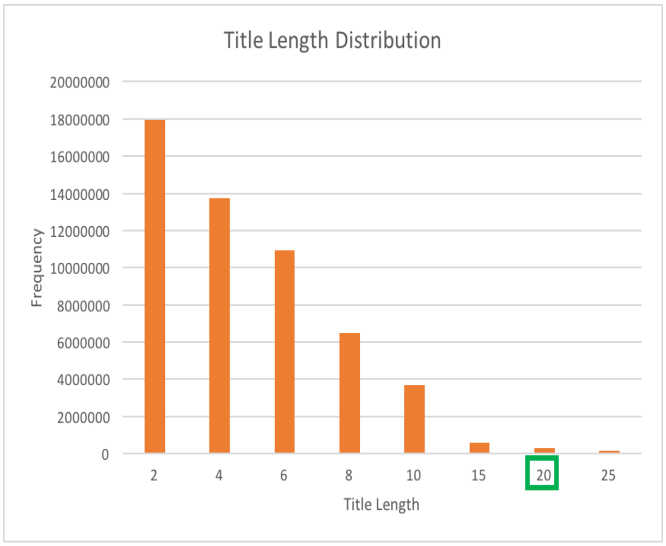


Figure 5: Title Length

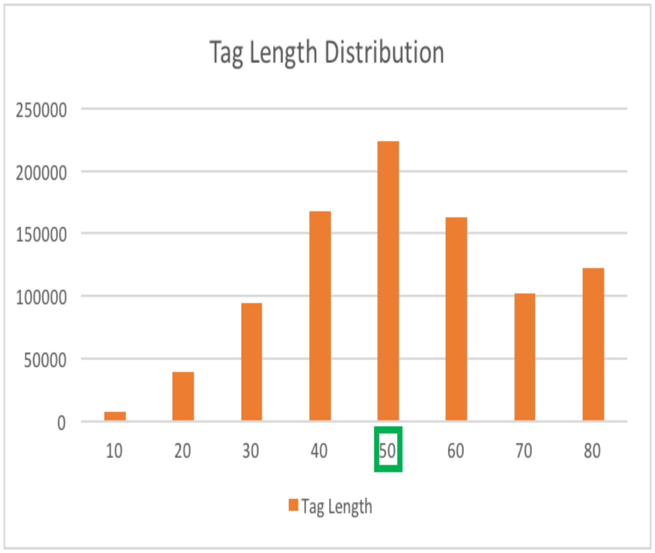


Figure 6: Tag Length

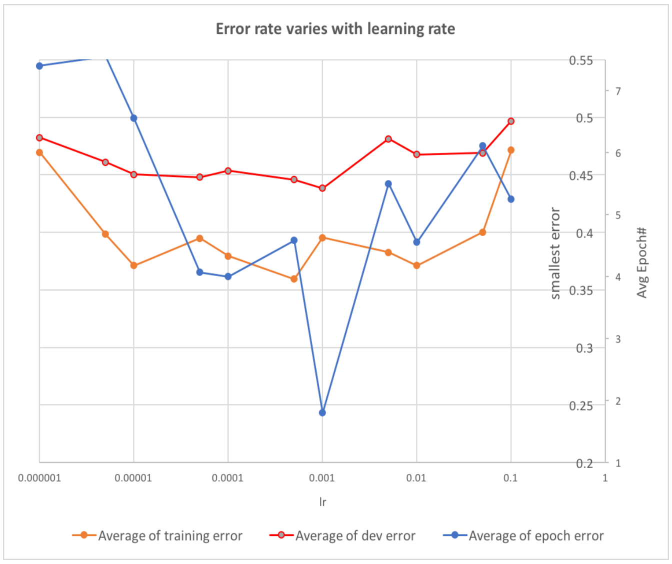


Figure 7: Error rate with learning rate

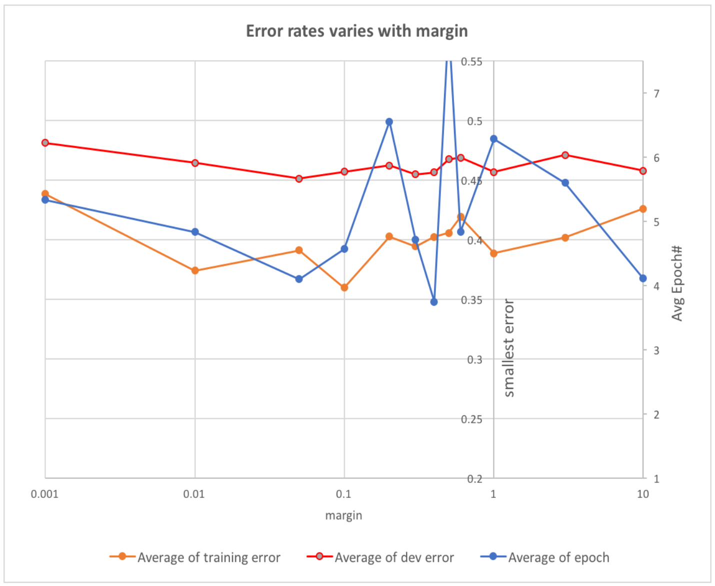


Figure 8: Error rate with margin

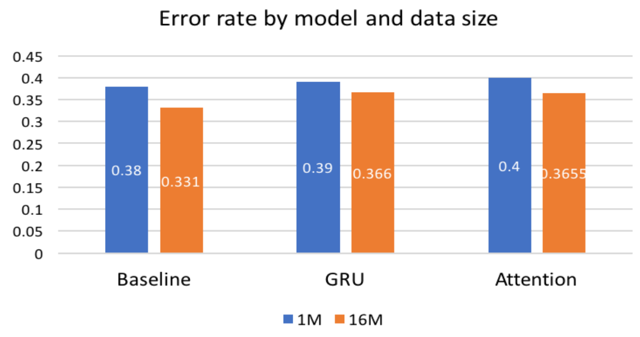
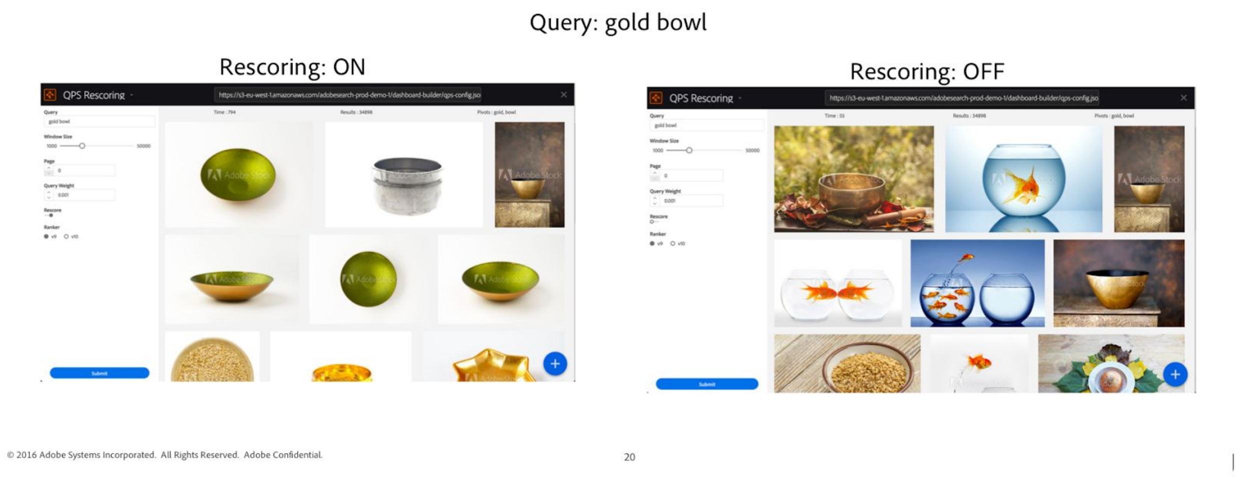


Figure 9: Error rate by model


Query: gold bowl
Rescoring: ON
Rescoring: OFF
QPS Rescoring
Submit
© 2016 Adobe Systems Incorporated. All Rights Reserved. Adobe Confidential.
20

Rescoring: ON

https://s3-eu-west-1.amazonaws.com/adobesearch-prod-demo-1/dashboard-builder/qps-config.jso

Time : 794 Results : 34898 Pivots : gold, bowl

Query: gold bowl
Window Size: 1000 – 50000
Page: 0
Query Weight: 0.001
Rescore
Ranker: [✓ v9] v10

Rescoring: OFF

https://s3-eu-west-1.amazonaws.com/adobesearch-prod-demo-1/dashboard-builder/qps-config.jso

Time : 55 Results : 34898 Pivots : gold, bowl

Query: gold bowl
Window Size: 1000 – 50000
Page: 0
Query Weight: 0.001
Rescore
Ranker: [✓ v9] v10

Figure 10: demo of query “golden bowl”