Fast Multi-task Learning for Query Spelling Correction

Xu Sun
Dept. of Statistical Science
Cornell University
Ithaca, NY 14853
xusun@cornell.edu

Anshumali Shrivastava
Dept. of Computer Science
Cornell University
Ithaca, NY 14853
anshu@cs.cornell.edu

Ping Li
Dept. of Statistical Science
Cornell University
Ithaca, NY 14853
pingli@cornell.edu

ABSTRACT

In this paper, we explore the use of a novel online multi-task learning framework for the task of search query spelling correction. In our procedure, correction candidates are initially generated by a ranker-based system and then re-ranked by our multi-task learning algorithm. With the proposed multi-task learning method, we are able to effectively transfer information from different and highly biased training datasets, for improving spelling correction on all datasets. Our experiments are conducted on three query spelling correction datasets including the well-known TREC benchmark dataset. The experimental results demonstrate that our proposed method considerably outperforms the existing baseline systems in terms of accuracy. Importantly, the proposed method is about one order of magnitude faster than baseline systems in terms of training speed. Compared to the commonly used online learning methods which typically require more than (e.g.,) 60 training passes, our proposed method is able to closely reach the empirical optimum in about 5 passes.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Query Alteration*

General Terms

Algorithms, Performance, Experimentation

Keywords

Query Spelling Correction, Multi-task Learning

1. INTRODUCTION

Search queries present a particular challenge for traditional spelling correction methods, for at least three reasons [3]. Firstly, spelling errors are more common in search queries than in regular written texts. For example, [12] showed that roughly 10-15% of queries contain misspelled terms. Secondly, most search queries consist of only a few key words rather than grammatical sentences, making a grammar-based approach inappropriate. Most importantly, many queries contain search terms, such as proper nouns and names, which are not well established in the language. For example, Chen *et al.* [10] reported that 16.5% of valid search terms do not occur in their spelling lexicon, which contains more than 200,000 entries.

Due to the practical importance, query spelling correction has received much attention and a variety of datasets have been developed (from different domains). As discussed in [17], queries in one specific dataset can be very biased from another dataset, and combining such biased datasets for query spelling correction is difficult. In fact, in this paper, we will also demonstrate an empirical phenomenon that simply merging biased datasets for training a unified speller may not bring improvement at all.

The difficulty in effectively combining biased datasets for spelling correction lies in at least three aspects: (i) their error-patterns can be high biased; (ii) their distributions of misspelled queries can be quite different; (iii) their domains can be quite different (hence, a domain adaptation problem exists). Thus, a natural question arises: *how can we effectively integrate such highly biased datasets for improving query spelling correction?* To the best of our knowledge, there are still no satisfactory solutions in the literature.

Our goal in this paper is to solve this well-known difficult problem with high accuracy as well as high efficiency. The training speed can be crucial because industrial query spelling correction tasks may have to deal with very large-scale training data (e.g., click-through query spelling logs which contain more than millions of entries, see [34]). Our proposed multi-task learning framework for query spelling correction is capable of adaptively integrating highly biased training datasets via automatically learning the task-relationships (data-similarities). This allows us to "softly" merge biased datasets for learning a unified speller.

In our procedure, the fist step is to generate a large number of spelling correction candidates using a candidate generation module. Then the top candidates are accurately identified through our novel multi-task learning algorithm. Encouragingly, with the ability to effectively transfer information among those biased training datasets, our proposed method is capable of improving spelling correction accuracies on *all* related datasets. The prior standard practice for combining different datasets is to simply merge them and feed them to a single-task learner. Compared to this

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

baseline, our experimental results will demonstrate that our new approach will result in significantly better performance.

Our **major contributions** can be summarized as follows:

- To the best of our knowledge, this is the first study of multi-task learning for query spelling correction. The proposed method outperforms the state-of-the-art results on the TREC dataset.
- Our novel *adaptive* multi-task learning framework can effectively utilize *2nd-order gradient information*.
- Our method is of fast convergence speed. It is able to approach close to the empirical optimum in only a few (e.g., ≤ 5) passes. This is about one order of magnitude faster than traditional training (e.g., batch or online) methods. Fast training speed is crucial for large-scale query spelling correction tasks (e.g., [34]).

The rest of this paper is structured as follows. Section 2 describes the architecture of our spelling correction system. Section 3 presents the proposed multi-task learning framework for query spelling correction. Section 4 presents the extensive experiments on query spelling correction. Section 5 reviews related work, and Section 6 concludes the paper.

2. SYSTEM ARCHITECTURE

2.1 Candidates Generation

We first describe our procedure for generating correction candidates. Following Ganjisaffar *et al.* [17], we implement three candidate generators in our system to produce about 2,000 candidates (on average) for each query. Initially, a character-based candidate generator is adopted for producing all possible candidates within an edit distance 1. It considers replacing each character with all possible characters in the alphabet, transposing each pair of adjacent characters, deleting each character, inserting all possible characters after each character, and so on.

The AOL query logs showed that 16% of the query corrections are different from the original query only in adding/removing spaces [17]. For example, "ebayauction" is a query which should be corrected to "ebay auction". As a more complicated example, the long query "broccoliandcheesebake" actually indicates "broccoli and cheese bake". In order to handle this class of queries, we implement the word segmentation algorithm based on the Microsoft word breaker[1], with some minor modifications. For each possible segmentation of the character sequence, we use a language model to compute the probability of the segmentation. The most probable segmentation candidates are then added to the candidate list.

Neither of the above two candidate generators is able to produce candidates which have more than 1 edit distance from the original query (in spite of adding or removing multiple spaces). For example, for the query "washton university", we might want to have "washington university" as a candidate, which however can not be produced by the above two candidate generators. To ease this problem, we perform a fuzzy search process based on a lexicon containing most frequent words, to quickly find known unigrams with a small edit distance to unigrams and bigrams located in the original query. For the lexicon containing frequent-word, we use the top 100K words based on their frequency on the web. With the three candidate generators, we can generate 2,000 candidates (on average) for each query.

[1] `http://web-ngram.research.microsoft.com`

2.2 Naive Ranking

An important factor in selecting and ranking the correction candidates is the prior probability of a correction phrase. It represents our prior belief about how likely a query will be chosen by the user without seeing any input from the user. In this work we make use of the Web n-gram service provided by Microsoft. Web n-gram model intends to model the n-gram probability of English phrases with the parameters estimated from the entire Web data. It also differentiates the data sources to build different language models from the title, anchor text and body of Web pages, as well as the queries from query log. To build our spelling system, we make use of the tri-gram language model. Note that, even though the Web n-gram model contains a large amount of entries, it may still suffer from data sparseness in higher-order language models.

2.3 Re-Ranking

Re-ranking of top search results has shown to improve the quality of rankings in query spelling correction [17, 28]. Since the main focus of spelling correction is on the top ranked list of candidates, we add a re-ranker on top of the naive ranker. This step significantly improved the results of the naive ranker, and the cost is tractable. This is because re-ranking only needs to deal with top-k candidates, and k is typically small (in our case, following [28], we set $k = 40$). Hence, the re-ranker is solving a much easier problem, compared to the original naive ranker which needs to consider about 2,000 candidates for each query.

We use a conditional log-linear method, the *Maximum Entropy* model, for re-ranking. The maximum entropy model produces a probability distribution over multiple classes and have the advantage of handling large numbers of overlapping features. Assuming a feature function that maps a pair of observation sequence $\boldsymbol{x}$ and a classification label y to a feature vector $\boldsymbol{f}$, the probability of y conditioned on the observation sequence $\boldsymbol{x}$ is modeled as follows [23]:

$$P(y|\boldsymbol{x}, \boldsymbol{w}) = \frac{\exp\left[\boldsymbol{w}^\top \boldsymbol{f}(y, \boldsymbol{x})\right]}{\sum_{\forall y'} \exp\left[\boldsymbol{w}^\top \boldsymbol{f}(y', \boldsymbol{x})\right]}, \quad (1)$$

where $\boldsymbol{w}$ is a weight vector. Given a training set consisting of n labeled samples, $(\boldsymbol{x}_i, y_i)$, for $i = 1 \ldots n$, weights are gained via maximizing the objective function,

$$\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{n} \log P(y_i|\boldsymbol{x}_i, \boldsymbol{w}) - R(\boldsymbol{w}). \quad (2)$$

The first term of this equation represents a conditional log-likelihood of the training data. The second term is a regularizer for reducing overfitting. We employe an L_2 prior. In what follows, we denote the conditional log-likelihood of each sample, $\log P(y_i|\boldsymbol{x}_i, \boldsymbol{w})$, as $\ell(i, \boldsymbol{w})$. Then, we have

$$\mathcal{L}(\boldsymbol{w}) = \sum_{i=1}^{n} \ell(i, \boldsymbol{w}) - \frac{||\boldsymbol{w}||^2}{2\sigma^2}. \quad (3)$$

Since the evaluation is based on expected F-score, probability information is required in computing expected F-score.

The original conditional probability produced by maximum entropy models is not proper for computing expected F-score, because the sum of conditional probabilities of top-k candidates are not 1. To deal with this problem, we re-compute the probability of a candidate as follows:

$$P'(y) = \exp[cP(y)] / \sum_{k'=1,\ldots,k} \exp[cP(y_{k'})],$$

where c is a scalar to control the density of the distribution. Since essentially c has a similar function like the σ in regularization, to avoid introducing new hyper-parameters, we simply set c based on σ.

2.4 Re-Ranking Features

In the re-ranking phase we add features which are extracted from top-k query-candidate pairs. In addition, other than the features which based on transformations between query-candidate pairs, we also design features which consider the top candidates of the query (i.e., comparisons with other candidates in the top-k list). In this way, the features can include valuable information that may help in the ranking process.

We have about 30 feature templates for each query-candidate pair. These features include: error model features (e.g., edit distance), candidate language model features, query language model features, surface features capturing differences of the query and the candidate, frequency of the query and the candidate, the rank of the candidate in the naive ranking phase, and so on. Although we have used language model scores for naive ranking for producing top candidates, we keep using Web-scale n-gram language model features for re-ranking the top candidates, and we find such Web-scale n-gram language model features are still quite useful in the re-ranking phase.

In the Web-scale n-gram language model features, the log of n-gram language model probabilities of an original query and its candidate corrections are used for re-ranking. In addition, the average, max, min, and standard deviation of the language model probabilities of the top-k candidates are employed as features.

2.5 Online Learning

There are two major approaches for training a log-linear model: batch training and online training. Batch training methods include, for example, steepest gradient descent, conjugate gradient descent (CG), and limited-memory BFGS (LBFGS) [30]. In such training methods, gradients are computed by using all training instances. Typically, the training process is quite slow in practice.

To speed up the training process, online algorithms have become increasingly popular. A representative online learning method is the stochastic gradient descent (SGD) [7]. The SGD uses a small randomly-drawn subset of the training samples to approximate the gradient of the objective function, which allows one to update the model weights much more frequently, and consequently, to speed up the convergence. Suppose $\hat{\mathcal{S}}$ is a randomly drawn subset of the full training set $\mathcal{S}$, the stochastic objective function is then given by

$$\mathcal{L}_{stoch}(\boldsymbol{w}, \hat{\mathcal{S}}) = \sum_{i \in \mathcal{S}} \ell(i, \boldsymbol{w}) - \frac{|\hat{\mathcal{S}}|}{|\mathcal{S}|} \frac{||\boldsymbol{w}||^2}{2\sigma^2}.$$

The extreme case is a batch size of 1, and it gives the maximum frequency of updates, which we adopt in this work. In this case, $|\hat{\mathcal{S}}| = 1$ and $|\mathcal{S}| = n$ (suppose the full training set contains n samples). In this case, we have

$$\mathcal{L}_{stoch}(\boldsymbol{w}, \hat{\mathcal{S}}) = \ell(i, \boldsymbol{w}) - \frac{1}{n} \frac{||\boldsymbol{w}||^2}{2\sigma^2}, \tag{4}$$

where $\hat{\mathcal{S}} = \{i\}$. Model weights are updated like this:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \gamma_k \nabla_{\boldsymbol{w}_k} \mathcal{L}_{stoch}(\boldsymbol{w}, \hat{\mathcal{S}}), \tag{5}$$

where k is the update counter, γ_k is the learning rate.

3. OUR PROPOSAL

In this section, we introduce our adaptive (and online) ***m***ulti-***t***ask ***l***earning framework (MTL below). For every positive integer q, we define $\mathcal{N}_q = \{1, \ldots, q\}$. Let T be the number of tasks which we want to simultaneously learn. For each task $t \in \mathcal{N}_T$, there are n data examples $\{(\boldsymbol{x}_{t,i}, \boldsymbol{y}_{t,i}) : i \in \mathcal{N}_n\}$ available. In practice, the number of examples per task may vary but we keep it constant for simplicity of notation. We use $\boldsymbol{D}$ to denote the $n \times T$ matrix whose t-th column is given by the vector $\boldsymbol{d}_t$ of data examples.

3.1 Adaptive Multi-Task Learning Model

Our goal is to learn the weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T$ from the data $\boldsymbol{D}$. For simplicity of notation, we assume that each of the weight vectors is of the same size f (this is also the feature dimension), and corresponds to the same ordering of features. We use $\boldsymbol{W}$ to denote the $f \times T$ matrix whose t-th column is given by the vector $\boldsymbol{w}_t$. We learn $\boldsymbol{W}$ by maximizing the objective function,

$$\text{Obj}(\boldsymbol{W}, \boldsymbol{D}) \triangleq \text{Likelihood}(\boldsymbol{W}, \boldsymbol{D}) - R(\boldsymbol{W}), \tag{6}$$

where $\text{Likelihood}(\boldsymbol{W}, \boldsymbol{D})$ is the accumulative likelihood over all interactive tasks, namely,

$$\text{Likelihood}(\boldsymbol{W}, \boldsymbol{D}) = \sum_{t \in \mathcal{N}_T} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{D}), \tag{7}$$

and $\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{D})$ is defined as follows:

$$\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{D}) \triangleq \sum_{t' \in \mathcal{N}_T} \left[\alpha_{t,t'} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'})\right]. \tag{8}$$

$\alpha_{t,t'}$ is a real-valued *task-similarity*, with $\alpha_{t,t'} = \alpha_{t',t}$ (symmetric). Intuitively, a task-similarity $\alpha_{t,t'}$ measures the *similarity of patterns* between the t-th task and the t'-th task. $\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'})$ is defined as follows:

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'}) &\triangleq \sum_{i \in \mathcal{N}_n} \log P(\boldsymbol{y}_{t',i} | \boldsymbol{x}_{t',i}, \boldsymbol{w}_t) \\ &= \sum_{i \in \mathcal{N}_n} \ell_{t'}(i, \boldsymbol{w}_t), \end{aligned} \tag{9}$$

where $P(\cdot)$ is a prescribed probability function. We can flexibly use any prescribed probability function. This makes our MTL method a flexible and general framework for no matter structured or non-structured classification tasks. In this paper, we will adopt the maximum entropy probability function (Eq. 1), which works well for spelling correction.

Finally, $R(\boldsymbol{W})$ is a regularization term for dealing with overfitting. In this paper, we simply use L_2 regularization:

$$R(\boldsymbol{W}) = \sum_{t \in \mathcal{N}_T} \frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2}. \tag{10}$$

MTL with *fixed* task-similarities (**MTL-F**)

Input: Initialize $\boldsymbol{W}^{(0)}$; given $\boldsymbol{D}, \boldsymbol{A}^*, \beta$; $k \leftarrow 0$
  for $t \leftarrow 1$ to T
  . Initialize $\boldsymbol{\eta}^{(0)}$
  . **Repeat** until convergence
  . . $\boldsymbol{g}_t \leftarrow -\frac{1}{n}\nabla_{\boldsymbol{w}_t}\frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2}$
  . . **for** $t' \leftarrow 1$ to T
  . . . Draw $i \in \mathcal{N}_n$ at random
  . . . $\boldsymbol{g}_t \leftarrow \boldsymbol{g}_t + \boldsymbol{A}^*_{t,t'}\nabla_{\boldsymbol{w}_t}\ell_{t'}(i, \boldsymbol{w}_t)$
  . . $\boldsymbol{w}_t^{(k+1)} \leftarrow \boldsymbol{w}_t^{(k)} + \boldsymbol{\eta}^{(k)} \cdot \boldsymbol{g}_t$
  . . **if** $k+1 \bmod 2 = 0$
  . . . $\boldsymbol{v}_i \leftarrow \frac{\boldsymbol{w}_t^{(k+1)}(i) - \boldsymbol{w}_t^{(k)}(i)}{\boldsymbol{w}_t^{(k)}(i) - \boldsymbol{w}_t^{(k-1)}(i)}$
  . . . Lower-bounds $\boldsymbol{v}_i$ with β
  . . . $\boldsymbol{\eta}^{(k+1)} \leftarrow \boldsymbol{v} \cdot \boldsymbol{\eta}^{(k)}$
  . . **else**
  . . . $\boldsymbol{\eta}^{(k+1)} \leftarrow \boldsymbol{\eta}^{(k)}$
  . . $k \leftarrow k+1$
Output: $\forall t$, $\boldsymbol{w}_t^{(k)}$ converges to $\boldsymbol{w}_t^*$: $\boldsymbol{W}^{(k)}$ converges.

MTL with *unknown* task-similarities (**MTL**)

Input: Initialize $\boldsymbol{W}^{(0)}, \boldsymbol{A}^{(0)}$; given $\boldsymbol{D}$; $k \leftarrow 0$
  Repeat until convergence
  . $\boldsymbol{W}^{(k+1)} \leftarrow$ MTL-F$(\boldsymbol{W}^{(k)}, \boldsymbol{A}^{(k)}, \boldsymbol{D})$
  . **for** $t \leftarrow 1$ to T
  . . **for** $t' \leftarrow 1$ to T
  . . . Update $\boldsymbol{A}_{t,t'}^{(k+1)}$ with Eq.20/21
  . $k \leftarrow k+1$
Output: $\boldsymbol{A}^{(k)}$ converges to $\hat{\boldsymbol{A}}$; $\boldsymbol{W}^{(k)}$ converges to $\hat{\boldsymbol{W}}$.

Figure 1: MTL algorithms (using batch size of 1). The derivation of $\frac{1}{n}$ before the regularization term was explained in Eq. 4. Lower-bounding $\boldsymbol{v}_i$ with β is for stability consideration in the online setting.

To summarize, the overall objective function is as follows:

$$\text{Obj}(\boldsymbol{W}, \boldsymbol{D}) = \sum_{t,t' \in \mathcal{N}_T} \Big[\alpha_{t,t'} \sum_{i \in \mathcal{N}_n} \ell_{t'}(i, \boldsymbol{w}_t)\Big] - \sum_{t \in \mathcal{N}_T} \frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2}.$$

To simplify denotations, we introduce a $T \times T$ matrix $\boldsymbol{A}$, such that $\boldsymbol{A}_{t,t'} \triangleq \alpha_{t,t'}$. We also introduce a $T \times T$ functional matrix $\boldsymbol{\Phi}$, such that $\boldsymbol{\Phi}_{t,t'} \triangleq \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'})$. Then, the objective function can be compactly expressed as follows:

$$\text{Obj}(\boldsymbol{W}, \boldsymbol{D}) = \text{tr}(\boldsymbol{A}\boldsymbol{\Phi}^\top) - \sum_{t \in \mathcal{N}_T} \frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2}, \tag{11}$$

In the following content, we will first discuss a simple case that the task-similarity matrix $\boldsymbol{A}$ is fixed. After that, we will focus on the case that $\boldsymbol{A}$ is unknown, because the task-relationships are unknown among the three query spelling correction tasks that we will focus on.

3.2 MTL with Fixed Task-Similarities

Although the task-similarities are unknown for query spelling correction, we will present a learning algorithm that iteratively reduce the problem to a case of fixed task-similarities. Hence, it is important to discuss the case of fixed task-similarities. With fixed task-similarities, the optimization problem is as follows:

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmax}} \left[\text{tr}(\boldsymbol{A}^*\boldsymbol{\Phi}^\top) - \sum_{t \in \mathcal{N}_T} \frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2} \right]. \tag{12}$$

It is clear to see that we can independently optimize $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t'}$ $(t \neq t')$ given fixed task-similarities. Hence, we can independently optimize each column of $\boldsymbol{W}$ and derive $\boldsymbol{W}^*$:

$$\boldsymbol{w}_t^* = \underset{\boldsymbol{w}_t}{\text{argmax}}\, \psi(\boldsymbol{w}_t, \boldsymbol{D}), \tag{13}$$

where $\psi(\boldsymbol{w}_t, \boldsymbol{D})$ has the form as follows:

$$\psi(\boldsymbol{w}_t, \boldsymbol{D}) = \sum_{t' \in \mathcal{N}_T} \left[\alpha^*_{t,t'} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'})\right] - \frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2}. \tag{14}$$

3.2.1 2nd-Order Gradient Information

For high convergence speed, an important issue of MTL-F is to effectively and efficiently approximate the Hessian matrix. Following the work of [20] on single-task learning, we present a simple yet effective method to approximate the eigenvalues of the Jacobian matrix of a fixed point iterative mapping. In MTL-F, the update formula is as follows:

$$\boldsymbol{w}_t^{(k+1)} = \boldsymbol{w}_t^{(k)} + \boldsymbol{\eta}_t \cdot \boldsymbol{g}_t, \tag{15}$$

The update term $\boldsymbol{g}_t$ is derived by weighted sampling over different tasks. The weighted sampling is based on fixed task-similarities, $\boldsymbol{A}^*$. $\boldsymbol{g}_t$ has a form as follows:

$$\boldsymbol{g}_t = \sum_{t' \in \mathcal{N}_T} \left[\alpha^*_{t,t'} \nabla_{\boldsymbol{w}_t} \ell_{t'}(i_{t'}, \boldsymbol{w}_t)\right] - \frac{1}{n}\nabla_{\boldsymbol{w}_t}\frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2}, \tag{16}$$

where $\alpha^*_{t,t'} = \boldsymbol{A}^*_{t,t'}$ and $i_{t'}$ indexes a random sample selected from $\boldsymbol{d}_{t'}$. Then, the expectation (over distribution of data) of the update term is as follows:

$$\begin{aligned}
\text{E}(\boldsymbol{g}_t) &= \sum_{t' \in \mathcal{N}_T} \left\{ \alpha^*_{t,t'} \left[\frac{1}{n} \nabla_{\boldsymbol{w}_t} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'}) \right] \right\} - \frac{1}{n}\nabla_{\boldsymbol{w}_t}\frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2} \\
&= \frac{1}{n}\left\{ \sum_{t' \in \mathcal{N}_T} \left[\alpha^*_{t,t'} \nabla_{\boldsymbol{w}_t} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{d}_{t'})\right] - \nabla_{\boldsymbol{w}_t}\frac{||\boldsymbol{w}_t||^2}{2\sigma_t^2} \right\} \\
&= \frac{1}{n}\nabla_{\boldsymbol{w}_t}\psi(\boldsymbol{w}_t, \boldsymbol{D}).
\end{aligned}$$

In addition, $\boldsymbol{\eta}_t \in \mathbb{R}^f_+$ is a positive vector-valued step size and $\cdot$ denotes component-wise (Hadamard) product of two vectors. As presented in [6], the optimal step size is the one that asymptotically approaches to $\boldsymbol{H}_t^{-1}$, the inverse Hessian matrix of $\psi(\boldsymbol{w}_t, \boldsymbol{D})$ in our setting. To avoid actually evaluating $\boldsymbol{H}_t^{-1}$, we can approximate $\boldsymbol{H}_t^{-1}$ with its eigenvalues. Following [20], we consider an update iterate as a fixed-point iterative mapping (though a stochastic one) $\mathcal{M}$. Taking partial derivative of $\mathcal{M}$ with respect to $\boldsymbol{w}_t$, we have

$$\boldsymbol{J}_t = \mathcal{M}' = \boldsymbol{I} - diag(\boldsymbol{\eta}_t)\boldsymbol{H}_t. \tag{17}$$

By exploiting this linear relation between Jacobian and Hessian, we can obtain approximate eigenvalues of inverse Hessian using eigenvalues of Jacobian:

$$eigen_i(\boldsymbol{H}_t^{-1}) = \frac{\boldsymbol{\eta}_t(i)}{1 - eigen_i(\boldsymbol{J}_t)}. \tag{18}$$

In addition, $eigen_i(\boldsymbol{J}_t)$ can be asymptotically approximated:

$$eigen_i(\boldsymbol{J}_t) \approx \lambda_i \triangleq \frac{\boldsymbol{w}_t^{(k+1)}(i) - \boldsymbol{w}_t^{(k)}(i)}{\boldsymbol{w}_t^{(k)}(i) - \boldsymbol{w}_t^{(k-1)}(i)}. \tag{19}$$

When k is sufficiently large, λ_i will be sufficiently close to $eigen_i(\boldsymbol{J}_t)$. Therefore, we can asymptotically approximate the inverse of the Hessian matrix via efficient estimation of the Jacobian matrix of fixed-point mapping.

In multi-task setting, this optimization problem is a cost-sensitive optimization problem. To summarize our discussion, we present the *2nd-order gradient based MTL algorithm with fixed task-similarities* (*MTL-F* below) in Figure 1 (upper). As we can see, the algorithm iteratively approximates the eigenvalues of the inverse Hessian matrix via Eq. 19. This approximation is based on data points that are cost-sensitively sampled from multiple tasks. Then, it updates the vector-valued step size based on the inverse Hessian information, and the multi-task model weights are updated accordingly. This iterative process continues until convergence.

3.3 MTL with Unknown Task-Similarities

For our query spelling correction tasks, the task-similarities are unknown. To solve this problem, we present an algorithm to learn the task-similarities and model weights in an alternating optimization manner. Our alternating learning algorithm with unknown task-similarities, called *MTL*, is presented in Figure 1 (bottom).

In the MTL learning, the MTL-F algorithm is employed as a subroutine. In the beginning of the MTL, model weights $\boldsymbol{W}$ and task-similarities $\boldsymbol{A}$ are initialized. $\boldsymbol{W}$ is then optimized to $\hat{\boldsymbol{W}}$ by using the MTL-F algorithm, based on the fixed $\boldsymbol{A}$. Then, in an alternative way, $\boldsymbol{A}$ is updated based on the optimized weights $\hat{\boldsymbol{W}}$. After that, $\boldsymbol{W}$ are optimized based on updated (and fixed) task-similarities. This iterative process continues until empirical convergence of $\boldsymbol{A}$ and $\boldsymbol{W}$.

In updating task-similarities $\boldsymbol{A}$ based on $\boldsymbol{W}$, a natural idea is to estimate a task-similarity $\alpha_{t,t'}$ based on the similarity between weight vectors, $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t'}$. It is unclear which similarity measure best fits the tasks of query spelling correction. To study this, we propose two candidate similarity measures for query spelling correction:

Polynomial kernel (Poly): We can use (normalized) polynomial kernel to estimate similarities:

$$\alpha_{t,t'} \triangleq \frac{1}{C} \frac{\langle \boldsymbol{w}_t, \boldsymbol{w}_{t'} \rangle^d}{||\boldsymbol{w}_t||^d \cdot ||\boldsymbol{w}_{t'}||^d}, \tag{20}$$

where $\langle \boldsymbol{w}_t, \boldsymbol{w}_{t'} \rangle$ means inner product between the two vectors; d is the degree of the polynomial kernel; $||\boldsymbol{w}_t||^d \cdot ||\boldsymbol{w}_{t'}||^d$ is the normalizer. C is a real-valued constant for tuning the magnitude of task-similarities. Intuitively, a big value of C will result in "weak multi-tasking" and a small value of C will make "strong multi-tasking". For example, when $d = 1$, the normalized kernel has exactly the form $\frac{1}{C}\cos\theta$, where θ is the angle between $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t'}$ in the Euclidean space.

Correlation (Cor): Since the *covariance* of task weight vectors is a natural way to estimate inter-task interactions, we consider using covariance information for estimating task-similarities. However, we find directly using a covariance matrix (to estimate task-similarities) faces the problem of stability in our online setting. Hence, we use the correlation matrix via normalizing the covariance matrix.

$$\alpha_{t,t'} \triangleq \frac{1}{C} cor(\boldsymbol{w}_t, \boldsymbol{w}_{t'}) = \frac{1}{C} \frac{cov(\boldsymbol{w}_t, \boldsymbol{w}_{t'})}{std(\boldsymbol{w}_t) std(\boldsymbol{w}_{t'})}, \tag{21}$$

Table 1: Statistics of the three spelling correction datasets, and the top-1 accuracy of naive ranking (without re-ranking). *#Candidates* is the number of correction candidates generated for re-ranking.

Data	#Queries	#Candidates	Top-1 Acc. of NR (%)
JDB	1.2×10^4	4.8×10^5	45.7
MSN	5.0×10^3	2.0×10^5	35.5
TREC	6.0×10^3	2.4×10^5	31.0

where $cov(\boldsymbol{w}_t, \boldsymbol{w}_{t'})$ is the covariance between $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t'}$. $std(\cdot)$ is *standard deviation*.

3.4 Accelerated MTL Learning

The MTL learning algorithm can be further accelerated. The naive 2MTL learning algorithm waits for the convergence of the model weights $\boldsymbol{W}$ (in the MTL-F step) before updating the task-similarities $\boldsymbol{A}$. In practice, we can update task-similarities $\boldsymbol{A}$ before the convergence of the model weights $\boldsymbol{W}$.

For example, we can update task-similarities $\boldsymbol{A}$ after running the MTL-F step over a small number of training passes. We will adopt this accelerated version of the MTL learning for experiments so that the task-similarities will not be repeatedly updated. In the experiment section, we will compare the accelerated MTL method with a variety of strong baseline methods.

4. EXPERIMENTS ON TARGET DATA

We will test the proposed method on three query spelling correction tasks. We will study the MTL method with different similarity measures for spelling correction, and compare them against a number of strong baselines, including traditional batch and online learning methods. The results have been averaged over 5 runs for random permutations of the training data order, and standard deviations are given.

4.1 Datasets

Since the **TREC** dataset (released by Microsoft Speller Challenge 2011) is a new and well-known benchmark dataset for query spelling correction, we will use this dataset for experiments. The TREC dataset is based on the publicly available TREC queries (2008 Million Query Track). This dataset contains 5,892 queries and corrections annotated by the Speller Challenge organizers. There could be more than one plausible corrections for a query. In this dataset only 5.3% of queries are judged as misspelled. We use the same split of the training and testing data.

To improve the performance of query spelling correction, we use two auxiliary datasets for performing multi-task learning. One auxiliary dataset is the **JDB** dataset [17] collected from the publicly available AOL query sets, with a total of 12,000 samples, and 2,000 of them (16.7%) are different from the original query. The second auxiliary dataset is collected from **MSN** queries, and for each query there is at most only one correction. In this dataset, about 11% of queries are judged as misspelled. Also, we evenly split the two auxiliary datasets into training and testing data. We show the statistics of the three datasets in Table 1. In the table, we also show the top-1 accuracy (accuracy of the top-1 candidate) of naive ranking using language models (without

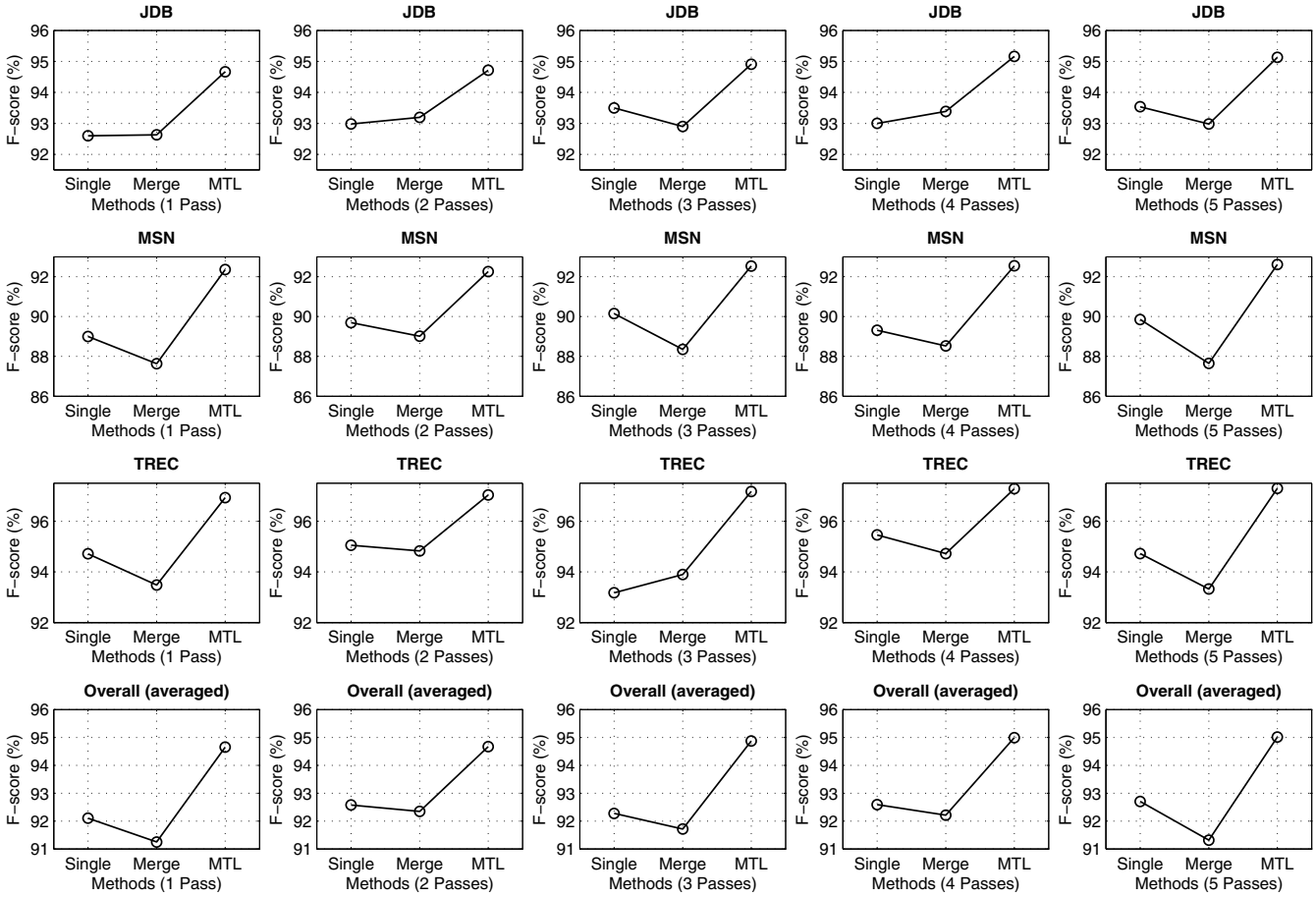

Figure 2: F-scores of different methods in 5 passes. *Single* and *Merge* are based on SGD training.

re-ranking). As we can see, the naive ranking has low accuracies. This reflects the importance of the re-ranking.

The two auxiliary datasets are biased from the TREC dataset in three aspects. First, the queries in those datasets are from three different domains: TREC, AOL, and MSN domains. Second, the distribution of misspelled queries are very different: 5.3% in TREC dataset, 16.7% in JDB dataset, and 11% in MSN dataset. Finally, the number of correct spelling suggestions are different: For the MSN dataset, for each query there is only one correction. On the other hand, for the TREC and JDB datasets, for each query there can be multiple corrections. Hence, it is expected to be a big challenge to integrate those three highly biased datasets for improving spelling correction.

4.2 Settings

Four baselines are adopted to make a comparison with the proposed multi-task learning method, including the limited-memory BFGS batch training method [30] with *single-task setting* (LBFGS-Single), the limited-memory BFGS batch training method with *merged setting* (LBFGS-Merge), the stochastic gradient descent method with single-task setting (SGD-Single), and the SGD method with merged setting (SGD-Merge). For the *single-task setting*, it uses only the task's data to train the re-ranker of the speller (i.e., no data from other spelling correction tasks). For the *merged setting*, it merges all of the training datasets of different tasks to train a unified re-ranker for the speller.

For the multi-task learning method, its hyper-parameters of similarity kernels are tuned in preliminary experiments, and we find using $d = 1$ worked well. In practice, we break the symmetric setting of C (in similarity kernels) and set different values of C for different tasks. We set $\beta = 0.99$ for the proposed MTL method, following the prior work on single-task learning [20]. The proposed method and the four baseline methods use exactly the same features, which is presented before. For the LBFGS method, we use the OWLQN software. The hyper-parameters of LBFGS were left unchanged from the default settings of the OWLQN software.

4.3 Results

We use the expected F1 score as the evaluation metric. This is the same evaluation metrics as the Speller Challenge 2011[2]. Following the previous work of [28], top 40 corrections are used as the default setting. Using this setting, we can compare our results with the results of [28].

The performance comparisons from 1-pass to 5-pass settings are highlighted in Figure 2. In this figure, the *MTL* represents MTL-Poly; *Single* represents the *SGD-Single* baseline. Similarly, *Merge* represents *SGD-Merge*. We can

[2]`http://web-ngram.research.microsoft.com/spellerchallenge/Rules.aspx`

Table 2: Results (expected F1-score, standard deviation, and training time) of MTL and baselines in 5 passes. *MTL-Poly* and *MTL-Cor* represent the MTL method with polynomial kernel and correlation kernel, respectively. The standard deviations have been derived over 5 runs for random permutations of the training data order.

Method (#passes)	JDB F1 (%)	MSN F1 (%)	TREC F1 (%)	Overall F1 (%)	Overall Time (sec)
LBFGS-Single (1) (batch)	19.3	15.8	10.8	15.3 (±0.0)	6.0
LBFGS-Single (5) (batch)	93.6	91.1	77.7	87.4 (±0.0)	36.4
LBFGS-Merge (1) (batch)	19.5	15.7	10.7	15.3 (±0.0)	5.5
LBFGS-Merge (5) (batch)	93.5	90.9	96.4	93.6 (±0.0)	35.5
SGD-Single (1) (online)	92.5	88.9	94.7	92.1 (±0.4)	22.3
SGD-Single (5) (online)	93.5	89.8	94.7	92.7 (±0.5)	111.3
SGD-Merge (1) (online)	92.6	87.6	93.4	91.2 (±1.1)	24.2
SGD-Merge (5) (online)	92.9	87.6	93.3	91.3 (±1.6)	116.4
MTL-Poly (1) (new)	94.6	92.3	96.9	94.6 (±0.1)	23.1
MTL-Poly (5) (new)	95.1	92.6	97.2	95.0 (±0.1)	226.7
MTL-Cor (1) (new)	94.7	92.1	96.9	94.6 (±0.1)	22.1
MTL-Cor (5) (new)	95.2	92.6	97.3	95.0 (±0.1)	226.7

Table 3: Results of different methods on their convergence.

Method (#passes)	JDB F1 (%)	MSN F1 (%)	TREC F1 (%)	Overall F1 (%)	Overall Time (sec)
LBFGS-Single (200) (batch)	95.0	92.3	97.0	94.8	1479.1
LBFGS-Merge (200) (batch)	95.0	91.9	97.2	94.7	1360.9
SGD-Single (60) (online)	95.1	92.2	96.9	94.7	1370.9
SGD-Merge (60) (online)	95.0	91.8	97.2	94.7	1393.4
MTL-Poly (30) (new)	95.3	92.5	97.3	95.1	1463.3
MTL-Cor (30) (new)	95.3	92.5	97.3	95.1	1468.1

see that the proposed method MTL achieves much better F-score than all of the baseline methods on the 1-pass setting. In a similar way, MTL outperforms all of the baselines on the 2-pass, 3-pass, 4-pass, and 5-pass settings, and most of the differences are statistically significant.

In general, we find SGD-Merge does not have considerable advantage over the SGD-Single method. This indicates that it is difficult to combine such highly biased datasets for spelling correction, and a simple merge of such biased datasets does not give considerable improvement.

The experimental results in 5 passes are summarized in Table 2 with more details, including the results of batch training, the training time of different methods, and the comparisons between different similarity kernels. In addition, the results of different methods on their convergence state are shown in Table 3.

As we can see, the batch training (LBFGS) has weak performance in 5 passes, compared with SGD and MTL online training methods. The two similarity kernels (polynomial one and correlation one) have very similar performance on the MTL setting. Comparing the 5-pass results of MTL with the baseline results on their convergence, we find the MTL (with 5 passes) outperforms the SGD with 60 passes and LBFGS with 200 passes. In addition, the MTL with 5 passes is quite close to its empirical optimum on convergence. Compared with no matter the online or batch baselines, the MTL method is about one-order magnitude faster in terms of the empirical convergence speed.

In addition, we summarize all F-score curves by changing the number of passes, so that we can check the training process. The curves are shown in Figure 3. We also show the F-score curves based on training time. As we can see, the MTL method can achieve better performance than baselines, with fewer training passes and fewer training time.

5. RELATED WORK

5.1 Spelling Correction

Spelling correction for regular written text is a long standing research topic. Previous researches can be roughly grouped into two categories: correcting non-word errors and real-word errors. In non-word error spelling correction, any word that is not found in a pre-compiled lexicon is considered to be misspelled. Then, a list of lexical words that are similar to the misspelled word are proposed as candidate spelling corrections.

Most traditional systems use a manually tuned similarity function (e.g., edit distance function) to rank the candidates, as reviewed by [11, 22]. During the last two decades, statistical error models learned on training data (i.e., query-correction pairs) have become increasingly popular, and have proven more effective [21, 37].

Real-word spelling correction is also referred to as context sensitive spelling correction (CSSC). Real-word spelling correction tries to detect incorrect usages of a valid word based on its context. A common strategy in CSSC is as follows. First, a pre-defined confusion set is used to generate candidate corrections, then a scoring model, such as a tri-gram language model or naive Bayes classifier, is used to rank the candidates according to their context (e.g., [19, 29]).

When designed to handle regular written text, both CSSC and non-word error speller systems rely on a pre-defined vocabulary (i.e., either a lexicon or a confusion set). However, in query spelling correction, it is impossible to compile such a vocabulary, and the boundary between the non-word

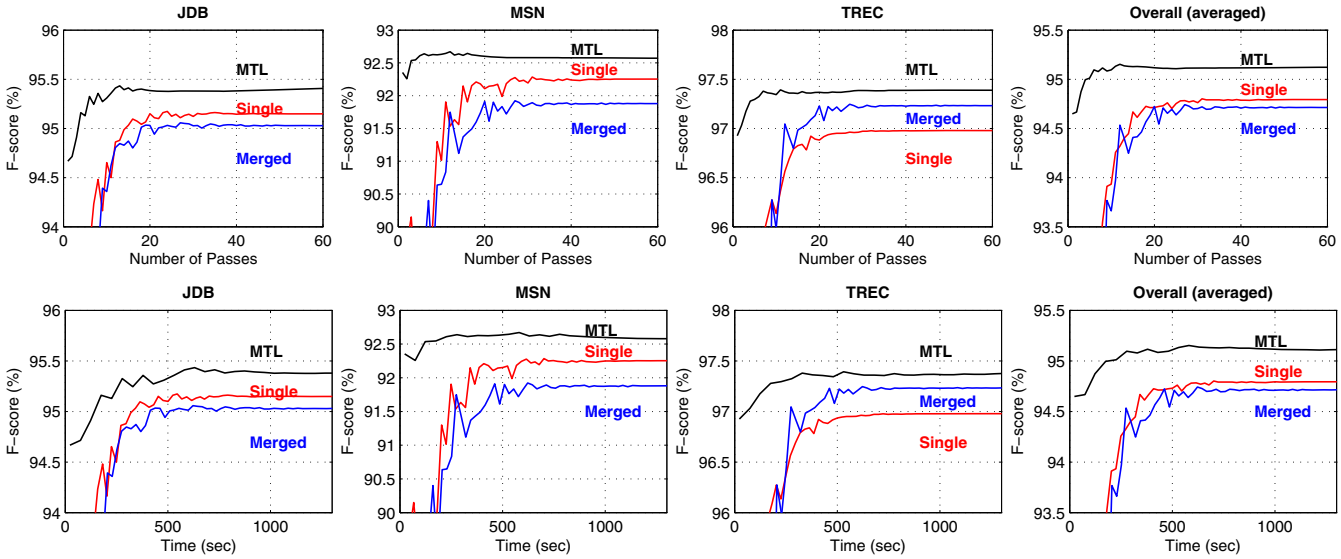

Figure 3: F-score curves of different methods. Upper panels are based on training passes. Bottom panels are based on training time (seconds). *Single* and *Merge* are based on SGD training.

and real-word errors is quite vague. Therefore, recent research on query spelling correction has focused on exploiting noisy Web data and query logs to infer knowledge about misspellings and word usage in search queries.

Cucerzan and Brill [12] discuss in detail the challenges of query spelling correction, and suggest the use of query logs. Ahmad and Kondrak [3] propose a method of estimating an error model from query logs using the EM algorithm. Li et al. [25] extend the error model by capturing word-level similarities learned from query logs. Chen *et al.* [10] suggest using web search results to improve spelling correction. Whitelaw *et al.* [39] present a query speller system in which both the error model and the language model are trained using Web data. Research in this direction also includes more recent work on utilizing large web corpora and query logs [10, 17, 28], employing large-scale n-gram models [17, 28], training phrase-based error model from clickthrough data [34, 18], and so on. Other related work includes [38, 15, 32, 36].

5.2 Multi-task Learning

Multi-task learning has been the focus of much interest in machine learning societies over the last decade. Traditional multi-task learning methods include: sharing hidden nodes in neural networks [8]; feature augmentation among interactive tasks [13]; producing a common prior in hierarchical Bayesian models [42, 43]; sharing parameters or common structures on the learning or predictor space [24, 4]; multi-task feature selection [41]; and matrix regularization based methods [5, 40], among others.

Recent development of multi-task learning is online multi-task learning, started from [14]. [14] assumes the tasks are related by a global loss function and the goal is to reduce the overall loss via online algorithm. With a similar but somewhat different motivation, [1] and [2] studied alternate formulations of online multi-task learning under traditional expert advice models. This is a formulation to exploit low dimensional common representations [16, 31]. Online multi-task learning is also considered via reducing mistake bounds [9], and via perceptron-based online multi-task learning [33].

One of our target is adaptive online multi-task learning. Our adaptive (online) multi-task learning methods not only learn model weights, but also learn task relationships simultaneously from data [35]. More importantly, the proposed method can effectively estimate 2nd-order information, so that we can achieve very fast convergence of the multi-task learning. Finally, our proposal is a general framework which allows *non-structured* and *structured* classification.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an adaptive (and online) multi-task learning method to integrate highly biased datasets for query spelling correction. We performed experiments on three different query spelling correction datasets, including the well-known TREC benchmark dataset. Experimental results demonstrated that the proposed method considerably outperformed the existing baseline systems in terms of accuracy.

Importantly, the proposed method was about one-order magnitude faster than baseline systems in terms of training speed. In contrast to the baseline methods which require more than 60 passes in training, the proposed method can approach very close to the empirical optimum in five passes. Since query spelling correction can have very large-scale training data in industrial applications (e.g., query spelling logs which contain more than millions of entries, see [34]), the proposed method's ability to approach empirical optimum in 1-pass or a few passes will be of critical importance in such large-scale applications.

The proposed multi-task learning method is a general technique, and it can be easily applied to other classification tasks. As future work, we plan to apply this method to other large-scale web search and data mining tasks. Also, we will explore the possibility of integrating online learning with modern hashing algorithms [26, 27] to solve extremely large-scale problems.

7. ACKNOWLEDGMENTS

This work is partially supported by NSF (DMS-0808864, SES-1131848), ONR (YIP-N000140910911), and DARPA (FA-8650-11-1-7149). Xu Sun was a Postdoctoral Associate supported by ONR and NSF. Anshumali Shrivastava is a Ph.D. student supported by ONR and NSF. The authors thank Yanen Li for helpful discussions.

8. REFERENCES

[1] Abernethy, J., Bartlett, P., and Rakhlin, A. Multitask learning with expert advice. In *COLT'07* (2007), vol. 4539 of *Lecture Notes in Computer Science*, Springer, pp. 484–498.

[2] Agarwal, A., Rakhlin, A., and Bartlett, P. Matrix regularization techniques for online multitask learning. Tech. Rep. UCB/EECS-2008-138, University of California, Berkeley, Oct 2008.

[3] Ahmad, F., and Kondrak, G. Learning a spelling error model from search query logs. In *HLT-EMNLP'05* (October 2005), pp. 955–962.

[4] Ando, R. K., and Zhang, T. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research 6* (2005), 1817–1853.

[5] Argyriou, A., Micchelli, C. A., Pontil, M., and Ying, Y. A spectral regularization framework for multi-task structure learning. In *Proceedings of NIPS'07* (2007), MIT Press.

[6] Benveniste, A., Metivier, M., and Priouret, P. Adaptive algorithms and stochastic approximations. *Berlin: Springer* (1990).

[7] Bottou, L. Online algorithms and stochastic approximations. *Online Learning and Neural Networks. Saad, David. Cambridge University Press* (1998).

[8] Caruana, R. Multitask learning. *Machine Learning 28*, 1 (1997), 41–75.

[9] Cavallanti, G., Cesa-Bianchi, N., and Gentile, C. Linear algorithms for online multitask classification. In *COLT'08* (2008), Omnipress, pp. 251–262.

[10] Chen, Q., Li, M., and Zhou, M. Improving query spelling correction using web search results. In *EMNLP-CoNLL'07* (Prague, Czech Republic, June 2007), pp. 181–189.

[11] Church, K. W., and Gale, W. A. Probability scoring for spelling correction. *Statistics and Computing volumn 1* (1991), 93–103.

[12] Cucerzan, S., and Brill, E. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP'04* (Barcelona, Spain, July 2004), pp. 293–300.

[13] Daumé III, H. Frustratingly easy domain adaptation. In *ACL'07* (Prague, Czech Republic, June 2007), pp. 256–263.

[14] Dekel, O., Long, P. M., and Singer, Y. Online multitask learning. In *COLT'06* (2006), vol. 4005 of *Lecture Notes in Computer Science*, Springer, pp. 453–467.

[15] Duan, H., and Hsu, B.-J. P. Online spelling correction for query completion. In *WWW'11* (2011), ACM, pp. 117–126.

[16] Evgeniou, T., Micchelli, C. A., and Pontil, M. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research 6* (2005), 615–637.

[17] Ganjisaffar, Y., Zilio, A., Javanmardi, S., Cetindil, I., Sikka, M., Katumalla, S. P., Khatib-Astaneh, N., Li, C., and Lopes, C. qSpell: Spelling correction of web search queries using ranking models and iterative correction. In *Spelling Alteration for Web Search Workshop* (July 2011).

[18] Gao, J., Li, X., Micol, D., Quirk, C., and Sun, X. A large scale ranker-based system for search query spelling correction. In *COLING'10* (2010), pp. 358–366.

[19] Golding, A. R., and Roth, D. Applying winnow to context-sensitive spelling correction. In *ICML'96* (1996), Morgan Kaufmann, pp. 182–190.

[20] Hsu, C.-N., Huang, H.-S., Chang, Y.-M., and Lee, Y.-J. Periodic step-size adaptation in second-order gradient descent for single-pass on-line structured learning. *Machine Learning 77*, 2-3 (2009), 195–224.

[21] Kernighan, M. D., Church, K. W., and Gale, W. A. A spelling correction program based on a noisy channel model. In *COLING'90* (1990), pp. 205–210.

[22] Kukich, K. Techniques for automatically correcting words in text. *ACM Comput. Surv. 24*, 4 (1992), 377–439.

[23] Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML'01* (2001), pp. 282–289.

[24] Lawrence, N. D., and Platt, J. C. Learning to learn with the informative vector machine. In *ICML'04* (2004), vol. 69, ACM.

[25] Li, M., Zhu, M., Zhang, Y., and Zhou, M. Exploring distributional similarity based models for query spelling correction. In *COLING-ACL'06* (Sydney, Australia, July 2006), pp. 1025–1032.

[26] Li, P. and Konig, A.C. Theory and Applications of b-bit Minwise Hashing. *Communications of the ACM 54* (2011), 101–109.

[27] Li, P., Shrivastava, A., Moore, J., and Konig, A.C. Hashing Algorithms for Large-Scale Learning. In *NIPS'11* (Granada, Spain, December 2011).

[28] Li, Y., Duan, H., and Zhai, C. Cloudspeller: Spelling correction for search queries by using a unified hidden markov model with web-scale resources. In *Spelling Alteration for Web Search Workshop* (July 2011).

[29] Mangu, L., and Brill, E. Automatic rule acquisition for spelling correction. In *ICML'97* (Nashville, TN, 1997), Morgan Kaufmann, pp. 187–194.

[30] Nocedal, J., and Wright, S. J. Numerical optimization. *Springer* (1999).

[31] Rai, P., and III, H. D. Infinite predictor subspace models for multitask learning. *Journal of Machine Learning Research - Proceedings Track 9* (2010), 613–620.

[32] Reynaert, M. Character confusion versus focus word-based correction of spelling and ocr variants in corpora. *IJDAR 14*, 2 (2011), 173–187.

[33] Saha, A., Rai, P., Daumé III, H., and Venkatasubramanian, S. Online learning of multiple tasks and their relationships. In *AISTATS'10* (Ft. Lauderdale, Florida, 2011).

[34] Sun, X., Gao, J., Micol, D., and Quirk, C. Learning phrase-based spelling error models from clickthrough data. In *ACL'10* (Uppsala, Sweden, July 2010), pp. 266–274.

[35] Sun, X., Kashima, H., Tomioka, R., Ueda, N., and Li, P. A new multi-task learning method for personalized activity recognition. In *ICDM'11* (2011), IEEE.

[36] Sun, X., Shrivastava, A., and Li, P. Query spelling correction using multi-task learning. In *WWW (Companion Volume)* (2012), A. Mille, F. L. Gandon, J. Misselis, M. Rabinovich, and S. Staab, Eds., ACM, pp. 613–614.

[37] Toutanova, K., and Moore, R. Pronunciation modeling for improved spelling correction. In *ACL'02* (Philadelphia, USA, 2002), pp. 144–151.

[38] Wang, Z., Xu, G., Li, H., and Zhang, M. A fast and accurate method for approximate string search. In *ACL'11* (2011), pp. 52–61.

[39] Whitelaw, C., Hutchinson, B., Chung, G., and Ellis, G. Using the web for language independent spellchecking and autocorrection. In *EMNLP'09* (2009), pp. 890–899.

[40] Xue, Y., Dunson, D., and Carin, L. The matrix stick-breaking process for flexible multi-task learning. In *ICML'07* (Corvalis, Oregon, 2007), ACM, pp. 1063–1070.

[41] Yang, H., King, I., and Lyu, M. R. Online learning for multi-task feature selection. In *CIKM'10* (2010), ACM, pp. 1693–1696.

[42] Yu, K., Tresp, V., and Schwaighofer, A. Learning gaussian processes from multiple tasks. In *ICML'05* (2005), vol. 119, ACM, pp. 1012–1019.

[43] Zhang, J., Ghahramani, Z., and Yang, Y. Learning multiple related tasks using latent independent component analysis. In *NIPS'05* (2005).