Fast Query Expansion for an Accounting Corpus using Sub-word Embeddings

Hrishikesh V. Ganu
Intuit India Development Center
Bangalore, India
hrishikeshvganu@gmail.com

Viswa Datha P.
mail@vishwa.be

Abstract

We present early results from a system under development which uses sub-word embeddings for query expansion in the presence of mis-spelled words and other aberrations. We work for a company which creates accounting software and the end goal is to improve customer experience when they search for help on our "Customer Care" portal. Our customers use colloquial language, non-standard acronyms and sometimes misspell words when they use our Search portal or interact over other channels. However, our Knowledge Base has curated content which leverages technical terms and is in language which is quite formal. This results in the answer not being retrieved even though the answer might actually be present in the documentation (as assessed by a human). We address this problem by creating equivalence classes of words with similar meanings (with the additional property that the mappings to these equivalence classes are robust to mis-spellings) using sub-word embeddings and then use them to fine tune a Search index to improve recall.

1 Introduction

Accounting and taxation is a complex domain-especially for small businesses who might not have the necessary accounting skills but yet need to be compliant with regulations. Hence consumers of accounting software frequently seek advice both about accounting as well as about the product itself.

During an audit process when we started to analyze customers' queries with the help of internal experts, we realised that for a significant number of queries the answers were already available but not retrieved by the search engine because it relies only on keyword based search.

This is primarily due to the following reasons:

1. Self employed and small business owners use colloquial language and terms like *I didn't receive the money customer XYZ owes to me* instead of *I didn't receive my receivables from XYX*

2. Even when customers use accounting terms there are misspellings or structural variants (Form1040 vs Form1040-ES)

1.1 Previous Work

While there have been earlier approaches which deal with these problems through methods like lemmatization, fuzzy matching etc. details of which are given in (Gormley and Tong, 2015), we wanted a method that could be fine-tuned to our dataset without creating hand-crafted rules. Recent progress in creating distributional representations of words has found applications in Information Retrieval (IR) due to work by Zamani and Croft (2017), Korpusik et al. (2017) and Cao and Lu (2017) among others. Some of these models provide the query as an input to a neural network at prediction time while others rely on using the embeddings for query expansion. Our work is in-line with some of these earlier efforts but with a goal of creating an end-end working system that is robust to mis-spellings and other aberrations around the composition of words. Thus in this paper we focus on how we integrated sub-word embeddings with Search systems to create a real-life retrieval system which is currently under development. We demonstrate how this can solve a very practical problem around Information Retrieval from accounting/taxation related corpora.

2 Approach

Our basic approach is to create a search index to enable searching for either the exact word

Proceedings of the Second Workshop on Subword/Character LEvel Models, pages 61–65
New Orleans, Louisiana, June 6, 2018.

or it's synonyms if they are present in the corpus/vocabulary. In addition our approach is able to map mis-spelled (and hence OOV) words to their "correct" versions on the fly at query time. It involves the following steps:

1. Our expert Customer Care Agents (CCA) create a hand-curated list of "important" words including named entities, actions that customers perform with our products etc.

2. We generate word and sub-word embeddings from our own data and then use these embeddings to create nearest neighbors for the important words. The procedure is similar to that used by Bojanowski et al. (2016) so we don't reproduce it here. Hyperparameter settings that were varied by us are mentioned in Section 3.3.

3. We use the "synonym contraction" mechanism of Elasticsearch to update the index with "synonyms". Since this is a standard procedure we refer the reader to Gormley and Tong (2015) for details. During search, the query is also "analyzed" using the same "analyzer". This can be seen as a kind of query reformulation where $word$ is replaced by $\mathbf{OR}(word, synonym_1, synonym_2...)$. Note that for Out of Vocabulary (OOV) words at query time we get "synonyms" by relying on the embeddings of the sub-words, following the procedure in Bojanowski et al. (2016).

3 Experiments

In this section we discuss the overall objective of the experiment and then mention details about the dataset, pre-processing, training and hyperparameter tuning.

3.1 Objective

As mentioned in Section 2, we wish to understand if an Elasticsearch "Synonyms" file based on sub-word embeddings can lead to a higher recall during search compared to a "Synonyms" file based purely on word embeddings (without sub-word embeddings) especially in the presence of mis-spellings and other perturbations. See Figure 1 for a high level overview of how the "Synonyms" files are created.

Algorithm 1 Offline: Create Nearest Neighbours

Require: Set: RootWords ▷ Hand-crafted set of the core/important words
Require: Hashmap H: RootWord→ ListofWords
Require: Set: V ▷ Vocabulary
Require: Integer K ▷ # neighbours to retrieve
Require:
  function NN(rootWord,V, K)
    return $\{w|w \in \text{K\_NN}(\text{rootWord}) \cap V\}$
  end function
  function POPULATENN(H,Rootwords,V,K)
    for rootWord ∈ RootWords **do**
      ▷ Insert word and it's neighbours into H
      H[rootWord]= NN(rootWord, V, K)
        ▷ Remove assigned words from V
      V ← V \ Set(NN(rootWord,V,K))
    end for
  end function
Ensure: H ▷ Hashmap with root words as keys and K-Nearest Neighbours as values

3.2 Dataset and Pre-processing

We realised that the dataset for validating the benefit from this system had to have the following properties:

1. For each query we had to know the matching answer. This was to be used as ground truth.

2. We wanted the answer to be relevant to the matching query but did not want the words in the query to be a subset of words in the answer. This is because had there been good word overlap there wouldn't be a need for query expansion.

After failing to find a well-researched corpus around accounting we decided to use (IRS, 2017), which is a manual written by the Internal Revenue Service (IRS), USA to provide information about personal income tax to taxpayers. While we have also performed experiments on our proprietary knowledge base, for this paper we chose (IRS, 2017) because it is a widely used document and is also available in the public domain. This will enable the larger NLP community to verify and expand upon our findings. Figure 2 shows the layout of a typical page in books like (IRS, 2017). Our basic idea was to use perturbed versions of "headings" (see Section 3.2.1) as queries (perturbed to mimic mis-spellings/typos from real

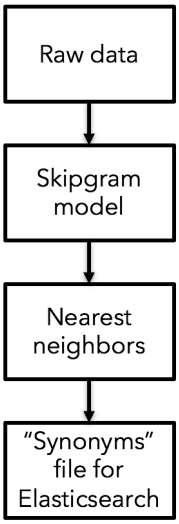

Figure 1: Starting with raw data from (IRS, 2017), a skipgram model (see Section 3.3) was trained, followed by identifying and indexing 10 nearest neighbors per word. This set of nearest neighbors was then provided to Elasticsearch (ES) as a "Synonyms" file which was internally incorporated into the ES index.

users) and to use content in the "body" as documents. Since several downstream models depend on tokenization, we wish to state that for all experiments in this paper we tokenized words using whitespace and using punctuation symbols.

3.2.1 Mis-spelling synthesizer

Our objective here is to compare sub-word embeddings with word embeddings and understand how the robustness to small character level perturbations affects the final recall after search. Since "headings" in books are well formatted and don't have mis-spellings we had to either synthetically generate mis-spelled words or collect statistics around how users mis-spell words in real life. We describe both approaches below.

1. **Synthetic word-level perturbations**: We assume that a word w is a sequence of k characters indexed as c_i so that $w = \{c_1, c_2, .., c_i, .., c_k\}$ and a sentence S is a sequence of n words, indexed as w_j. We select 10% of the queries uniformly at random and create perturbed versions of them. Then for each such query we synthesize perturbed/mis-spelled versions of words by choosing one word uniformly at random (say w_j) to perturb from each sentence and then substitute a random selection of $\lceil 0.20|w_j| \rceil$ adjacent characters by random alphanumeric characters different from the original.

2. **Mis-spelling statistics learned from user data**: Ideally we wanted to generate perturbed versions of all queries from real-world user data. However because we had limited access to human agents [1], we were able to get only word-level perturbations as against full query level-perturbations from human editors. We followed a process where we first picked the most frequent 200 words out of 903 distinct words forming the vocabulary for queries . Let's call the set of these words as set T. For each such word $w_i \in T$ we asked 5 human annotators to type in these words at a targeted pace of 33 words per minute [2] to simulate the pace at which users type in the real world. This allowed us to collect the set of typed versions S_i of each word w_i (which includes mis-spelled versions as well as the correct word itself) and thus compute a distribution $D_i = Pr(x|\text{correct word is } w_i)$ where $x \in S_i$ for each w_i. Finally, we used these distributions to perturb queries by sampling from D_i for each word in the query that occurred in the top 200. Note that this method does not create true query-level perturbations since whether a word is mis-spelled or not might depend on factors like length of the query, presence of other words etc. Here we look at word mis-spellings when they are typed out in isolation.

3.3 Training skipgram model

A skipgram model was trained on the entire corpus from (IRS, 2017) (using original, unperturbed headings). While in a conventional ML setting this procedure (of using the entire dataset for training without holding out a test sample) might be inappropriate, in our case the main objective is to understand the retrieval systems' robustness to mis-spellings and other character level perturbations. In our setup, such perturbations are not present anywhere in the corpus which is clean and free of all mis-spellings. This means that training the

[1] We performed this experiment based on suggestions from reviewers and hence had less than a week to gather and process input from human editors

[2] The average pace for typing is 33 words per minute based on some studies. See https://en.wikipedia.org/wiki/Typing

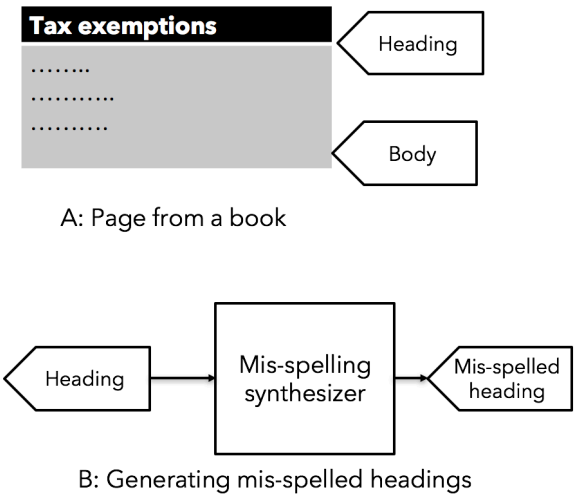

Figure 2: "A: Page from a book" illustrates how a page in a book consists of a heading and some body of text beneath it. This raw corpus was used for training sub-word skipgram models.
"B: Generating mis-spelled headings"" illustrates how we generate misspelled versions of words in "headings" by passing them through a "Mis-spelling synthesizer" (See Section 3.2.1). Mis-spelled headings were used only during search and not for training the skipgram model.

skipgram model on the entire corpus does not prevent us from drawing valid conclusions about the generalization ability of the system as far as robustness to mis-spelled words is concerned. We largely followed the process employed by (Bojanowski et al., 2016) and also leveraged their open-source code [3]. Additionally, because we wanted to retrieve nearest neighbors, we experimented with different settings for the number of neighbors retrieved and sub-word sizes. We relied on internal domain experts to help us determine the configuration with the best synonyms. Details of the experiments and the best configuration chosen are given in Table 1

4 Results

To achieve the objective mentioned in Section 3.1 and in-line with details in Section 3.2, we sent the same set of 805 queries (in our case "headings" perturbed through Section 3.2.1 are queries) to two different Elasticsearch instances: 1) an ES instance having a "Synonyms" file derived from word-level embeddings, called $\mathbf{ES_{base}}$ and 2) an instance having a "Synonym" file derived from sub-word embeddings, called $\mathbf{ES_{sub-word}}$ (see Figure 3).

[3] https://github.com/facebookresearch/fastText

Sub-word size	# neigh-bours	Comments
3-6	10	Good quality neighbours. This is the best configuration
2-8	10	Quality worsened
3-6	20	Irrelevant synonyms
2-4	10	Some non-sensical words in synonyms

Table 1: Hyperparameter tuning for generating embeddings

4.1 Metrics

We use following notation and metrics:

- $correct_{\mathbf{ES_{base}}}$: number of questions for which $\mathbf{ES_{base}}$ returned correct results. This is denoted as "# correct baseline" as per the legend in Figure 3
- $correct_{\mathbf{ES_{sub-word}}}$: number of questions for which $\mathbf{ES_{sub-word}}$ returned correct results. This is denoted as "# correct sub-word" as per the legend in Figure 3
- Recall Ratio (x 100)= $100\frac{correct_{\mathbf{ES_{sub-word}}}}{correct_{\mathbf{ES_{base}}}}$

In Figure 3 we report results based on queries generated using the *"Synthetic word-level perturbations"* method in Section 3.2.1. The primary result from this dataset and from some of our experiments on internal proprietary corpora indicate that the Recall Ratio (x 100) metric varies from 110 to 114 depending on how many top results are retrieved and other parameters. This means that we get between a 10% to 14% lift [4] in the number of questions for which correct answers are retrieved when using sub-word embeddings vs when using word embeddings.

Another insight from Figure 3 is that the increase in recall does not come for free. For $K =$

[4] Following a suggestion from an anonymous reviewer we also collected statistics on word-level errors by human users and used them to generate perturbed queries (See "Misspellings statistics learned from user data" in Section 3.2.1). The lift was 8% with this more realistic dataset but we could perturb only the top 200 words due to limited availability of editors. Note that this dataset also has some synthetic elements.

10, $\mathbf{ES_{base}}$ gives an empty response 30 times compared with 6 times for $\mathbf{ES_{sub-word}}$. Note that in Figure 3 the total of #correct baseline and #null baseline does not equal total number of queries because for some queries the result set was not empty but contained all "wrong" answers. Similar connotation holds for the corresponding sub-word versions.

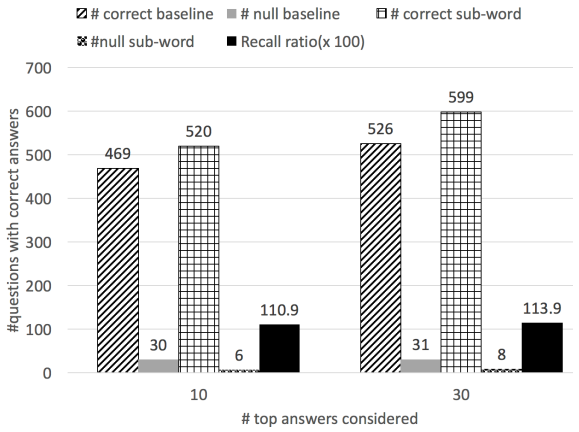

Figure 3: Comparison of the Elasticsearch index using sub-word embeddings, called $\mathbf{ES_{sub-word}}$ with the one using word level embeddings, called $\mathbf{ES_{base}}$. X-axis indicates how many results (say K) from Elasticsearch are considered. E.g: if $K = 10$, we consider only the top 10 results. Y-axis denotes for how many of the 805 queries fired, we received the correct answer in the top K. Because queries are just the perturbed headings, note that for this dataset we assume there's only 1 correct answer per query-the one that appears beneath the heading in (IRS, 2017).

5 Discussion

To test the hypothesis that sub-word embeddings are more robust to spelling error and other perturbations we conducted an experiment with search retrieval as the end goal.

- Through a comparison of sub-word and word level embeddings on a dataset (IRS, 2017) in the public domain, we have demonstrated that using sub-word embeddings for creating the "Synonyms" file for Elasticsearch indeed leads to better recall.

- Since we work in industry we also wish to emphasize that our solution has several features which make it attractive from a practical perspective. In particular, because our solution integrates well with IR systems like ES or Solr, we have very low latency (50ms) compared to query expansion systems utilizing ML model predictions at query time.

- One of the limitations of this study is that while we have demonstrated a system which increases recall through query expansion, we need to complement it with a layer which can re-rank the retrieved results in a more meaningful way. The default ranking module used by ES does not work well for our use case. Adding such a layer will allow us to measure and track ranking based metrics (like DCG) once the solution goes live and we start receiving user feedback.

Acknowledgments

We thank the two anonymous reviewers for their suggestions to improve the paper in terms of both content and style. Their suggestions, led us to conduct some quick experiments with human editors on how people mis-type words and we have modified the paper to reflect these new results.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Shaosheng Cao and Wei Lu. 2017. Improving word embeddings with convolutional feature learning and subword information. In *AAAI*, pages 3144–3151.

Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The Definitive Guide*, 1st edition. O'Reilly Media, Inc.

IRS. 2017. *Your Federal Income Tax*. Department of Treasury, Internal Revenue Service, USA.

Mandy Korpusik, Zachary Collins, and James Glass. 2017. Character-based embedding models and reranking strategies for understanding natural language meal descriptions. *Proc. Interspeech 2017*, pages 3320–3324.

Hamed Zamani and W. Bruce Croft. 2017. Relevance-based word embedding. *CoRR*, abs/1705.03556.

A Supplemental Material

At the following url, we provide examples of the nearest neighbors generated offline for the synonyms file as well as neighbors generated for OOV words at query time. https://goo.gl/3oQ2xF