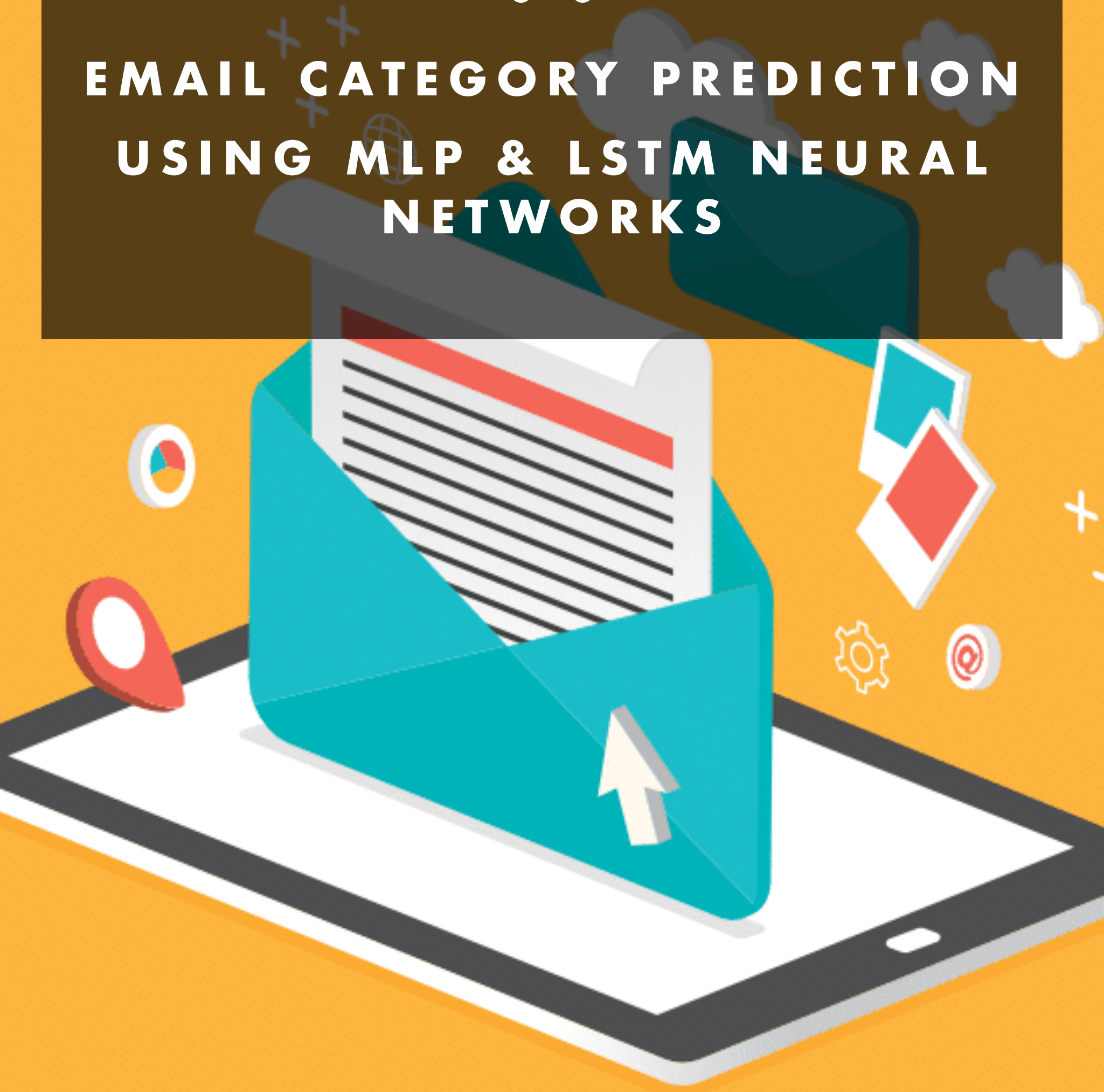
EMAIL CATEGORY PREDICTION USING MLP & LSTM NEURAL NETWORKS



Email Category Prediction

Aston Zhang*[1], Lluis Garcia-Pueyo[2], James B. Wendt[2], Marc Najork[2], Andrei Broder[2]
[1]University of Illinois at Urbana-Champaign, IL, USA, [2]Google, CA, USA
lzhang74@illinois.edu, {lgpueyo, jwendt, najork, broder}@google.com

ABSTRACT

According to recent estimates, about 90% of consumer received emails are machine-generated. Such messages include shopping receipts, promotional campaigns, newsletters, booking confirmations, etc. Most such messages are created by populating a fixed *template* with a small amount of personalized information, such as name, salutation, reservation numbers, dates, etc. Web mail providers (Gmail, Hotmail, Yahoo) are leveraging the structured nature of such emails to extract salient information and use it to improve the user experience: e.g. by automatically entering reservation data into a user calendar, or by sending alerts about upcoming shipments. To facilitate these extraction tasks it is helpful to classify templates according to their *category*, e.g. restaurant reservations or bill reminders, since each category triggers a particular user experience.

Recent research has focused on discovering the *causal thread* of templates, e.g. inferring that a shopping order is usually followed by a shipping confirmation, an airline booking is followed by a confirmation and then by a "ready to check in" message, etc. Gamzu *et al.* took this idea one step further by implementing a method to predict the template category of future emails for a given user based on previously received templates. The motivation is that predicting future emails has a wide range of potential applications, including better user experiences (e.g. warning users of items ordered but not shipped), targeted advertising (e.g. users that recently made a flight reservation may be interested in hotel reservations), and spam classification (a message that is part of a legitimate causal thread is unlikely to be spam).

The gist of the Gamzu *et al.* approach is modeling the problem as a Markov chain, where the nodes are templates or temporal events (e.g. the first day of the month). This paper expands on their work by investigating the use of neural networks for predicting the category of emails that will arrive during a fixed-sized time window in the future. We consider two types of neural networks: multilayer perceptrons (MLP), a type of feedforward neural network; and long short-term memory (LSTM), a type of recurrent neural network. For each type of neural network, we explore the effects of varying their configuration (e.g. number of layers or number of neurons) and hyper-parameters (e.g. drop-out ratio). We find that the prediction accuracy of neural networks vastly outperforms the Markov chain approach, and that LSTMs perform slightly better than MLPs. We offer some qualitative interpretation of our findings and identify some promising future directions.

*The work was completed at Google Research.

©2017 International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC-BY-NC-ND 2.0 License.

WWW 2017 Companion,, April 3–7, 2017, Perth, Austraila.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3055166

CCS Concepts

•Information systems → Email; •Computing methodologies → Machine learning;

Keywords

Email; prediction; time series analysis

1. INTRODUCTION

According to a recent recent study, about 90% of the messages received by Yahoo users are machine-generated [21]. Such messages include shopping receipts, promotional campaigns, newsletters, flight booking confirmations, etc. Virtually all such messages are created by populating a fixed *template* with a small amount of personalized information, such as name, salutation, reservation numbers, dates, etc. Web mail providers (Gmail, Hotmail, Yahoo) are leveraging the structured nature and the volume of such emails to extract salient information from received messages [32], and use it to improve the user experience: e.g. by automatically entering reservation data into a user calendar, or by sending alerts about upcoming shipments.

To facilitate these extraction tasks it is helpful to classify templates according to their *category*, e.g. "hotel reservations" or "bill reminders", since the category of a template determines the information that should be extracted, for instance check-in and check-out dates for a hotel reservation, due date for a bill reminder, and so on. Determining the category of a template can be framed as a multi-class classification problem [12, 28].

Ailon *et al.* [2] observed that an email of a certain category is often followed by a second email of a related category. For example, a purchase confirmation email is often followed by a shipping confirmation containing a tracking number. Ailon *et al.* clustered a sample of emails from Yahoo Mail into templates, classified them into categories, and induced a *causality graph* that captured the temporal sequence of email messages within the same causal thread. The aim of their research was to improve email organization by grouping all emails belonging to the same causal thread into a bundle.

Gamzu *et al.* took the idea of causal threads one step further, by investigating whether it is possible to predict the arrival of future emails – emails generated from the template expected next in an ongoing causal thread [9]. Predicting the template or category of

future emails has a number of possible applications. For example, the web email service could use the prediction as a signal that the user is "in market" for a product or a service and show appropriate contextualized advertisement. The prediction can also be used as a signal for an email spam classifier – an email belonging to a causal thread is almost certainly not spam unless the entire thread is spam. Gamzu *et al.* modeled the problem as a Markov chain, where the nodes are a combination of templates and temporal events created by inferring causal relations (e.g. the first of the month may be predictive of the arrival of a bill reminder email), and the edges are transition probabilities. Using this approach, they were able to predict with about 45% precision that an email belonging to a particular template would arrive within the next two weeks.

This paper expands on Gamzu *et al.'s* work. In particular, we explored whether prediction accuracy could be improved through a more sophisticated model. We investigated the performance of two different types of neural networks: multi-layer perceptrons (MLP), a type of feedforward neural network; and long short-term memory (LSTM), a type of recurrent neural network. LSTMs differ from MLPs in that they have internal memory, making them particularly suitable for time series analysis. We explored the effects of varying the configuration of the neural networks (e.g. the number of layers and the number of neurons), their hyper-parameters (e.g. the LSTM's drop-out ratio) as well as the amount of history provided to each model. We performed experiments on a data set of email category sequences spanning 90 days of activity for about 100,000 anonymized email users. We found that both MLPs and LSTMs vastly outperform the Markov chain baseline, and that LSTMs perform slightly better than MLPs, presumably due to their superior ability for time series analysis afforded by their internal memory. In their best configurations, the Markov chain model yields a mean reciprocal rank of 0.840 while MLP and LSTM yield 0.918 and 0.923, respectively.

The remainder of this paper is structured as follows: Section 2 reviews related work. Section 3 provides a more formal definition of the problem. Section 4 reviews Markov chains, multi-layer perceptrons and long short-term memory. Section 5 describes our experimental evaluation. Finally, Section 6 offers concluding remarks and avenues for future research.

2. RELATED WORK

Email was invented in 1972, about 19 years earlier than the World Wide Web. Nowadays, email is known to host significantly larger user-generated contents than the entire Web, but on the other hand *The New York Times* refers to the email overload as "a $650 billion drag on the economy" [2, 27]. Driven by users' demand for automatic management of the overwhelming volume of emails, research in email "intelligence" has been growing, such as email response prediction [16], email operation prediction [8], email prioritization [26, 27], email threading [2], and email communication control [13].

Email categorization has been studied for more than twenty years. Cohen proposed a rule-based learning algorithm [7], and Provost subsequently suggested naïve Bayesian learning as an alternative [24].

Brutlag & Meek [5] identified the unique technical challenges in the email domain compared to general text categorization problems. The release of the Enron email data set [19] provided the impetus for further research. Recent email categorization techniques are based on graph mining [1, 6], co-training or semi-supervised learning [18, 28], and sampling probability distributions [4]. Koren *et al.* leveraged folder names for categorization [20], while Grbovic *et al.* investigated automatically tagging emails with a small set of categories [12].

Time	Received email category
2016-10-17-16-45	Newsletter
2016-10-17-20-15	Hotel Booking
2016-10-18-09-34	Bill
2016-10-21-07-26	Newsletter
2016-10-21-11-45	Recommended Event
Probability	**Predicted email category**
0.61	Hotel Booking
0.25	Newsletter
0.12	Recommended Event

Table 1: Example of the email category prediction task: Given a series of past email categories received at different timestamps by one user, the system provides prediction probabilities of email categories to be received in the future.

Very recently Gamzu *et al.* investigated whether it is possible to predict the templates of future emails [9]. They construct a causality graph that captures the sequence of observed templates. Given a user having received a chain of templates, they use the causality graph to predict the most likely template to arrive next and achieve 45% precision for this task.

While the techniques proposed by Gamzu *et al.* predict the next template, our work aims to predict the *category* of the next template – a somewhat easier problem, since there are far fewer categories than templates. We investigate the prediction performance of two powerful learning approaches based on deep neural networks and compare it to a baseline approach based on a k-dependent Markov chain, which is similar in spirit to Gamzu *et al.'s ThreadingBasedPredictor*. Deep neural networks, particularly ones designed for time series analysis, turned out to perform far better than Markov chains.

3. EMAIL CATEGORY PREDICTION

In this section we describe and formalize the email category prediction problem.

A machine-generated email is a single instantiation of a predefined structural template. We specifically consider and target machine-generated emails as they constitute a majority percentage of all emails and their categorization can be highly accurate [2, 28]. Recent work by Wendt *et al.* demonstrated that such machine-generated emails can be categorized with a 91% precision and a 93% recall [28]. For succinctness, we refer to a machine-generated email as an email in the rest of the paper.

Consider that a user receives a series of emails. Each email has a timestamp indicating when it was received by the user. We assume that each email is assigned to a pre-defined category by an email categorization method. This assumption is general enough to allow the re-using of existing email categorization research. In this work, the email categorization algorithm deployed at Google is used to generate categories for emails. Examples of categories include flight reservations, restaurant bookings, and event reminders.

Intuitively, a user's received past email categories with temporal information may be leveraged to predict future categories. As illustrated in Table 1, given a series of categories Newsletter, Hotel Booking, Bill, Newsletter, and Recommended Event, the future category could be Hotel Booking as triggered by the latest category Recommended Event and repeating an earlier category Hotel Booking. Or, the future category could be Newsletter since it appeared twice recently, which might suggest the user's subscriptions

to newsletters. Or simply, the future category might just repeat one of the recent categories, such as Recommended Event. Among all such possibilities, the future category may be related to one or many past categories and their temporal information.

More succinctly put, the email categorization problem is to predict, given the categories and timestamps of a user's past emails, a probability distribution over the categories of emails that the user will receive over the next n days.

More formally, suppose that a user has a series of emails at prediction time. Each of these received emails is assigned with a time step index t as 1, 2, ... in the ascending order of time ($t = 0$ is used for describing variable initialization as discussed later). We conduct a one-to-one mapping between category names and category integer IDs. The set of all the pre-defined categories is defined by $\mathcal{A} = \{1, 2, \ldots, |\mathcal{A}|\}$, where $|\mathcal{A}|$ is the cardinality of the category set. We denote an email category as $a \in \mathcal{A}$. An email at time step t has a category $a^{(t)} \in \mathcal{A}$ and a timestamp $m^{(t)}$.

At the prediction time step t, the output of the predictor is a probability distribution indicating for each category a the probability that an email belonging to a will be received during the prediction time window $(m^{(t)}, m^{(t)} + \Delta]$. For instance, the prediction time window size Δ may be set to 1 day, 2 days, and so on. Hence, the output at the prediction time step t is a $|\mathcal{A}|$-dimensional probability vector $\mathbf{y}^{(t)}$ where the a-th element corresponds to the predicted likelihood for category a.

4. PREDICTING THE CATEGORY OF FUTURE EMAILS

In the following, we provide some detail on the three different category prediction approaches we explore, after describing the features that are available to them.

For each user in the training set, we consider the last 90 days of email messages received. We consider only those messages for which Google's category classifier (which we treat as a "black box") was able to assign a category. For each email message in that subset, we record the category of the email and the time at which it was received. We also record some derived information of the time of receipt: the day of the week, the period of the month (beginning, middle or end), and finally the inter-arrival gap to the previous message.

In order to evaluate the different prediction approaches, we partition the data set into two parts: the first 45 days of each user's history go into the training set, while the second 45 days go into test and validation sets.

4.1 k-dependent Markov chains

We use the k-dependent Markov chain [25] as a baseline system. This method is based on counting to calculate the probability of occurrence of an event after the observation of a chain of k consecutive events. Given the list of categories $\langle a^{(1)}, \cdots, a^{(n)} \rangle$ of the n emails received by a user and sorted by received timestamp, we create state pairs:

$$\{(\langle a^{(i)}, \cdots, a^{(i+k-1)} \rangle, \langle a^{(i+1)}, \cdots, a^{(i+k)} \rangle) \; \forall i \in \{1, \cdots, n-k\}$$

The first state is a list capturing a sequence of k received emails, the second state is a suffix of the first list plus the category of the next received email. Using these state pairs, we construct a directed graph, where the nodes are states, the edges indicate the presence of a pair, and the edge weights indicate the probability of a transition from source to target state. One can view this graph as a mapping from source states to a probability distribution over target states, or alternatively (considering only the final category of a target state) over categories.

Prediction is performed as follows: For each user in the test data set, we consider a sequence of k received emails and test whether we can predict the category of the next email. The k emails form a source state, which can be mapped as described above to a probability distribution over categories. This probability distribution forms the output of the predictor.

4.2 Multilayer Perceptrons

Multilayer perceptrons (MLPs), sometimes called the "quintessential deep learning model" [10], are a form of feedforward neural network. Such a network consists of "neurons" that can perform simple operations. Neurons are arranged into "layers". In a feedforward neural network, each neuron at layer l is connected to every neuron at layer $l-1$. The top and bottom layers form the input and output layer, the layers in between are called the hidden layers. Input vectors flow from the input layer to the output layer, passing through weighted connecting edges and being transformed by neurons.

The input layer of the neural network takes a vector of numbers, one value per neuron. We provide a sequence of k received emails as the input, representing the category and time of receipt of each email as a vector and concatenating the k vectors. The individual features of each email are themselves represented as vectors and concatenated to form the email's vector. The individual features of an email received at time step t are encoded as follows:

- We treat the category $a^{(t)}$ as a number between 1 and $|\mathcal{A}|$, and we represent it through an $|\mathcal{A}|$-dimensional vector, with the $a^{(t)}$-th dimension set to 1 and all other dimensions set to 0 (a common technique known as *one-hot encoding*). Embedding features of one-hot vectors has been found useful for training neural networks [29, 30], therefore we embed the $|\mathcal{A}|$-dimensional vector into an a'-dimensional space, resulting in an a'-dimensional vector. The embedding matrix is inferred during the training phase.

- We encode the day of the week at which the email was received as a 7-dimensional binary vector using a similar one-hot encoding, and embed it into a d'-dimensional space.

- We encode the period of the month as a 3-dimensional binary vector, again using one-hot encoding, and embed it into a p'-dimensional space.

- We encode the time gap $m^{(t)} - m^{(t-1)}$ as a real-value, clamped not to exceed one week and normalized to be between 0 and 1.

Using this encoding, the entire sequence of k emails is represented by a vector $\mathbf{h}_0$ of ks values, where $s = a' + d' + p' + 1$ is number of dimensions for a single email.

We chose the neurons to be *rectified linear units* (ReLUs). Each layer l of ReLUs transforms its input vector $\mathbf{h}_{l-1}$ as follows:

$$\mathbf{h}_l = \max(\mathbf{W}_{l-1}\mathbf{h}_{l-1} + \boldsymbol{b}_{l-1}, 0) \tag{4.1}$$

The vector $\mathbf{h}_l$ is the output of the neurons at layer l. The matrix $\mathbf{W}_{l-1}$ encodes the weights of the edges between layers $l-1$ and l; and the bias vector $\boldsymbol{b}_{l-1}$ encodes the threshold at which each ReLU neuron is "activated" (returns a value greater than 0).

For an MLP with L hidden layers, the state information $\mathbf{h}_L$ contains the encoded information about email categories and their temporal information for the sequence of k emails encoded by $\mathbf{h}_0$. The MLP's output layer uses $\mathbf{h}_L$ to produce the probability vector for

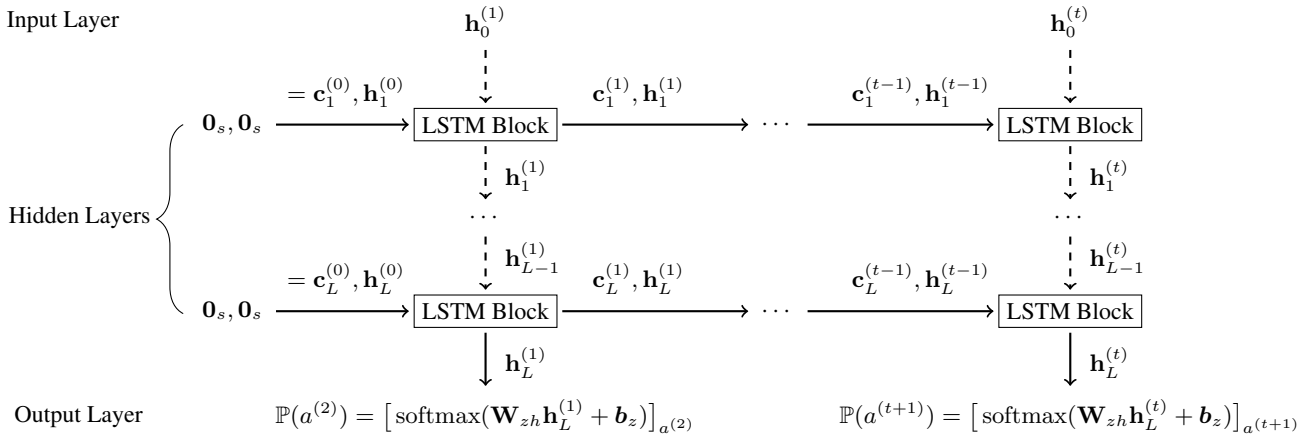


Figure 1: Category prediction LSTM. Dashed arrows between layers indicate connections where dropout regularization is applied.

all the categories as the final output. We define a $|\mathcal{A}|$-dimensional logit vector $\mathbf{z}$ to apply an affine transformation of $\mathbf{h}_L$:

$$\mathbf{z} = \mathbf{W}_{zh}\mathbf{h}_L + \boldsymbol{b}_z \tag{4.2}$$

where weighting matrix $\mathbf{W}_{zh} \in \mathbb{R}^{|\mathcal{A}| \times ks}$ and bias vector $\boldsymbol{b}_z \in \mathbb{R}^{|\mathcal{A}|}$ are decoding parameters that will be inferred during training. With the logit vector defined in (4.2), the output probability vector $\mathbf{y}$ is computed with an element-wise $\mathrm{softmax}$ operator:

$$\begin{aligned}\mathbf{y} &= \mathrm{softmax}(\mathbf{z}) \\ &= \left[\frac{\exp(z_1)}{\sum_{i=1}^{|\mathcal{A}|}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{|\mathcal{A}|}\exp(z_i)}, \ldots, \frac{\exp(z_{|\mathcal{A}|})}{\sum_{i=1}^{|\mathcal{A}|}\exp(z_i)}\right]^\top\end{aligned} \tag{4.3}$$

where z_i is the i-th element of vector $\mathbf{z}$. $\mathbf{y}$ in (4.3) is the probability distribution over categories in $\mathcal{A}$. The probability $\mathbb{P}(a)$ is $[\mathbf{y}]_a$, where the category a is treated as an integer between 1 and $|\mathcal{A}|$ as described earlier.

As described above, an MLP has a number of configuration parameters: the choice of neuron (ReLUs in our case), the number of layers, and the number of neurons at layer l (which is equal to the dimensionality of $\boldsymbol{b}_{l-1}$). The input layer uses embedding parameters for the one-hot encoded vectors, the output layer uses decoding parameters $\mathbf{W}_{zh}$ and $\boldsymbol{b}_z$ in (4.2), and the hidden layers use various parameters $\mathbf{W}$ and $\boldsymbol{b}$ with different subscripts. To infer these parameters during the training phase, we need to formalize the objective function of the training loss. Here we introduce and use notations for users in the training data set. Suppose that a training data set is formed by email series of a set of users $\mathcal{U}$, where each user is denoted by $u \in \mathcal{U}$. User u has a series of T_u emails: there are T_u time steps along the email series for u where each email with category $a_u^{(t)}$ and time stamp $m_u^{(t)}$ is received at time step t. During training, we maximize the likelihood of all the categories that arrive within a Δ time that is equivalent to the prediction window size. It is equivalent to minimize the following training loss (known as perplexity):

$$L = \exp\left[-\frac{\sum_{u\in\mathcal{U}}\sum_{t=1}^{T_u-1}\log\mathbb{P}(a_u^{(t+1)})\mathbf{1}(m_u^{(t+1)} - m_u^{(t)} \le \Delta)}{\sum_{u\in\mathcal{U}}\sum_{t=1}^{T_u-1}\mathbf{1}(m_u^{(t+1)} - m_u^{(t)} \le \Delta)}\right] \tag{4.4}$$

where indicator function $\mathbf{1}$ equals to 1 if its predicate is true, otherwise equals to 0.

To minimize the training loss in (4.4) with respect to parameters $\mathbf{W}$ and $\boldsymbol{b}$ with different subscripts, we use the Adam stochastic optimization algorithm [17]. To compute the gradients $\partial L/\mathbf{W}$ and $\partial L/\boldsymbol{b}$ of L with respect to $\mathbf{W}$ and $\boldsymbol{b}$ for Adam, the truncated backpropagation through time algorithm is used [10, 11]. Examples of the details of computing such gradients of composite functions by applying chain rules are provided by Graves [11].

4.3 Long short-term memory

In feedforward neural networks such as the multilayer preceptron discussed above, edges may not form cycles. Neural networks with cyclic connections are called recurrent neural networks. The most popular of these is the *long short-term memory* (LSTM) [14]. In the past, LSTMs have been used successfully for many sequence prediction tasks, making them a promising candidate for the category prediction problem.

As the name suggests, LSTM neurons have memory, and that memory can be used to store information derived from previous observations. The MLP described in Section 4.2 took a sequence of the last k emails received by a user as input to predict the next email. The LSTM, on the other hand, starts out with a blank memory, processes the emails received by a given user one email at a time, and uses its internal memory to store information about all previous emails received by that user. Once all emails of the user have been processed, the memory is blanked again.

Figure 1 shows a high-level view of an LSTM, with the neurons of each layer abstracted into a "black box". The input layer takes the sequence of emails received by a given user, one email at a time. We encode the category and temporal information of each email at time step t in the same way as we did for multilayer perceptrons, producing a vector $\mathbf{h}_0^{(t)}$ of dimensionality $(a' + d' + p' + 1)$. The input flows through the L hidden layers to the output layer, where it is mapped to a probability distribution over labels in the same way as for the MLP approach.

For an MLP, the output $\mathbf{h}_l$ of layer l depends only on $\mathbf{h}_{l-1}$, the output of layer $l-1$. By contrast, given the email at time step t, the output $\mathbf{h}_l^{(t)}$ of the LSTM block at layer l depends on three factors: the output $\mathbf{h}_{l-1}^{(t)}$ of the LSTM block at layer $l-1$ for that same email, the output $\mathbf{h}_l^{(t-1)}$ of the same LSTM block for the previous email; and the state $\mathbf{c}_l^{(t-1)}$ of the memory cell. In addition to producing an output, the LSTM block also updates updates its memory cell to a new state $\mathbf{c}_l^{(t)}$. This can be expressed as:

$$\mathbf{c}_l^{(t-1)}, \mathbf{h}_{l-1}^{(t)}, \mathbf{h}_l^{(t-1)} \xrightarrow{\text{LSTM Block}} \mathbf{c}_l^{(t)}, \mathbf{h}_l^{(t)} \tag{4.5}$$

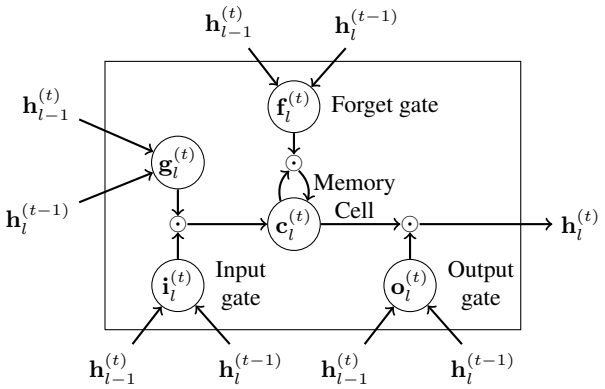


Figure 2: Graphical representation of the used LSTM block.

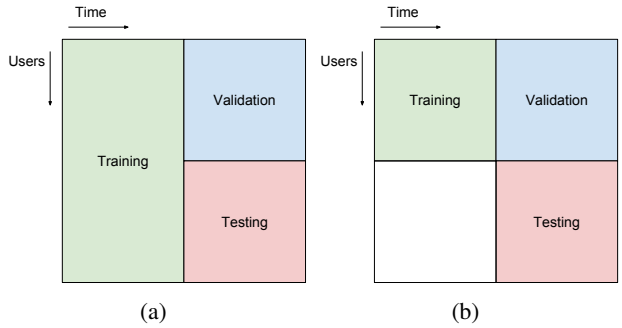


Figure 3: Dataset splits for training, validation, and testing.

where $\mathbf{c}_l^{(t-1)}, \mathbf{h}_{l-1}^{(t)}, \mathbf{h}_l^{(t-1)}, \mathbf{c}_l^{(t)}, \mathbf{h}_l^{(t)}$ are all s-dimensional vectors. Here s is the dimension of a hidden state, which is determined by the dimension of $\mathbf{h}_0^{(t)}$. By initializing $\mathbf{h}_l^{(0)}$, $\mathbf{c}_l^{(0)}$ as

$$\mathbf{h}_l^{(0)}, \mathbf{c}_l^{(0)} = \mathbf{0}_s, \mathbf{0}_s$$

where $\mathbf{0}_s$ is a s-dimensional zero-vector, the LSTM block recurrently updates memory and state information as the time step increases. With such a recurrent LSTM block, at time step t, the state information actually encodes all the captured information by the input layers from the very beginning of an email series until t.

Figure 1 treats each LSTM block as a black box. An LSTM block has mechanisms to enable "memorizing" information for an extended number of time steps. We use the LSTM block decribed in a recent work by Zaremba *et al.* [30], shown in Figure 2. The following transformations are defined for mapping the inputs to the outputs in (4.5):

$$\mathbf{i}_l^{(t)} = \sigma(\mathbf{W}_{it}\mathbf{h}_{l-1}^{(t)} + \mathbf{W}_{il}\mathbf{h}_l^{(t-1)} + \boldsymbol{b}_i), \tag{4.6}$$

$$\mathbf{f}_l^{(t)} = \sigma(\mathbf{W}_{ft}\mathbf{h}_{l-1}^{(t)} + \mathbf{W}_{fl}\mathbf{h}_l^{(t-1)} + \boldsymbol{b}_f), \tag{4.7}$$

$$\mathbf{o}_l^{(t)} = \sigma(\mathbf{W}_{ot}\mathbf{h}_{l-1}^{(t)} + \mathbf{W}_{ol}\mathbf{h}_l^{(t-1)} + \boldsymbol{b}_o), \tag{4.8}$$

$$\mathbf{g}_l^{(t)} = \tanh(\mathbf{W}_{gt}\mathbf{h}_{l-1}^{(t)} + \mathbf{W}_{gl}\mathbf{h}_l^{(t-1)} + \boldsymbol{b}_g), \tag{4.9}$$

$$\mathbf{c}_l^{(t)} = \mathbf{f}_l^{(t)} \odot \mathbf{c}_l^{(t-1)} + \mathbf{i}_l^{(t)} \odot \mathbf{g}_l^{(t)}, \tag{4.10}$$

$$\mathbf{h}_l^{(t)} = \mathbf{o}_l^{(t)} \odot \tanh(\mathbf{c}_l^{(t)}), \tag{4.11}$$

where $\odot$ is an element-wise multiplication operator, and for all $\mathbf{x} = [x_1, x_2, \ldots, x_k]^\top \in \mathbb{R}^k$ the two activation functions $\sigma(\mathbf{x}) = \left[1/[1 + \exp(-x_1)], \ldots, 1/[1 + \exp(-x_k)]\right]^\top$ and $\tanh(\mathbf{x}) = \left[[1 - \exp(-2x_1)]/[1 + \exp(-2x_1)], \ldots, [1 - \exp(-2x_k)]/[1 + \exp(-2x_k)]\right]^\top$. Recall that $\mathbf{h}_0^{(t)} \in \mathbb{R}^s$, the dimensions of the hidden-layer parameters $\mathbf{W}$ and $\boldsymbol{b}$ with different subscripts in (4.6)—(4.9) are characterized as $\mathbf{W} \in \mathbb{R}^{s \times s}$ and $\boldsymbol{b} \in \mathbb{R}^s$. Hence, all the vectors on the left-hand sides of (4.6)—(4.11) have the same dimension s.

In (4.10), the memory cell $\mathbf{c}_l^{(t)}$ stores the "long-term" memory in the vector form. In other words, the information accumulatively captured and encoded until time step t is stored in $\mathbf{c}_l^{(t)}$ and is only passed along the same layer over different time steps. Referring to Figure 2, given the inputs $\mathbf{c}_l^{(t-1)}$, $\mathbf{h}_{l-1}^{(t)}$, $\mathbf{h}_l^{(t-1)}$ as described in (4.5), the input gate $\mathbf{i}_l^{(t)}$ in (4.6) and forget gate $\mathbf{f}_l^{(t)}$ in (4.7) will help the memory cell to decide how to overwrite or keep the memory information. The output gate $\mathbf{o}_l^{(t)}$ in (4.8) further lets the LSTM block decide how to retrieve the memory information to generate the current state $\mathbf{h}_l^{(t)}$ in (4.11) that is passed to both the next layer of the current time step and the next time step of the current layer. Such decisions are made using the hidden-layer parameters $\mathbf{W}$ and $\boldsymbol{b}$ with different subscripts in (4.6)—(4.9): these parameters will be inferred during the training phase.

To reduce overfitting in the training, we adopt the dropout regularization method for recurrent neural networks proposed by Pham *et al.* [22]. With dropout regularization, a random subset of the state $\mathbf{h}_{l-1}^{(t)}$ from the previous layer in (4.6)—(4.9) will be cleared to zero. Note that dropout regularization is only applied on the state passed between layers for the same time step as elucidated as the dashed arrows in Figure 1.

5. EVALUATION

In this section we evaluate and compare the performance of the k-dependent Markov chain, MLP, and LSTM architectures for the email category prediction task.

5.1 Data Description

We use an experimental dataset consisting of 102,743 users and 2,562,361 machine-generated emails spanning 90 days. Each email is represented with its timestamp of receipt and one of 17 categorical labels that define the body contents (e.g. restaurant reservation, online purchase, job listing, etc.). We anonymize the dataset by keeping only the timestamp and category of each email, and discarding all other information (including user data and email contents).

The data is split into training and testing/validation sets by bisecting the data at the 45 day midpoint. The training set comprises the earlier half and the testing/validation sets comprise the later half. The first configuration in Figure 3 ensures that the testing set consists of unseen future emails for a known set of users on which the models have been trained. The second configuration enables evaluation of the models over an unseen set of future emails as well as an unseen set of new users.

Unless stated otherwise, we use a prediction time window of 3 days in our experiments. The effects of varying the prediction time window sizes are studied in Section 5.3.5. Mean values of at least 10 replications of each experiment are reported.

5.2 Metrics

Mean reciprocal rank (MRR) is a common measure to evaluate the ranking accuracy of a prediction [3, 31]. We define the *best rank* of a prediction to be the position of the first correctly predicted category in the prediction output when sorted by decreasing probability. If the category with the highest predicted probability is

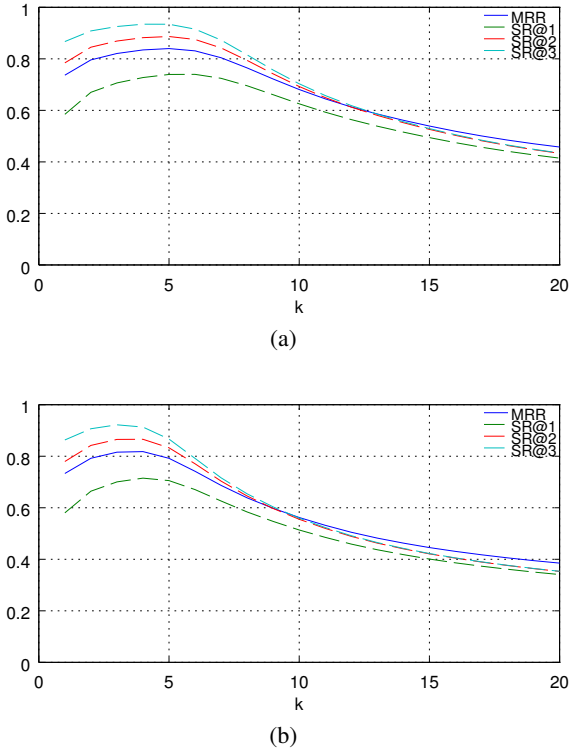


Figure 4: Markov chain model MRR and SR@r. (a) Training and testing sets differ in time period, not user base. (b) Training and testing sets differ by time period and user base.

Table 2: MRR for varying configurations of 2-layer MLPs trained using $k = 1$ past emails. Tested on new users. The columns correspond to the number of neurons in the first layer, n_1. The rows correspond to the number of neurons in the second layer, n_2.

	8	16	32	64	127	256
8	0.8562	0.8555	0.8454	0.8525	0.8529	0.8467
16	0.8615	0.8636	0.8638	0.8627	0.8613	0.8614
32	0.8628	0.8644	0.8644	0.8623	0.8637	0.8635
64	0.8636	0.8626	0.8644	0.8638	0.8644	0.8646
128	0.8627	0.8630	0.8638	0.8644	0.8642	0.8637
256	0.8633	0.8632	0.8644	0.8646	0.8645	0.8643

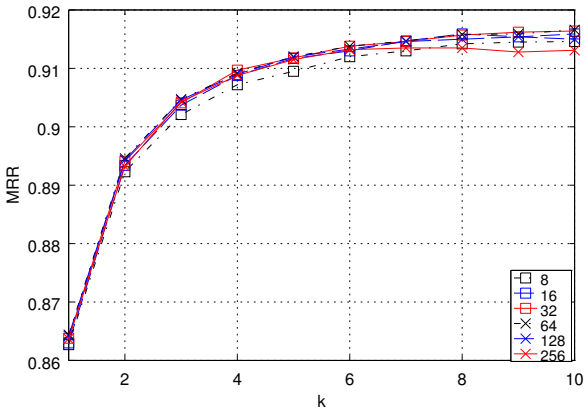


Figure 5: MRR for varying configurations of 2-layer MLPs with 256 neurons in the second layer. Tested on new users. The legend corresponds to the number of neurons in the first layer, n_1.

indeed observed within the prediction time window, the best rank is 1. For our email category prediction task we compute the MRR using the best rank of a predicted category of any email that is received within the prediction time window.

The success rate at r (SR@r) is the proportion of testing examples where the best rank is smaller than or equal to r. For example, a success rate of 0.9 for SR@2 implies that 90% of testing examples had a best rank of either 1 or 2. SR@r is also used to evaluate the ranking accuracy for prediction [15, 31].

In general, a higher MRR or SR@r indicates higher accuracy in email category prediction.

5.3 Experimental Results

5.3.1 k-dependent Markov chain

Figure 4 shows that the MRR of the baseline k-dependent Markov chain peaks at 0.840 ($k = 5$) and 0.818 ($k = 4$) for the respective dataset split configurations described in Figure 3.

Memory information is effectively encoded into the Markov chain model by training on k past categories where $k > 1$. While an immediate increase in k improves overall performance, at $k > 5$ the input space grows to 24+ million possible encodings, which is 10 times the number of training examples in our dataset, eventually deteriorating the model's performance to even worse than at $k = 1$. Increasing the size of the training input will theoretically improve overall performance as the model observes more training examples for each embedding, however this is ultimately an intractable solution since improving the performance at k requires an exponential increase in training input for each increase in k.

It should be pointed out that the Markov chain approach is handicapped compared to the other two approaches, since it uses only the categories but not the temporal features of past emails.

5.3.2 MLP

The MLP architecture provides a substantial improvement over the k-dependent Markov chain. The best configured MLP predicts the next email category with an average MRR of 0.918 and 0.917 for the respective dataset configurations described in Figure 3.

To find this best hyper-parameter configuration we perform a parameter sweep over the number of hidden layers, the number of neurons at each hidden layer, and the number of past emails used to train the model. Table 2 and Figure 5 present slices of the configuration results for the 2-layer architecture. Our first observation is that increasing the number of hidden layers above 2 provides no further gains in the performance metrics. In fact, the number of neurons at each layer has a relatively small impact on performance past 8 neurons in either layer.

Since the sizes of either layer have a relatively similar impact on the performance of the network, Figure 5 can then provide us with insight into the impact of the number of past emails, k, used in training. Our first observation is that increasing the amount of past history used as input to the MLP from $k = 1$ to $k = 4$ provides substantial improvements. However, the marginal benefits diminish as k increases.

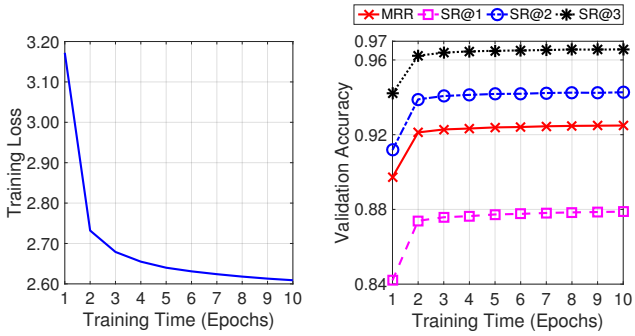


Figure 6: Example LSTM training loss and validation accuracy.

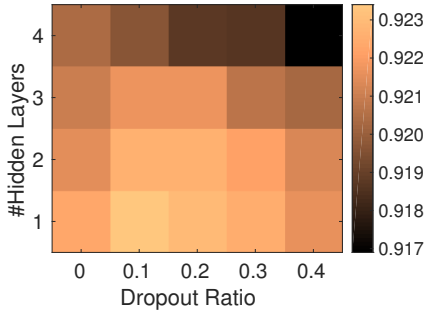


Figure 7: MRR for various configurations of LSTM hyper-parameters.

5.3.3 LSTM

The performance improvement of LSTM over MLP is statistically significant, albeit small, predicting an average MRR of 0.923 and 0.922 for the dataset configurations described in Figure 3.

An example of the training loss (recall equation (4.4)) and validation accuracy across training epochs is depicted in Figure 6. Early stopping is suggested to prevent overfitting when stable results are observed on the validation data [23]. Thus, we limit training to 10 epochs in our experiments.

To find the best configuration for the LSTM architecture we perform a sweep across hyper-parameters including the number of hidden layers and dropout ratio. Recall from Section 4.3 that dropout regularization clears a random subset of the state $\mathbf{h}_{l-1}^{(t)}$ from the previous layer in (4.6)—(4.9). The dropout ratio is the proportion of states that are zeroed.

Figure 7 depicts the results of our parameter sweep. Note that some memory loss via a positive dropout ratio improves the performance of the LSTM network by reducing the potential for overfitting. Also note that a single hidden layer is sufficient in our case to achieve our optimal results. This confirms that the LSTM architecture is able to capture the complexity of sequential patterns that traditional feedforward networks require multiple layers and wider layers to model.

LSTM is known for its application to sequential prediction problems such as for natural language processing and time series prediction. In our first experiments, we only provided the model with sequences of email categories and excluded the detailed timestamp information and derivatives (e.g. time of week). The results of these experiments were comparable to the 1-dependent Markov chain, thus, they have been omitted here for brevity. In summary, while LSTM is able to "memorize" sequences, it still requires explicit time-based input features if those sequences have a dependence on time, not just order.

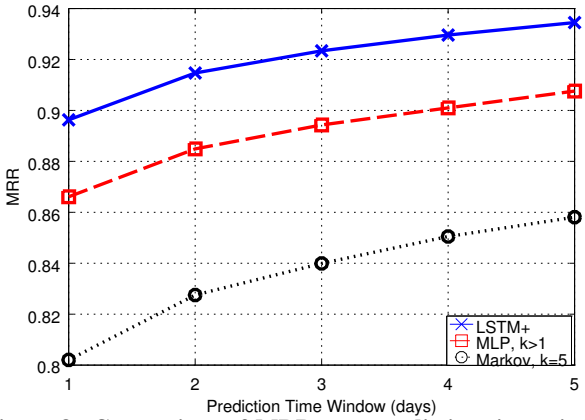


Figure 8: Comparison of MRR over prediction time window size for memory-imparied and memory-enabled Markov chain, MLP, and LSTM models.

5.3.4 Model Comparison

Finally, we compare the results of MRR and SR@r for two configurations of each of the Markov chain, MLP, and LSTM models in Tables 3 and 4. Each model is configured once using minimal memory capabilities—i.e. $k = 1$ for Markov chain and MLP, and clearing the memory cell to zero in LSTM—and one *best* configuration established via parameter sweeping.

While each memory-enabled model performs better than its minimal memory counterpart, LSTM performs the best overall in both cases.

Another important observation is that the performance of all models suffer when tested on unseen users versus trained and tested on the same user set. While this is expected, the majority of the models are relatively resilient to the differences between the two configurations. Only the 5-dependent Markov model suffered a significant reduction in performance.

5.3.5 Prediction Time Window Size

Figure 8 depicts the change in performance metrics for varying configurations of the Markov chain, MLP, and LSTM models as we change the prediction time window size. As expected, when the prediction time window increases, the problem becomes less challenging and the MRR of each model increases. Notably, the advantage of the LSTM over weaker methods becomes less obvious.

6. CONCLUSION

Email categorization has been extensively studied but their outputs have not been fully utilized for important problems such as improving ads quality and improving users' email experiences. The goal of this research is to study the email category prediction problem: given the categories and timestamps of user's past emails, we want to predict the probability distribution over the categories of emails that the user will receive in the next n days.

We considered three different techniques for predicting the categories of future emails: a k-dependent Markov chain (capturing the last k emails received by the user) as a baseline; a multi-layer per-

Table 3: Comparison of configurations of the Markov chain, MLP, and LSTM models. k refers to the number of past emails used in training. n_l refers to the number of neurons in hidden layer l.

Model	MRR		SR@1		SR@2		SR@3	
	Mean	±Std.	Mean	±Std.	Mean	±Std.	Mean	±Std.
Markov chain ($k = 1$)	0.7366	0.00004	0.5846	0.00008	0.7844	0.00007	0.8671	0.00005
Markov chain ($k = 5$)	0.8399	0.00005	0.7394	0.0001	0.8867	0.00005	0.9343	0.00004
MLP ($n_1 = 32, n_2 = 256, k = 1$)	0.8652	0.0005	0.7968	0.0010	0.8772	0.0003	0.9152	0.0002
MLP ($n_1 = 32, n_2 = 256, k = 10$)	0.9182	0.0002	0.8678	0.0003	0.9381	0.0002	0.9637	0.0001
LSTM*	0.8672	0.0002	0.8012	0.0003	0.8760	0.0003	0.9147	0.0002
LSTM†	0.9229	0.0003	0.8737	0.0005	0.9435	0.0003	0.9686	0.0002

* 1-layer memory-impaired LSTM in which both memory cell and hidden state from previous time steps are cleared to zero.
† 1-layer LSTM with dropout ratio of 0.1.

Table 4: Predicting categories of future emails on new users only.

Model	MRR		SR@1		SR@2		SR@3	
	Mean	±Std.	Mean	±Std.	Mean	±Std.	Mean	±Std.
Markov chain ($k = 1$)	0.7331	0.0004	0.5803	0.0005	0.7796	0.0007	0.8632	0.0004
Markov chain ($k = 5$)	0.7919	0.0001	0.7054	0.0002	0.8328	0.0002	0.8666	0.0002
MLP ($n_1 = 32, n_2 = 256, k = 1$)	0.8642	0.0003	0.7963	0.0007	0.8739	0.0006	0.9137	0.0004
MLP ($n_1 = 32, n_2 = 256, k = 10$)	0.9165	0.0003	0.8655	0.0005	0.9365	0.0002	0.9620	0.0003
LSTM*	0.8653	0.0003	0.7991	0.0004	0.8722	0.0006	0.9132	0.0003
LSTM†	0.9220	0.0003	0.8726	0.0004	0.9423	0.0003	0.9676	0.0002

ceptron as a prototypical feedforward neural network, and a long short-term memory as a prototypical recurrent neural network. We compared these techniques experimentally, using categorized historical emails of about 100,000 anonymized email users. We explored the effects of providing more or less history, different neural network configurations (varying the number of layers and the number of neurons), and the impact of hyper-parameter settings (e.g. drop-out ratio for LSTMs). We found that both types of neural networks substantially outperform k-dependent Markov chains, that LSTMs slightly outperform perceptrons (due to their aptitude for time series analysis afforded by their internal state). Under the best configuration, our success rate at 1 is 0.8737 – meaning that for 87.37% of the predictions, an email of the predicted top category will indeed arrive within the next 3 days.

Regarding future work, one avenue for future research is to investigate whether prediction accuracy can be improved by providing more features of past emails, beyond the current category and temporal features. We also think that features from other emails (e.g. those without an assigned category) could further improve prediction performance. Another suitable research direction is the exploration of email chains beyond predetermined email categories to infer new labels and user needs.

7. REFERENCES

[1] M. Aery and S. Chakravarthy. eMailSift: Email classification based on structure and content. In *5th IEEE International Conference on Data Mining*, pages 1–8, 2005.

[2] N. Ailon, Z. S. Karnin, E. Liberty, and Y. Maarek. Threading machine generated email. In *6th ACM International Conference on Web Search and Data Mining*, pages 405–414, 2013.

[3] Z. Bar-Yossef and N. Kraus. Context-sensitive query auto-completion. In *20th International Conference on World Wide Web*, pages 107–116, 2011.

[4] P. Bermejo, J. A. Gámez, and J. M. Puerta. Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Systems with Applications*, 38(3):2072–2080, 2011.

[5] J. D. Brutlag and C. Meek. Challenges of the email domain for text classification. In *8th International Conference on Machine Learning*, pages 103–110, 2000.

[6] S. Chakravarthy, A. Venkatachalam, and A. Telang. A graph-based approach for multi-folder email classification. In *10th IEEE International Conference on Data Mining*, pages 78–87, 2010.

[7] W. W. Cohen. Learning rules that classify e-mail. In *1996 AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25, 1996.

[8] D. Di Castro, Z. Karnin, L. Lewin-Eytan, and Y. Maarek. You've got mail, and here is what you could do with it!: Analyzing and predicting actions on email messages. In *9th ACM International Conference on Web Search and Data Mining*, pages 307–316, 2016.

[9] I. Gamzu, Z. Karnin, Y. Maarek, and D. Wajc. You will get mail! Predicting the arrival of future email. In *24th International Conference on World Wide Web*, pages 1327–1332, 2015.

[10] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[11] A. Graves. Neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 15–35. Springer, 2012.

[12] M. Grbovic, G. Halawi, Z. Karnin, and Y. Maarek. How many folders do you really need?: Classifying email into a handful of categories. In *23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 869–878, 2014.

[13] R. Gupta, G. Liang, H.-P. Tseng, R. K. Holur Vijay, X. Chen, and R. Rosales. Email volume optimization at LinkedIn. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2016.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] J.-Y. Jiang, Y.-Y. Ke, P.-Y. Chien, and P.-J. Cheng. Learning user reformulation behavior for query auto-completion. In

37th International ACM SIGIR Conference on Research & Development in Information Retrieval, pages 445–454, 2014.

[16] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, L. Lukács, M. Ganea, P. Young, and V. Ramavajjala. Smart reply: Automated response suggestion for email. In *22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 495–503, 2016.

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint:1412.6980*, 2014.

[18] S. Kiritchenko and S. Matwin. Email classification with co-training. In *2001 Conference of the Center for Advanced Studies on Collaborative Research*, page 8, 2001.

[19] B. Klimt and Y. Yang. The Enron corpus: A new dataset for email classification research. In *15th European Conference on Machine Learning*, pages 217–226, 2004.

[20] Y. Koren, E. Liberty, Y. Maarek, and R. Sandler. Automatically tagging email by leveraging other users' folders. In *17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 913–921, 2011.

[21] Y. Maarek. Is mail the next frontier in search and data mining? In *9th ACM International Conference on Web Search and Data Mining*, pages 203–203, 2016.

[22] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *14th International Conference on Frontiers in Handwriting Recognition*, pages 285–290, 2014.

[23] L. Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[24] J. Provost. Naïve-bayes vs. rule-learning in classification of email. *University of Texas at Austin*, 1999.

[25] W. J. Stewart. *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*. Princeton University Press, Princeton, NJ, USA, 2009.

[26] B. Wang, M. Ester, J. Bu, Y. Zhu, Z. Guan, and D. Cai. Which to view: Personalized prioritization for broadcast emails. In *25th International Conference on World Wide Web*, pages 1181–1190, 2016.

[27] B. Wang, M. Ester, Y. Liao, J. Bu, Y. Zhu, Z. Guan, and D. Cai. The million domain challenge: Broadcast email prioritization by cross-domain recommendation. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1895–1904, 2016.

[28] J. B. Wendt, M. Bendersky, L. Garcia-Pueyo, V. Josifovski, B. Miklos, I. Krka, A. Saikia, J. Yang, M.-A. Cartright, and S. Ravi. Hierarchical label propagation and discovery for machine generated email. In *9th ACM International Conference on Web Search and Data Mining*, pages 317–326, 2016.

[29] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu. Recurrent neural networks for language understanding. In *14th Annual Conference of the International Speech Communication Association*, pages 2524–2528, 2013.

[30] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint:1409.2329*, 2014.

[31] A. Zhang, A. Goyal, R. Baeza-Yates, Y. Chang, J. Han, C. A. Gunter, and H. Deng. Towards mobile query auto-completion: An efficient mobile application-aware approach. In *25th International Conference on World Wide Web*, pages 579–590, 2016.

[32] W. Zhang, A. Ahmed, J. Yang, V. Josifovski, and A. J. Smola. Annotating needles in the haystack without looking: Product information extraction from emails. In *21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2257–2266, 2015.