A Neural Autoencoder Approach for Document Ranking and Query Refinement in Pharmacogenomic Information Retrieval

Jonas Pfeiffer$^{1,2}$ Samuel Broscheit$^1$ Rainer Gemulla$^1$ Mathias Göschl$^2$

$^1$University of Mannheim, Mannheim, Germany
$^2$Molecular Health GmbH, Heidelberg, Germany

{lastname}@informatik.uni-mannheim.de
Mathias.Goesch@molecularhealth.com
Jonas.Pfeiffer.90@Gmail.com

Abstract

In this study, we investigate learning-to-rank and query refinement approaches for information retrieval in the pharmacogenomic domain. The goal is to improve the information retrieval process of biomedical curators, who manually build knowledge bases for personalized medicine. We study how to exploit the relationships between genes, variants, drugs, diseases and outcomes as features for document ranking and query refinement. For a supervised approach, we are faced with a small amount of annotated data and a large amount of unannotated data. Therefore, we explore ways to use a neural document auto-encoder in a semi-supervised approach. We show that a combination of established algorithms, feature-engineering and a neural auto-encoder model yield promising results in this setting.

1 Introduction

Personalized medicine strives to relate genomic detail to patient phenotypic conditions (such as disease, adverse reactions to treatment) and to assess the effectiveness of available treatment options (Brunicardi et al., 2011). For computer-assisted decision making, knowledge bases need to be compiled from published scientific evidence. They describe biomarker relationships between key entity types: Disease, Protein/Gene, Variant/Mutation, Drug, and Patient Outcome (Outcome) (Manolio, 2010). While automated information extraction has been applied to simple relationships — such as Drug-Drug (Asada et al., 2017) or Protein-Protein (Peng and Lu, 2017); (Peng et al., 2015); (Li et al., 2017) interaction — with adequate precision and recall, clinically actionable biomarkers need to satisfy rigorous quality criteria set by physicians and therefore call upon manual data curation by domain experts.

To ascertain the timeliness of information, curators are faced with the labor-intensive task to identify relevant articles in a steadily growing flow of publications (Lee et al., 2018). In our scenario, curators iteratively refine search queries in an electronic library, such as PubMed. The information the curators search for, are biomarker-facts in the form of \{Gene(s) - Variant(s) - Drug(s) - Disease(s) - Outcome\}. For example, a curator starts with a query consisting of a single gene, e.g. \(q_1 = \{PIK3CA\}\), and receives a set of documents \(D_1\). After examining \(D_1\), the curator identifies the variants \(H1047R\) and \(E545K\), which yields queries \(q_2 = \{PIK3CA, H1047R\}\) and \(q_3 = \{PIK3CA, E545K\}\) that lead to \(D_2\) and \(D_3\). As soon as studies are found that contain the entities in a biomarker relationship, the entities and the studies are entered into the knowledge base. This process is then repeated until, theoretically, all published literature regarding the gene \(PIK3CA\) has been screened.

Our goal is to reduce the amount of documents which domain experts need to examine. To achieve this, an information retrieval system should rank documents high that are relevant to the query and should facilitate the identification of relevant entities to refine the query.

Classic approaches for document ranking, like tf-idf (Luhn, 1957); (Spärck Jones, 1972), or bm25 (Robertson and Zaragoza, 2009), and, for example, the Relevance Model (Lavrenko and Croft, 2001) for query refinement are established techniques in this setting. They are known to be robust and do not require data for training. However, as they are based on a bag-of-words model (BOW),

1https://www.ncbi.nlm.nih.gov/pubmed
they cannot represent a semantic relationship of entities in a document. This, for example, yields search results with highly ranked review articles that only list query terms, without the desired relationship between them. Therefore, we investigate approaches that model the semantic relationships between biomarker entities. This can either be addressed by combining BOW with rule-based filtering, or by supervised learning, i.e. learning-to-rank (LTR).

Our goal is, to tailor document ranking and query refinement to the task of the curator. This means that a document ranking model should assign a high rank to a document that contains the query entities in a biomarker relationship. A query refinement model should suggest additional query terms, i.e. biomarker entities, to the curator that are relevant to the current query. Given the complexity of entity relationships and the high variety of textual realizations this requires either effective feature engineering, or large amounts of training data for a supervised approach. The in-house data set of Molecular Health consists of 5833 labeled biomarker-facts, and 24 million unlabeled text documents from PubMed. Therefore, a good solution is to exploit the large amount of unlabeled data in a semi-supervised approach. Li et al. (2015) have shown that a neural auto-encoder with LSTMs (Hochreiter and Schmidhuber, 1997) can encode the syntactics and semantics of a text in a dense vector representation. We show that this representation can be effectively used as a feature for semi-supervised learning-to-rank and query refinement.

In this paper, we describe a feature engineering approach and a semi-supervised approach. In our experiments we show that the two approaches are, in comparison, almost on par in terms of performance and even improve in a joint model. In Section 2 we describe the neural auto-encoder, and then proceed in Section 3 to describe our models for document ranking and in Section 4 the models for query refinement.

2 Neural Auto-Encoder

In this study, we use an unsupervised method to encode text into a dense vector representation. Our goal is to investigate if we can use this representation as an encoding of the relations between biomarker entities.

Following Sutskever et al. (2014), Dai and Le (2015), and Li et al. (2015) we implemented a text auto-encoder with a Sequence-to-Sequence approach. In this model an encoder $\text{Enc}$ produces a vector representation $v = \text{Enc}(d)$ of an input document $d = [w_1, w_2, \ldots, w_n]$, with $w_i$ being word embedding representations (Mikolov et al., 2013). This dense representation $v$ is then fed to a decoder $\text{Dec}$, that tries to reconstruct the original input, i.e. $\hat{d} = \text{Dec}(v)$. During training we minimize error $(\hat{d}, d)$. After training we only use the $\text{Enc}(d)$ to encode the text. We want to explore if we can use $\text{Enc}$ to encode the documents and the query. We will use the output of the document encoder $\text{Enc}$ as features for a document ranking model and for a query refinement model.

3 Document Ranking

Information retrieval systems rank documents in the order that is estimated to be most useful to a user query by assigning a numeric score to each document. Our pipeline for document ranking is depicted in Figure 1: Given a query $q$, we first retrieve a set of documents $D_q$ that contain all of the query terms. Then, we compute a representation $\text{rep}_q(q)$ for the query $q$, and a representation $\text{rep}_d(d)$ for each document $d \in D_q$. Finally, we compute the score with a ranker model $\text{score}_{\text{rank}}$.

For $\text{rep}_d$ we need to find a representation for an arbitrary number of entity-type combinations, because a fact can consist of e.g. 3 Genes, 4 Variants, 1 Drug, 0 Diseases and 0 Outcomes. In the following, we describe several of the settings for $\text{rep}_q(q)$, $\text{rep}_d(d)$ and the ranker model.

![Figure 1: Document Ranking](image)
3.1 Bag-of-Words Models

We have implemented two commonly used BOW models tf-idf and bm25. For these models the text representations $rep_q(q)$ and $rep_d(d)$ is the vector space model.

3.2 Learning-to-Rank Models

For the learning-to-rank models, we chose a multilayer perceptron (MLP) as the scoring function $score_{rank}$. In the following we explain how $rep_q(q)$ and $rep_d(d)$ are computed.

Feature Engineering Model  We created a set of basic features: encoding the frequency of entity types, distance features between entity types, and context words of entities. In this model, features are query dependent and are computed on-demand by a feature function $f(q,d)$.

The algorithm to compute the distance feature is as follows: Given query $q$ with entities $e \in q$ and document $d = [w_1, w_2, \ldots, w_n]$, with $w$ being words in the document. Let $type(e)$ be the function that yields the entity type, f.ex. $type(e) = Gene$. Then, if $e_i, e_j \in q$ and there exists a $w_k = e_i, w_l = e_j$ then we add $|l - k|$ to the bucket of \{type($e_i$), type($e_j$)\}. To summarize the collected distances we compute $\min()$, $\max()$, $mean()$ and $std()$ over all collected distances for each bucket separately.

For the context words feature, we collected in a prior step a list of the top 20 words for each \{type($e_i$), type($e_j$)\} bucket, i.e. we collect words that are between $w_k = e_i$ and $w_l = e_j$ if $|k - l| < 10$. We remove stop words, special characters and numbers from this list and also manually remove words using domain knowledge. The top 20 of remaining words for each \{type($e_i$), type($e_j$)\} bucket are used as boolean indicator features.

Auto-Encoder Model  In this model we use the auto-encoder from Section 2 to encode the query and the document. The input to the score function is the element-wise product, denoted by $\circ$, of the query encoding $rep_q(q) = Enc(q) = q$ and the document encoding $rep_d(d) = Enc(d) = d$:

$$score_{rank}(d, q) = MLP(d \circ q)$$  \hspace{1cm} (1)

To encode the queries we compose a pseudo text using the query terms. The input to the autoencoder $Enc$ are the word embeddings of the pseudo text for $rep_q(q)$ and the word embeddings of the document terms for $rep_d(d)$.

4 Query Refinement

Query refinement finds additional terms for the initial query $q$ that better describe the information need of the user (Nallapati and Shah, 2006). In our approach we follow Cao et al. (2008), in which the ranked documents $D_q$ are used as pseudo relevance feedback. Our goal is to suggest relevant entities $e$ that are contained in $D_q$ and that are in a biomarker relationship to $q$. Therefore, we define a scoring function $score_{ref}$ for ranking of candidate entities $e$ to the query $q$ with respect to the retrieved document set $D_q$. See Figure 2 for a sketch of the query refinement pipeline. In the following, we describe several of the scoring functions.

4.1 Bag-of-Words Models

We implemented the two classic query refinement models the Rocchio algorithm (Rocchio and Salton, 1965) and Relevance Model.

4.2 Auto-Encoder Model

In this model, we also try to exploit the auto-encoding of the query. The idea is as follows: (i) Given a list of documents and their scores $D_q = [(d_1, s_1), (d_2, s_2), \ldots, (d_n, s_n)]$ for a query $q$ from the previous step, we use the ranking score $s_1$ as pseudo-relevance feedback to create a pseudo document $rep_q(D, s_1) = \sum_i d_i \hat{s}_i = \hat{d}$. The scores $s$ are normalized so that they are non-negative and $\sum_i \hat{s}_i = 1$, see Appendix A.1. (ii) From all entities $e_i \in D_q \backslash q$ we generate new query encoding $rep_q(q \cup e_i) = Enc(q \cup e_i) = \hat{q}_{e_i}$ (iii) We rank the entities based on the pseudo document using the scoring function

$$score_{ref}(\hat{q}_{e_i}, \hat{d}) = MLP(\hat{d} \circ \hat{q}_{e_i})$$  \hspace{1cm} (2)
i.e. we propose those entities as a query refinement that agree with the most relevant documents.

5 Experiments

In this section, we first explain our evaluation strategy to assess the performance of the respective models for document ranking and query refinement. Subsequently, we describe the settings for the data and the results of the experiments that we have conducted.

5.1 Evaluation Protocol

Document ranking models are evaluated by their ability to rank relevant documents higher than irrelevant documents. Query refinement models are evaluated both, by their ability to rank relevant query terms high, and by the recall of retrieved relevant documents when the query automatically is refined by the 1st, 2nd and 3rd proposed query term. We evaluate our models using mean average precision (MAP) (Manning et al., 2008, Chapter 11) and normalized discounted cumulative gain (nDCG) (Järvelin and Kekäläinen, 2000).

For both the document ranking and query refinement approach we interpret a biomarker-fact as a perfect query and the corresponding papers as the true-positive (or relevant) papers associated with this query. In this way, we use the curated facts as document level annotation for our approach. Because we want to assist the iterative approach of curators in which they refine an initially broad query to ever narrower searches, we need to create valid partial queries and associated relevant documents to mimic this procedure. Therefore, we generate sub-queries, which are partials of the facts. We generated two data sets: one for document ranking and one for query refinement. For document ranking, we generated all distinct subsets of the facts. For query refinement, we defined the eliminated entities (of the sub-query generation process) as true-positive refinement terms. For both data sets, we use all associated documents, of the original biomarker-fact, as true-positive relevant documents.

5.2 Data

Unlabeled Data As unlabeled data we use ~24 Million abstracts of PubMed. To automatically annotate PubMed abstracts with disambiguated biomarker entities, we use a tool set that has been developed together with biomedical curators. It employs ”ProMiner” (Hanisch et al., 2005) for Protein/Gene and Disease entity recognition and regular expression based text scanning using synonyms from “DrugBank” and “PubChem” for the identification of Drugs and manually edited regular expressions, relating to “HGVS” standards, to retrieve Variants. We restricted the PubMed documents to include at least one entity of type Gene, Drug and Disease leaving us with 2.7 Million documents. Additionally we replaced the text of every disambiguated entity with its id.

Labeled Data As labeled data we use a knowledge base that contains a set of 5833 hand curated \{Gene(s) - Variant(s) - Drug(s) - Disease(s) - Outcome\} biomarker-facts and their associated papers that domain experts extracted from ~1200 full-text documents. We only keep facts in which the disambiguated entities are fully represented in the available abstracts. This restricted our data set to a set of 1181 distinct facts. The 4 top curated genes are EGFR (29%), BRAF (13%), KRAS (8%), and PIK3CA (5%).

Cross Validation To exploit all of our labeled data for training and testing we do 4-fold cross-validation. Because in our scenario a curator starts with an initial entity of type Gene we have generated our validation and test sets based on Genes, instead of randomly sampling facts. This also guarantees us to never have the same sub-query included in the training, validation and test set. In total we have built 12 different splits of our data set basing the validation and test set each on a different gene. The respective training sets are built with all remaining facts that do not include the validation and test gene. Statistics can be found in Table 1.

5.3 Training of Embeddings and Auto-Encoder

The training data for the embeddings and the auto-encoder are the PubMed abstracts described in the previous Section 5.2. We trained the embeddings with Skip-Gram. For the vocabulary, we keep the top 100k most frequent words, while making sure all known entities are included. We use a window size of 10 and train the embeddings for 17
5.4 Document Ranking

In this section, we describe the document ranking models, their training and then discuss the results of their evaluation.

Models We evaluate the BOW models (tf-idf, bm25) (Section 3.1) and the two LTR models using feature engineering (feat) and the auto-encoded features (auto-rank) (Section 3.2). We also evaluate an additional set of models to investigate if maybe simpler solutions can be competitive. See Table 2 for an overview over all ranking models.

(a.) A simpler solution than learning a MLP for a score function is to compute the similarity between \( q = Enc(q) \) and the document encoding \( d = Enc(d) \). Therefore, we use the cosine similarity as scoring function between the vector representations \( q \) and \( d \) (auto/cosine).

(b.) Instead of encoding the documents and queries with the auto-encoder, we encode the documents and queries based on their tf-idf weighted embeddings, i.e. \( q = \sum tfidf(q_i) * w_{q_i} \). Similarly to the auto-rank model, the input to the classifier MLP is the element-wise product of the query encoding and the document encoding (emb).

(c.) Due to promising results of the auto-rank model, bm25, and the feat model, we also tested combinations of them. We tested the concatenation of the the bm25 score with the auto-rank features (auto-rank + bm25) as well as the concatenation of feat with the auto-rank features (auto-rank + feat).

Training We train our models with Adam (Kingma and Ba, 2014) and tune the initial learning rate, the other parameters are kept default of TensorFlow. We use a pairwise hinge loss (Chen et al., 2009) and compare relevant documents with irrelevant documents.

The ranking score function is parameterized by a MLP for which the number of layers is a hyper-parameter which is tuned using grid-search. The input layer size is based on the number of input features. To limit the total number of parameters, we decrease the layer size while going deeper, i.e. layer \( i \) has size \( l_i = \frac{b \cdot (l_{i-1})}{b + 1} \cdot |u| \), with \( b \) the depth of the network, \( |u| \) the number of input features. For activation functions between layers we use ReLU (Glorot et al., 2011).

We employ grid search over the hyper-parameters: dropout: \([0.3, 0.2, 0.1, 0.0]\), number of layers: \([1, 2, 3, 4]\), learning rates for Adam: \([0.0005, 0.001]\), batch size: \([40, 60]\), max 400 epochs. We conducted hyper-parameter tuning for each model and validation/test split separately. The best parameters for the models using the auto-encoded features were: 1 layer, dropout \( p \in [0.3, 0.2] \) with a batch size of 60 and learning rate 0.0005. The feat models were best with 1-2 layers, dropout 0.0 and a learning rate at 0.001.

The BOW models as well as auto/cosine were only computed for the respective validation and test sets.

Results In Table 3 we have listed the average MAP and nDCG scores of the test sets. The tf-idf model is outperformed by most of the other models. However, bm25, which additionally takes the length of a document into account, performs very well. tf-idf and bm25 have the major benefit of fast computation.

The feat model slightly outperforms the auto-rank model.
Table 2: Query and document representation for ranking models

<table>
<thead>
<tr>
<th>Model</th>
<th>(rep_{q}(q))</th>
<th>(rep_{d}(d))</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf-idf</td>
<td>BOW</td>
<td>BOW</td>
<td>dot product</td>
</tr>
<tr>
<td>bm25</td>
<td>BOW</td>
<td>BOW + doc length</td>
<td>bm25</td>
</tr>
<tr>
<td>emb</td>
<td>(q = \text{tf-idf} \cdot \text{BOW} \cdot \text{w})</td>
<td>(q = \text{tf-idf} \cdot \text{BOW} \cdot \text{w})</td>
<td>(\text{MLP}(q \odot d))</td>
</tr>
<tr>
<td>feat</td>
<td>(f(q,d))</td>
<td>(f(q,d))</td>
<td>(\text{MLP}(f(q, d)))</td>
</tr>
<tr>
<td>auto/cosine</td>
<td>(q = \text{Enc}(q))</td>
<td>(d = \text{Enc}(d))</td>
<td>(\text{cos} (\text{Enc}(q), \text{Enc}(d)))</td>
</tr>
<tr>
<td>auto</td>
<td>(q = \text{Enc}(q))</td>
<td>(d = \text{Enc}(d))</td>
<td>(\text{MLP}(q \odot d))</td>
</tr>
<tr>
<td>auto + bm25</td>
<td>(q = \text{Enc}(q), \text{BOW})</td>
<td>(d = \text{Enc}(d), \text{BOW} + \text{doc length})</td>
<td>(\text{MLP}(q \odot d, \text{bm25}))</td>
</tr>
<tr>
<td>auto + feat</td>
<td>(q = \text{Enc}(q))</td>
<td>(f(q,d))</td>
<td>(d = \text{Enc}(d))</td>
</tr>
</tbody>
</table>

Table 3: Test Scores Document Ranking

<table>
<thead>
<tr>
<th>Test</th>
<th>Metric</th>
<th>tf-idf</th>
<th>bm25</th>
<th>emb</th>
<th>feat</th>
<th>auto/ cosine</th>
<th>auto-rank</th>
<th>auto-rank + bm25</th>
<th>auto-rank + feat</th>
</tr>
</thead>
<tbody>
<tr>
<td>EGFR</td>
<td>MAP</td>
<td>0.289</td>
<td>0.632</td>
<td>0.310</td>
<td>0.575</td>
<td>0.054</td>
<td>0.545</td>
<td>0.588</td>
<td>0.699</td>
</tr>
<tr>
<td></td>
<td>nDCG</td>
<td>0.424</td>
<td>0.728</td>
<td>0.460</td>
<td>0.695</td>
<td>0.129</td>
<td>0.653</td>
<td>0.716</td>
<td>0.810</td>
</tr>
<tr>
<td>KRAS</td>
<td>MAP</td>
<td>0.327</td>
<td>0.610</td>
<td>0.466</td>
<td>0.609</td>
<td>0.058</td>
<td>0.575</td>
<td>0.774</td>
<td>0.820</td>
</tr>
<tr>
<td></td>
<td>nDCG</td>
<td>0.456</td>
<td>0.723</td>
<td>0.592</td>
<td>0.712</td>
<td>0.145</td>
<td>0.688</td>
<td>0.867</td>
<td>0.914</td>
</tr>
<tr>
<td>BRAF</td>
<td>MAP</td>
<td>0.342</td>
<td>0.656</td>
<td>0.427</td>
<td>0.704</td>
<td>0.063</td>
<td>0.563</td>
<td>0.702</td>
<td>0.812</td>
</tr>
<tr>
<td></td>
<td>nDCG</td>
<td>0.480</td>
<td>0.751</td>
<td>0.572</td>
<td>0.802</td>
<td>0.163</td>
<td>0.671</td>
<td>0.820</td>
<td>0.901</td>
</tr>
<tr>
<td>PIK3CA</td>
<td>MAP</td>
<td>0.341</td>
<td>0.633</td>
<td>0.486</td>
<td>0.625</td>
<td>0.079</td>
<td>0.541</td>
<td>0.779</td>
<td>0.810</td>
</tr>
<tr>
<td></td>
<td>nDCG</td>
<td>0.473</td>
<td>0.729</td>
<td>0.617</td>
<td>0.718</td>
<td>0.171</td>
<td>0.656</td>
<td>0.859</td>
<td>0.895</td>
</tr>
</tbody>
</table>

Figure 3: Correlation of Document Ranking Models

The cosine similarity of a query and a document (auto/cos) does not yield a good result. This shows that the auto-encoder has learned many features, most of which do not correlate with our task. We also find that \(\text{emb}\) does not yield an equal performance to auto-rank. The combination of the auto-rank + feat model is slightly better than the auto-rank + bm25 model, both of which have the overall best performance. This shows, that the auto-encoder learns something orthogonal to term frequency and document length. The best model with respect to document ranking is the auto-rank + feat model.

In Figure 3 we show the correlation between the different models. Interestingly, the \(\text{bm25}\) and the \(\text{feat}\) strongly correlate. However, the scores of \(\text{bm25}\) do not correlate with the scores of the combination of auto-rank and \(\text{bm25}\). This indicates, that the model does not primarily learn to use the \(\text{bm25}\) score but also focuses on the the auto-encoded representation. This underlines the hypothesis that the auto-encoder is able to represent latent features of the relationship of the query terms in the document.

Influence of the Data It is interesting to see that the learned models do not perform well for the EGFR set. The reason for this might be that testing on it reduces the amount of training data substantially, as EGFR is the best curated gene and thus the largest split of the data set.

In a manual error analysis we compared the
rankings of four of the best models (bm25, feat, auto-rank, and auto-rank + bm25). We observe cases where the auto-rank model is unable to detect, when similar entities are used, i.e. entities like Neoplasm and Colorectal Neoplasm. In these cases the bm25 helps, as it treats different words as different features. However, both bm25 and the feat models rank reviews high, that only list query terms. For example, when executing the query \{BRAF, PIK3CA, H1047R\}, these models rank survey articles high (i.e. (Li et al., 2016); (Janku et al., 2012); (Chen et al., 2014)).

The auto-rank model on the other hand ranks those document high, for which each entities are listed in a semantic relationship (i.e. (Falchook et al., 2013)).

5.5 Query Refinement

In this section we describe our training approaches for query refinement and discuss our findings. The pseudo relevance feedback for the query refinement is based on the ranked documents from the previous query. For our experiments we chose the second best document ranker (auto-rank + bm25) from the previous experiments, because our prototype implementation for auto-rank + feat was computationally too expensive.

Models

We combined both the auto-encoder features with the candidate terms of the respective BOW models (auto-ref + rocch + relev). In order to identify if the good results of this combination are due to the BOW models, or if the auto-encoded features have an effect, we trained a MLP with the same amount of parameters, but only use the features of the two BOW models as input (rocchio + relev).

Training

For training, we use the same settings for query refinement as for document ranking and again use a pairwise hinge loss. Here we compare entities that occur in the facts with randomly sampled entities which occur in the retrieved documents.

Due to limitations in time we were only able to test our query refinement models on one validation/test split. We chose to use the split data set of genes KRA5 and PIK3CA for validation and testing respectively. We have restricted our models to only regard the top 50 ranked documents for refinement.

Results

To evaluate the ranking of entity terms we have computed nDCG@10, nDCG@100 and MAP, see Table 4 for the results. We also compute Recall@k of relevant documents for automatically refined queries using the 1st, 2nd and 3rd ranked entities. The scores can be found in Table 5.

Tables 4 and 5 show that the Relevance Model outperforms the Rocchio algorithm in every aspect. Both models outperform the auto-encoder approach (auto-ref). We suspect that summing over the encodings distorts the individual features too much for a correct extraction of relevant entities to be possible.

The combination of all three models (auto-ref + rocchio + relevance) outperforms the other models in most cases. Especially the performance for ranking of entity terms is increased using the auto-encoded features. However, it is interesting to see that the rocchio + relevance model outperforms the recall for second and third best terms. This indicates that for user-evaluated term suggestions, the inclusion of the auto-encoded features is advisable. For automatic query refinement however, in average, this is not the case.

### Table 6: Refinement Terms for Query \{PIK3CA\}

<table>
<thead>
<tr>
<th>Variants</th>
<th>Diseases</th>
<th>Drugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1047R</td>
<td>Color. Neop.</td>
<td>Lapatinib</td>
</tr>
<tr>
<td>V600E</td>
<td>Liposarcoma</td>
<td>Mitomycin</td>
</tr>
<tr>
<td>T790M</td>
<td>Adenocarcin.</td>
<td>Linsitinib</td>
</tr>
<tr>
<td>E545K</td>
<td>Glioblastoma</td>
<td>Dactolisib</td>
</tr>
<tr>
<td>E542K</td>
<td>Stomach Neop.</td>
<td>Pictrelisib</td>
</tr>
</tbody>
</table>

Query Refinement Example

In Table 6 we show the top ranked entities of type Variants, Diseases and Drugs for the query \{PIK3CA\}. While the diseases and the drugs are all relevant, V600E and T790M are in fact not variants of the gene PIK3CA.

However, when refining the query \{PIK3CA, V600E, BRAF, H1047R, Dabrafenib\}, the top ranked diseases are [Melanoma, Neoplasms, Carcinoma Non Small Cell Lung (CNSCL), Thyroid Neoplasms, Colorectal Neoplasms]. Using Melanoma for refinement, retrieves the top ranked paper (Falchook et al., 2013) which perfectly includes all these entities in a biomarker relationship.


### Table 4: Ranked Entity Scores for KRAS Validation and PIK3CA Testing

<table>
<thead>
<tr>
<th>Metrics</th>
<th>rocchio</th>
<th>relevance model</th>
<th>auto-ref</th>
<th>rocchio + relevance</th>
<th>auto-ref + rocchio + relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDCG@10</td>
<td>0.232</td>
<td>0.274</td>
<td>0.195</td>
<td>0.341</td>
<td>0.464</td>
</tr>
<tr>
<td>nDCG@100</td>
<td>0.360</td>
<td>0.397</td>
<td>0.329</td>
<td>0.439</td>
<td>0.536</td>
</tr>
<tr>
<td>MAP</td>
<td>0.182</td>
<td>0.223</td>
<td>0.156</td>
<td>0.270</td>
<td>0.386</td>
</tr>
</tbody>
</table>

### Table 5: Refinement Recall Scores for KRAS Validation and PIK3CA Testing

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Top n-th Entity</th>
<th>rocchio</th>
<th>relevance model</th>
<th>auto-ref</th>
<th>rocchio + relevance</th>
<th>auto-ref + rocchio + relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall@10</td>
<td>1</td>
<td>0.594</td>
<td>0.603</td>
<td>0.272</td>
<td>0.574</td>
<td>0.696</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.535</td>
<td>0.561</td>
<td>0.339</td>
<td>0.580</td>
<td>0.522</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.533</td>
<td>0.544</td>
<td>0.366</td>
<td>0.555</td>
<td>0.458</td>
</tr>
<tr>
<td>Recall@40</td>
<td>1</td>
<td>0.683</td>
<td>0.691</td>
<td>0.307</td>
<td>0.680</td>
<td>0.779</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.603</td>
<td>0.633</td>
<td>0.374</td>
<td>0.649</td>
<td>0.586</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.610</td>
<td>0.623</td>
<td>0.402</td>
<td>0.626</td>
<td>0.522</td>
</tr>
</tbody>
</table>

### 6 Related Work

The focus of research in this domain has primarily targeted the extraction of entity relations. Peng and Lu (2017), Peng et al. (2015), and Li et al. (2017) try to extract Protein-Protein relationships. Asada et al. (2017) try to extract Drug-Drug interaction and Lee et al. (2018) target the extraction of Mutation-Gene and Mutation-Drug relations. Jameson (2017) have derived a document ranking approach for PubMed documents using word embeddings trained on all PubMed documents. Xu et al. (2017) propose using auto-encoders on the vector-space model in a supervised setting for information retrieval and show that it improves performance. The quality of biomedical word embeddings was investigated by Th et al. (2015) and Chiu et al. (2016). Dogan et al. (2017) have developed an open source data set to which we would like to adapt our approach. Sheikhshab et al. (2016) have developed a novel approach for tagging genes which we would like to explore.

Palangi et al. (2016) use LSTMs to encode the query and document and use the cosine similarity together with the click-through data as features for ranking in a supervised approach. Cao et al. (2008) define a distance based feature-engineered supervised learning approach to identify good expansion terms. They try to elaborate if the selected terms for expansion are useful for information retrieval by identifying if the terms are actually related to the initial query. Nogueira and Cho (2017) have introduced a reinforcement learning approach for query refinement using logging data. They learn a representation of the text and the query using RNNs and CNNs and reinforce the end result based on recall of a recurrently expanded query. Sordoni et al. (2015) have developed a query reformulation model based on sequences of user queries. They have used a Sequence-to-Sequence model using RNNs to encode and decode queries of a user.

### 7 Conclusion

We have considered several approaches for document ranking and query refinement by investigating classic models, feature engineering and, due to the large amount of unlabeled data, a semi-supervised approach using a neural auto-encoder. Leveraging the large amounts of unlabeled data to learn an auto-encoder on text documents yields semantically descriptive features that make subsequent document ranking and query refinement feasible. The combination with BOW features increases the performance substantially, which for our experiments, outputs the best results, for both document ranking and query refinement.

We were able to achieve promising results, however, there is a wide range of Sequence-to-Sequence architectures and text encoding strategies, therefore, we expect that there is room for improvement.
References


Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. 2015. A Hierarchical Neural Autoencoder for Paragraphs and Documents.


A Supplemental Material

A.1 Normalizing Document Scores for Query Refinement

Because we used a hinge loss instead of cross-entropy loss in the ranking model, we cannot interpret the scores $s$ of the ranker as logits. While we do not know the magnitude of the ranker score, we do, however, expect the scores to be positive for relevant documents. If however many documents have scores below zero, this should also be regarded. Based on this, we have defined a normalization setting of the document scores:

$$s_{min} = \min(\min(s), 0.0)$$  \hspace{1cm} (3)

$$\hat{s} = \frac{s - s_{min}}{\sum |s| \frac{s}{s_{min}}}$$  \hspace{1cm} (4)