Personalized Online Spell Correction for Personal Search

Jai Gupta*, Zhen Qin*, Michael Bendersky, Donald Metzler
Google Inc.
Mountain View, CA
{jaigupta,zhenqin,bemike,metzler}@google.com

ABSTRACT

Spell correction is a must-have feature for any modern search engine in applications such as web or e-commerce search. Typical spell correction solutions used in production systems consist of large indexed lookup tables based on a global model trained across many users over a large scale web corpus or a query log.

For search over personal corpora, such as email, this global solution is not sufficient, as it ignores the user's personal lexicon. Without personalization, global spelling fails to correct tail queries drawn from a user's own, often idiosyncratic, lexicon. Personalization using existing algorithms is difficult due to resource constraints and unavailability of sufficient data to build per-user models.

In this work, we propose a simple and effective personalized spell correction solution that augments existing global solutions for search over private corpora. Our event driven spell correction candidate generation method is specifically designed with personalization as the key construct. Our novel spell correction and query completion algorithms do not require complex model training and is highly efficient. The proposed solution has shown over 30% click-through rate gain on affected queries when evaluated against a range of strong commercial personal search baselines - Google's Gmail, Drive, and Calendar search production systems.

CCS CONCEPTS

• **Information systems → Information retrieval**.

KEYWORDS

Spell correction, personalization, personal search

ACM Reference Format:
Jai Gupta*, Zhen Qin*, Michael Bendersky, Donald Metzler. 2019. Personalized Online Spell Correction for Personal Search. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3308558.3313706

1 INTRODUCTION

Misspelling is common in real-world information retrieval systems. More than 10% of search engine queries are misspelled [8]. This rate is even higher for tail queries, for which there is more than a 20% misspelling rate [3]. Misspellings are not just limited to unintentional typing errors, but are often a result of the challenges imposed by the spelling itself. For example, due to factors such as ambiguous word breaks, phonetically similar words, introduction of new words (e.g. project names), and inconsistent spelling rules [1], remembering the correct spelling can be surprisingly difficult even for native speakers.

For these reasons, spelling correction is essential in modern information retrieval systems today. By providing high quality spell correction capabilities, search engines can significantly reduce the amount of effort required for users to find their desired results.

The traditional spell correction approach, referred to as *offline spell correction*, is provided only after the entire query is typed and passed to the search engine (e.g. after a user presses "Enter"). Such a system assumes that any query sent to the search engine has been completely typed by the user. The modern approach, referred to as *online spell correction* [10], provides spell correction and query completion as the user is typing. It is enabled for almost all popular web search services nowadays. In this case, most of the queries under consideration are incomplete. See Fig. 1 for an illustration.

The algorithm presented in this paper is particularly designed for online spell correction systems, and specifically for personal search scenarios. Providing online spell corrections for personal search is remarkably complicated primarily due to the large and diverse vocabulary of private corpora. Furthermore, the stringent latency requirements for online spell correction systems adds to the complexity and renders most of the alternative algorithms impractical for large-scale use.

Personal search, including email search [2, 5, 24], desktop search [11], and most recently on-device search [18], has recently attracted a considerable amount of attention from the information retrieval community. The key difference between personal and public search (e.g. web search) is that users, in the personal search scenario, have access only to their own private document corpora (e.g. emails,

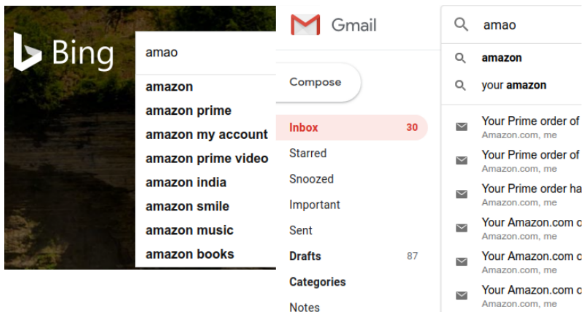


Figure 1: Online spell correction and query completion while the user types on-the-fly (Left) provides suggestions only, (Right) provides as suggestions as well as results.

*Equal contribution.

files, or mobile application data). This poses several new challenges for existing spell correction approaches: (1) A global lexicon may not work well or even hurt performance as each user may have a very unique private corpus. (2) Traditional machine learning-based models are difficult to train for personalization due to relatively small-sized personal corpus, sparse user interactions, and resource constraints. In many real-world cases, training a model directly from private data may not even be allowed due to privacy restrictions. (3) Tail-queries, which are difficult to handle using traditional spell correction systems, become even more problematic, since for personal search many frequently used private terms for a particular user are tail-queries from a global perspective. (4) Due to the lack of quality control, there could be more misspellings in private data itself (e.g. a typo in an email title). This is contrary to the rationale behind existing spell correction systems which build models over correctly spelled words.

This paper introduces an effective and efficient training-free spell correction solution for personal search. It can either be used independently or to complement existing global spell correction systems. We describe the spell correction candidate generation algorithm that provides a personalized lexicon created from a user's recent activities. We provide some details on how to realize such a system under real-world constraints (e.g. privacy and computing resources). We then describe a novel and simple algorithm augmenting the Levenshtein distance based approach for unified spell correction and query completion that allows serving millions of users in a real-time online spell correction system. We build and test our system using real-world online experiments conducted on millions of users for three of the world's largest commercial private corpora (Google's GMail, Drive, and Calendar), and show unanimously positive performance for each corpus across several languages, compared against strong production baselines with spell corrections learned using public web search data.

The remainder of the paper is organized as follows. We start with related works in Section 2. An overview of our system is presented in Section 3. Section 4 outlines candidate generation setup followed by detailed discussion on the spell correction and query completion algorithms in Section 5. Finally, evaluations from our experiments are presented in Section 6.

2 RELATED WORK

Most recent work on spell correction generally focuses on public search queries. Related methods usually produce a global model for all users. Also, a large amount of historical user interactions are normally needed for learning-based approaches. Such approaches typically cannot be applied to personal search settings, as the private data (e.g. search queries, corpora) available is often many orders of magnitude smaller.

For traditional offline spell correction (e.g. "did you mean"), Cucerzan and Brill [8] proposed an iterative approach that transforms unlikely queries into more likely variants based on web query logs. Chen *et al.*[7] explored web search results (queries and their top-ranked candidates) to improve existing spell correction models. Some methods [15] depend on user feedback using an existing spell correction system, which can have cold-start problems and bias concerns (e.g. position bias by the existing system). Sun *et al.*[23] learned phrase-based spell correction models from clickthrough data. Hasan *et al.*[16] collected similar user queries using user session and time information in which users "correct themselves," and uses Levenshtein distance [19] as the similarity criterion. Recently, approaches based on deep learning have become popular, such as those using sequence-to-sequence learning [13]. All of these approaches, especially deep models, typically require large amounts of user interactions or labeled data which does not scale to fully personalized spell correction and personal search scenarios.

A special less data intensive case is word embedding [21] based methods where pre-trained embeddings can be used. However, existing methods primarily use embeddings trained on a large public corpus in a user agnostic manner (e.g. Word2Vec [20] is trained over news articles). Some methods are designed in an *ad hoc* manner for specific language pairs [13, 14]. Our simple approach works well across several languages.

Online spell correction has drawn more interest in the research community [6, 9, 10] and real-world applications (e.g. Google and Bing web search engines). These systems may provide search results, but the incoming queries are often incomplete. Hence, spell correction and query completion are essential components. Chaudhuri and Kaushik [6] made use of edit distance for spell correction followed by a fuzzy search over database records to find completions. Our work unifies spell correction and query completion in an efficient augmented edit distance method. Huizhong and Bo-June [10] trained a Markov *n*-gram transformation model and adapted A^* search with pruning for efficiency. However, their method is specific to data-intense public corpora since it can only be applied to previously observed queries and requires a large amount of training data. Our approach does not require historical queries (which could be sparse) and complex model training.

More broadly, personalized search, including personalized query completion, has also been extensively explored [4, 22] on public corpora. Existing approaches are typically segment-based (e.g. using gender as a feature) while our work is specifically focusing on individual-level private corpora search. Some works utilize user's past search activities (e.g. queries) as signals, which could be sparse for many users. We are not aware of any prior work that focuses on individual-level spell correction in personal search, where personal lexicon is ubiquitous and user search activities are sparse, both of which pose challenges to existing personalized search methods.

3 OVERVIEW

A high level overview of our online spell correction system is present in Fig. 2, depicting the *candidate generation* and the *spell correction and query completion* components. Each user has their own personal lexicon constructed only from their set of private corpora. The lexicon can be updated periodically or in real-time (e.g. when a new document is created). The private user lexicon is small and highly relevant, and does not need any sophisticated data structure, language model or error model trained for that user.

4 CANDIDATE GENERATION

The candidate generation component is responsible for creating a personalized lexicon for every user represented as a set of contextually relevant *n*-grams. Generating these *n*-grams consists of

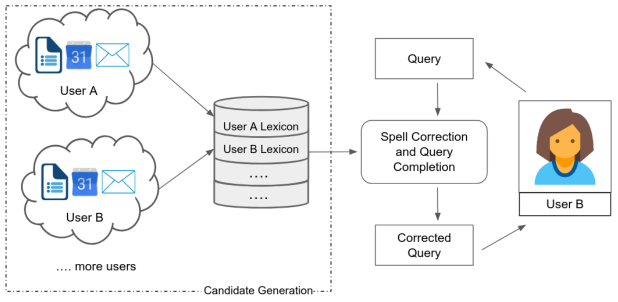


Figure 2: System Overview: Candidates from the personalized lexicon generator passed to spell correction and query completion component.

the following steps: 1) data ingestion, 2) candidate extraction, and 3) candidate update under constraints. Below, we discuss each of these steps in more detail providing general guidelines that can be useful for other applications.

4.1 Data Ingestion

We limit the input data for users to their private corpus. There are a couple of reasons for this design choice: 1) Users will only have access to their private data in personal search, so information from other users decreases the signal-to-noise ratio. 2) Privacy is critical in real-world systems, and non-aggregated data from other users can leak information. 3) By limiting candidates to the user's own set of corpora, the candidate space of n-grams for evaluation is trimmed to a significantly smaller (but highly relevant) set. Note that the user does not need to be the owner of a document, shared documents are taken into account as well.

4.2 Candidate Extraction

The lexicon of a user is composed of n-grams. We use three sources of data that are combined to form a single lexicon per user: 1) Email subjects and the email addresses of senders and recipients, 2) Drive document titles and the email addresses of the owners, and 3) Calendar event titles and the email addresses of the event participants. We found that titles (ignoring bodies) are simpler to manage and have high signal-to-noise ratio for spelling. Note that users can still search document bodies. Also, we keep email addresses as they are frequent query terms.

The snippets collected in the previous step are decomposed into candidate n-grams by standard techniques including tokenization and stopword removal. Additionally, we apply normalization techniques including lowercasing and lemmatization. Note that if documents are retrieved for a spell corrected query (see Fig. 1 right), these normalizations must be consistent with the normalization used when indexing the documents.

4.3 Candidate Update Under Constraints

Maintaining a personal lexicon for every user has many real-world constraints. For example, due to data retention policies, the system may be required to delete any candidate derived from deleted documents. Also, the system is bounded by space constraints as it stores a personal lexicon for every user. This is especially critical for real-world search engines that serve many millions of users. Additionally, a small but high-quality lexicon allows fast online spell correction by iterating over candidates without designing sophisticated index-based data structures for each user. We propose to use an event driven model that addresses these constraints in a unified manner.

The high-level idea of candidates update is to construct a small lexicon that is contextually relevant to the user when they issue a query. For this work, the model ranks snippets generated from the user's recent events based on the recency. The set of events used in our experiments are:

- GMail: Search interactions with emails.
- Drive: Interactions with documents including creation, view, edit, and comment events that are relevant to the user.
- Calendar: Ongoing and upcoming calendar events.

For example, when a user edits a document, n-grams from the document title are prioritized high in the lexicon. When space constraints are reached, the lowest ranked candidates are removed from the lexicon to accommodate new candidates. Prioritization of n-grams may not be necessary in some applications. For instance, in mobile apps search, it might be feasible to retain all of the currently installed app names.

4.4 Properties

There are several interesting properties of our event driven candidate generation method.

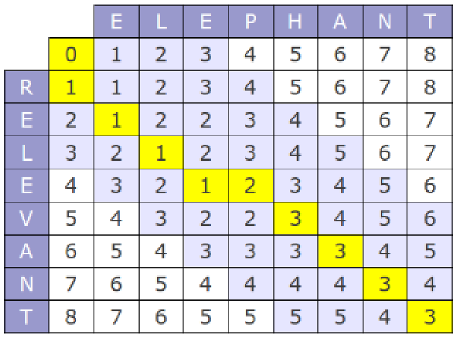
- Previous *search* activity and clicks are helpful but not required, unlike in most prior work. For example, n-grams from Drive documents can be from an edit event, or even initiated by another user (e.g. comment events where this user is tagged, share events, etc).
- Prioritization of n-grams based on recency provides a simple approach to address privacy and resource constraints.
- n-grams from multiple corpora (GMail, Drive, Calendar) are merged into a single lexicon and are used in search for each corpus. This is intuitively useful for our scenario. If someone has been working on a `TheWebConference` paper, it's possible that she gets emails and calendar invites discussing `TheWebConference` and has related documents about the `TheWebConference` paper. We advocate that applications should take into account such a unified representation whenever sensible, since (1) shared knowledge across corpora mitigates the data sparsity problem and (2) a single lexicon for each user is much easier to manage in practice.

5 EFFICIENT SPELL CORRECTION AND QUERY COMPLETION

We describe an efficient spell correction and query completion algorithm using a modified Levenshtein distance based approach. Given each candidate from a user's lexicon and an input query (which may be incomplete and have spelling mistakes), the algorithm calculates a modified edit distance between them, which will be used for spelling correction candidate selection or ranking. We also describe several improvements to further speed up the calculation in practice.

5.1 Levenshtein Distance

Edit distance is defined in terms of the minimum number of operations (insertions, deletions, or substitutions) required to convert the user input string to a candidate spell correction. Levenshtein Distance is a special case of edit distance with three allowed operation: insertion, deletion, and substitution with unit cost for each of them. See an example in Fig. 3. The algorithm is $O(m * n)$ where m is the length of the input string and n is the length of the candidate string.



		E	L	E	P	H	A	N	T
	0	1	2	3	4	5	6	7	8
R	1	1	2	3	4	5	6	7	8
E	2	1	2	2	3	4	5	6	7
L	3	2	1	2	3	4	5	6	7
E	4	3	2	1	2	3	4	5	6
V	5	4	3	2	2	3	4	5	6
A	6	5	4	3	3	3	3	4	5
N	7	6	5	4	4	4	4	3	4
T	8	7	6	5	5	5	5	4	3

Figure 3: Levenshtein Distance algorithm example between user input string `RELEVANT` and candidate string `ELEPHANT`.

5.2 Extending Levenshtein Distance For Query Completion

The main novelty of our algorithm lies in extending the Levenshtein distance algorithm for low-cost query completions that makes it feasible for use in online spell correction scenarios. Fig. 4 shows the edit distance matrix M for the evaluation of Levenshtein distance.

The entry *M[i][j]* denotes the minimum cost to transform *i* characters of the input string to *j* characters of output string. For example, *M[3][5]* in Fig. 4 denotes the cost of transforming `ELE` to `ELIZA`.

The most important row for extending the algorithm for query completion is the last row. If *input_length* denotes the length of the input string, the *jth* column of this row, denoted by *M[input_length][j]*, represents the cost of transforming the complete input string into the first *j* characters of the output string. In other words, this row represents the cost of transforming the input string to each prefix of the output string. Hence, in Fig. 4, each column in the last row represents the cost of transforming `ELEZA` into the corresponding prefix of `ELIZABETH`.

In order to support query completions, we define a new parameter *prefix_completion_cost*. In the edit distance algorithm above, each of the substitution, insertion, and deletion operations have unit cost. This new parameter allows insertion of characters at a different cost, which should be much lower than the unit cost. We take the entries in the last row and count the number of additional characters required to convert the corresponding prefix string to the complete candidate string. For the prefix corresponding to column *j*, we need to insert *output_length* − *j* characters, where *output_length* is the length of the output string. Let *R'* be a new row denoting the overall cost after allowing prefix completion. Hence, *R'[j]* denotes the cost after transforming the input string to a prefix of length *j* of the output string, followed by using low cost prefix completion operations to complete it to the output string.

$$\begin{aligned} R'[j] = {} & M[input\_length][j] \\ & + prefix\_completion\_cost * (output\_length - j). \end{aligned} \tag{1}$$

In Fig. 4, the input string is `ELEZA` and the candidate string is `ELIZABETH`. The edit distance produced by Levenshtein distance is *5*. But since *prefix_completion_cost* is 0.2, one possible sequence of operations is to replace the second "E" in the input string with an "I", which has unit cost, following by completing the obtained prefix by adding `BETH` at the end. This will incur a cost of $0.2 * 4 = 0.8$ resulting in a total cost of 1.8 as the minimum cost.

5.3 Further Speed-up techniques

Take the case of an input string `ELEPHANT` and the candidate string `RELEVANT`. The Levenshtein Distance between the two strings is 3 but we had to calculate all the entries in the matrix.

Suppose that the max distance for candidates that we are interested in is *max_allowed_distance* (i.e., any candidate with a higher edit distance from the query will be filtered out). For Levenshtein Distance, it can be proved that if we calculate a strip of $2 * max\_allowed\_distance + 1$ along the diagonal, any element outside this strip must be greater than *max_allowed_distance*. For example, for $max\_allowed\_distance = 3$, Fig. 3 highlights the strip of interest along the diagonal.

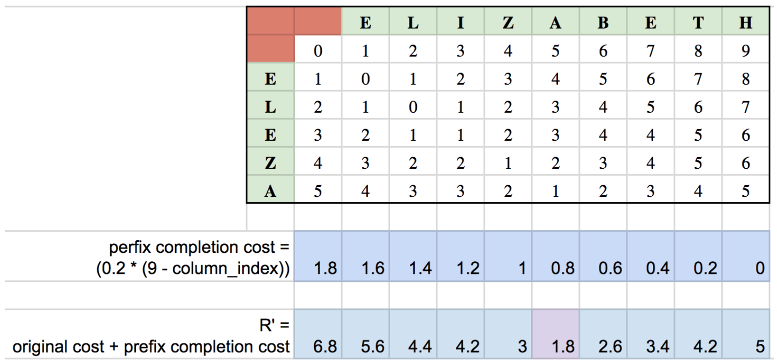
This reduces the complexity of the algorithm from $O(m * n)$ to $O(m * max\_allowed\_distance)$, which is linear in the length of the input string. In practice, we found a *max_allowed_distance* of 2 is sufficient for our use cases.

Another speed-up technique we use is to reject a candidate if its first or second characters do not match the first character of the input string. The first character of an input query is rarely found to contain any error [12]. But it was observed that due to lag in web browsers, the first character is often missing in the input query. Therefore, we relaxed our check to also allow candidates with the second character same as the first character of the input string.

6 EXPERIMENTS

Due to privacy concerns and counterfactual effects (e.g. enabling our algorithm will generate very different search results that are not available in search logs), it is difficult to perform thorough offline analysis except for limited qualitative check (e.g. Fig. 5). We thus directly conduct real-world online experiments against three of the world's largest private corpora search systems: Google's GMail, Drive, and Calendar. These systems use the sophisticated spell correction and query completion mechanisms of Google's web search, thereby providing strong baselines [17].

We demonstrate consistent and significant improvements in effectiveness over these strong production baselines. Since these are online experiments within live systems used by millions of users, we were required to meet the rigorous latency requirements of these systems. The algorithm proposed here easily achieves these requirements. More details about the performance of the proposed algorithm are described in section 6.4.



		E	L	I	Z	A	B	E	T	H
	0	1	2	3	4	5	6	7	8	9
E	1	0	1	2	3	4	5	6	7	8
L	2	1	0	1	2	3	4	5	6	7
E	3	2	1	1	2	3	4	4	5	6
Z	4	3	2	2	1	2	3	4	5	6
A	5	4	3	3	2	1	2	3	4	5
perfix completion cost = (0.2 * (9 - column_index))	1.8	1.6	1.4	1.2	1	0.8	0.6	0.4	0.2	0
R' = original cost + prefix completion cost	6.8	5.6	4.4	4.2	3	1.8	2.6	3.4	4.2	5

Figure 4: Illustration of our augmented Levenshtein Distance algorithm that unifies spell correction and query completion. Column *j* for row *R'* represents the cost of prefix completion after converting input string (`ELEZA`) to a prefix of length *j* of candidate string (`ELIZABETH`).

query_length	1	2	3	4+
Prefix Completion	X	✓	✓	✓
Spell Correction(dist = 1)	X	X	✓	✓
Spell Correction(dist = 2)	X	X	X	✓

Table 1: Desirable behavior of prefix completion and spell correction for different length of queries.

6.1 Setup

Our application scenario is "instant" search, which performs "search as you type" (e.g. see Fig. 1 right). This type of search has stringent latency requirements to ensure a high quality user experience. The spell corrected and prefix completed query is used to retrieve the final list of relevance ranked search results.

We conduct three A/B live experiments, one on each of the above mentioned corpora (GMail, Google Drive, Google Calendar) that augments their existing approaches to spell correction that use global models trained on web data. These experiments are run on millions of users for a month, resulting in many millions of queries being included in the experiment (the exact number being proprietary).

6.2 Hyperparameters

Real-world experiments that affect millions of users forbids trying many hyperparameter combinations. Thus we first define the "desirable behavior" in Tbl. 1 and tune the hyperparameters offline.

We proposed this setting because it has clear and intuitive semantics. For very short queries, neither spell correction nor prefix completion is enabled. But as the query length increases, prefix completion gets enabled followed by both spell correction and prefix completion getting enabled.

In order to achieve the above behavior, we introduce a new parameter *max_cost* which can be used to control *max_allowed_distance* based on the *query_length* as follows:

$$max\_allowed\_distance = max\_cost - \frac{\alpha}{query\_length^2} \qquad (2)$$

Since Levenshtein distance has unit cost for every operation, the number of spelling errors allowed are decided by the integer part of *max_allowed_distance*. We set *max_cost* to 2.7, α to 7, and

query_length	1	2	3	4+
$max\_allowed\_distance$ (max_cost - $\frac{\alpha}{query\_length^2}$)	-4.3	0.95	1.92	2.26
$\lfloor max\_allowed\_distance \rfloor$	-5	0	1	2
Prefix Completion ($max\_allowed\_distance > 0$)	X	✓	✓	✓
Spell correction(dist = 1) ($\lfloor max\_allowed\_distance \rfloor \geq 1$)	X	X	✓	✓
Spell correction(dist = 2) ($\lfloor max\_allowed\_distance \rfloor \geq 2$)	X	X	X	✓

Table 2: Behavior of the algorithm after setting *max_cost* to *2.7* , α to 7 and *prefix_completion_cost* to *0.08.*

prefix_completion_cost to 0.08. These parameters allow us to realize the desirable behavior (compare Tbl. 1 and Tbl. 2).

6.3 Metrics

Through the above mentioned algorithms, we measure the impact on the following search quality metrics:

- **Click-Through Rate (CTR)**: represents the fraction of queries for which a result was clicked.
- **Mean Reciprocal Rank (MRR)**: The reciprocal rank is the multiplicative inverse of the rank of the clicked result: 1 for first place, $\frac{1}{2}$ for second place and so on, and 0 for queries without click. MRR is the mean of reciprocal ranks of all query responses.
- **Has Result Rate**: represents the fraction of queries for which we have at least one result.
- **Number of Results**: represents the mean of the number of results that were shown to the user for a given query. GMail, Drive, and Calendar search engines show a maximum of 6, 5, and 4 results respectively.

For all metrics, larger numbers are better. *CTR* and *MRR* are traditional search quality metrics. *Has result rate* and *Number of Results* are particularly interesting for spell correction, because misspelled queries tend to produce either a low number of results, or no result at all. These metrics signify improved recall while CTR and MRR signify precision improvements.

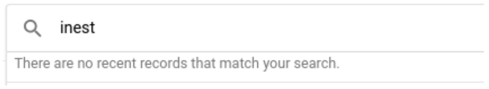
6.4 Runtime

Our per-user lexicon has at most 50 n-grams and 25 email addresses. We consider up to 7-grams. The average number of characters in the input query and candidate unigrams (from lexicon n-grams) was observed to be 5 and 8 respectively. The average number of characters in email addresses was observed to be 23.

Under this configuration, the algorithm takes 0.2ms on single-thread modern hardware to produce corrections. Hence, the time taken by the spell correction component is insignificant and is well-suited for our latency-sensitive system.

6.5 Illustrative Example

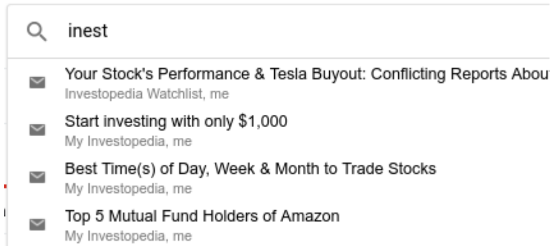
We show an illustrative example in Fig. 5 to demonstrate the quality improvements. As shown, our personalized online spell correction solution is able to successfully correct the query and provide relevant search results, while the baseline system without personalized spell correction produces bad quality results.



inest

There are no recent records that match your search.

(a) Performance without our algorithm.



inest

Your Stock's Performance & Tesla Buyout: Conflicting Reports Abou
Investopedia Watchlist, me

Start investing with only $1,000
My Investopedia, me

Best Time(s) of Day, Week & Month to Trade Stocks
My Investopedia, me

Top 5 Mutual Fund Holders of Amazon
My Investopedia, me

(b) Performance with our algorithm enabled.

Figure 5: A qualitative illustration of our solution in GMail. "investopedia", with its prefix misspelled as "inest", is a word only relevant to certain users. (a) The production system with web spelling is unable to find any relevant email. (b) Since the user interacts frequently with documents containing the word "investopedia", it is used to spell correct "inest" and show relevant emails.

6.6 Online Experiment Results

We report search quality metrics for affected queries for each of the three corpora under consideration. Affected queries in the experiment group are those rewritten by our component (i.e. a spell correction and query completion is applied, and the corrected word is used for search). Affected queries in the control group are those that would be affected if our component were deployed (i.e. we only detect that a spell correction is possible). Our current setup results in 3% of affected queries across all corpora, which (in absolute terms) is a non-trivial amount of search queries. We summarize the results to show relative improvements in Tbl. 3.

We observe consistently positive improvements in search quality metrics across each of the three corpora. These improvements are significant and highly meaningful in the production setting.

Metrics	Email	Drive	Calendar
CTR	+31.55%	+16.04%	+38.27%
MRR	+26.00%	+10.17%	+30.41%
Has Result Rate	+15.55%	+8.51%	+43.34%
Number of Results	+31.13%	+14.53%	+76.61%

Table 3: Search quality improvements on affected queries in our online experiment. All numbers in the table are statistically significant at the $p < 0.05$ level.

Language	Email CTR	Drive CTR	Calendar CTR
English	+28.78%	+14.04%	+38.57%
Spanish	+44.58%	+11.24%	+21.51%
French	+21.02%	+9.60%	+220.93%
Japanese	+12.48%	+1.72%	+32.33%
Portuguese	+25.44%	+65.88%	+50.29%
Italian	+27.19%	+33.04%	+68.05%
Chinese	+41.86%	+28.94%	NA

Table 4: CTR improvements on affected queries in different languages across corpora.

Furthermore, these metrics are quite intuitive. In personal search, a misspelled query without good spell correction is unlikely to locate any meaningful result due to the small personal corpora, leading to low CTR and result count based metrics such as Has Result Rate.

6.7 Cross-Language Performance

We further show that our simple approach works well across different languages in Tbl. 4 in terms of CTR (we omit other metrics with similar trends due to space constraints). These languages are the top languages in terms of query count and cover more than 90% of all queries in our experiments.

Overall we get positive CTR (and other metrics) on a diverse set of languages including languages without natural delimiters like Chinese, Japanese, and Korean. There is variance in terms of quality improvements and further adapting our approach for different languages (e.g. through simple parameter tuning) could yield additional improvements as part of future work.

7 CONCLUSION

In this paper, we introduced a practical solution for personalized spell correction in personal search. We discussed our approach to generate a personal lexicon for users that ensures high quality by extracting them from the user's context derived from recent activities. We then used it in our novel spell correction and query completion algorithm that performs well under the resource constraints and latency requirements.

We further evaluated it on some of the world's largest private corpora (GMail, Google Drive, and Calendar) on hundreds of millions of users and saw highly significant improvements in search quality metrics for each corpus. Even though more sophisticated approaches are needed for global models, restricting our corrections to a small highly relevant lexicon helped us achieve great results with a simple and elegant approach. While many spell correction algorithms need sophisticated mechanisms to support different languages, our improvements are agnostic of the language.

REFERENCES

[1] 2018. I before E except after C. Retrieved 2018-10-30 from https://en.wikipedia.org/wiki/I_before_E_except_after_C

[2] Qingyao Ai, Susan T. Dumais, Nick Craswell, and Dan Liebling. 2017. Characterizing Email Search Using Large-scale Behavioral Logs and Surveys. In *WWW*. 1511–1520.

[3] Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, Donald Metzler, Lance Riedel, and Jeffrey Yuan. 2009. Online Expansion of Rare Queries for Sponsored Search. In *WWW*. 511–520.

[4] Fei Cai, Shangsong Liang, and Maarten de Rijke. 2014. Time-sensitive Personalized Query Auto-Completion. In *CIKM*. 1599–1608.

[5] David Carmel, Guy Halawi, Liane Lewin-Eytan, Yoelle Maarek, and Ariel Raviv. 2015. Rank by Time or by Relevance?: Revisiting Email Search. In *CIKM*. 283–292.

[6] Surajit Chaudhuri and Raghav Kaushik. 2009. Extending Autocompletion to Tolerate Errors. In *ACM SIGMOD*. 707–718.

[7] Qing Chen, Mu Li, and Ming Zhou. 2007. Improving Query Spelling Correction Using Web Search Results. In *EMNLP-CoNLL*. 181–189.

[8] Silviu Cucerzan and Eric Brill. 2004. Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users.. In *EMNLP*. 293–300.

[9] Dong Deng, Guoliang Li, He Wen, H. V. Jagadish, and Jianhua Feng. 2016. META: An Efficient Matching-based Method for Error-tolerant Autocompletion. In *VLDB*. 828–839.

[10] Huizhong Duan and Bo-June Hsu. 2011. Online Spelling Correction for Query Completion. In *WWW*. 117–126.

[11] Susan Dumais, Edward Cutrell, JJ Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. 2003. Stuff I'Ve Seen: A System for Personal Information Retrieval and Re-use. In *SIGIR*. 72–79.

[12] Mohammad Ali Elmi and Martha Evens. 1998. Spelling correction using context. In *ACL*. 360–364.

[13] Pravallika Etoori, Manoj Chinnakotla, and Radhika Mamidi. 2018. Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning. In *ACL, Student Research Workshop*. 146–152.

[14] Pieter Fivez, Simon Šuster, and Walter Daelemans. 2017. Unsupervised Context-Sensitive Spelling Correction of English and Dutch Clinical Free-Text with Word and Character N-Gram Embeddings. *arXiv preprint arXiv:1710.07045* (2017).

[15] Jianfeng Gao, Xiaolong Li, Daniel Micol, Chris Quirk, and Xu Sun. 2010. A Large Scale Ranker-based System for Search Query Spelling Correction. In *COLING*. 358–366.

[16] Sasa Hasan, Carmen Heger, and Saab Mansour. 2015. Spelling Correction of User Search Queries through Statistical Machine Translation. In *EMNLP*. 451–460.

[17] Michael Herscovici, Dan Guez, and Hyung-Jin Kim. 2017 granted. Autocompletion using previously submitted query data. In *US9740780B1*.

[18] Maryam Kamvar, Melanie Kellar, Rajan Patel, and Ya Xu. 2009. Computers and Iphones and Mobile Phones, Oh My!: A Logs-based Comparison of Search Users on Different Devices. In *WWW*. 801–810.

[19] Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*. 707–710.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[21] Harshit Pande. 2017. Effective search space reduction for spell correction using character neural embeddings. In *EAACL*. 170–174.

[22] Milad Shokouhi. 2013. Learning to Personalize Query Auto-completion. In *SIGIR*. 103–112.

[23] Xu Sun, Jianfeng Gao, Daniel Micol, and Chris Quirk. 2010. Learning Phrase-based Spelling Error Models from Clickthrough Data. In *ACL*. 266–274.

[24] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to Rank with Selection Bias in Personal Search. In *SIGIR*. 115–124.