Email Volume Optimization at LinkedIn

Rupesh Gupta, Guanfeng Liang, Hsiao-Ping Tseng, Ravi Kiran Holur Vijay, Xiaoyu Chen, Rómer Rosales
LinkedIn Corporation
Mountain View, CA, USA
{rugupta, gliang, htseng}@linkedin.com, rvijay@pinterest.com, {xochen, rrosales}@linkedin.com

ABSTRACT
Online social networking services distribute various types of messages to their members. Common types of messages include news, connection requests, membership notifications, promotions and event notifications. Such communication, if used judiciously, can provide an enormous value to members thereby keeping them engaged. However sending a message for every instance of news, connection request, or the like can result in an overwhelming number of messages in a member’s mailbox. This may result in reduced effectiveness of communication if the messages are not sufficiently relevant to the member’s interests. It may also result in a poor brand perception of the networking service. In this paper we discuss our strategy and experience with regard to the problem of email volume optimization at LinkedIn. In particular, we present a cost-benefit analysis of sending emails, the key factors to administer an effective volume optimization, our algorithm for volume optimization, the architecture of the supporting system and experimental results from online A/B tests.

Keywords
Machine learning, optimization, email

1. INTRODUCTION
LinkedIn and other online social networking services provide members with an easy way to connect and exchange information. They facilitate exchange of information by giving members the opportunity to send private messages to other members, upload and share rich media with other members, self-organize into interest groups around a subject matter and ask/answer questions, subscribe to channels delivering news on topics of interest, etc. The networking services help members keep up with this information through pull-model based in-app services like feed and in-app notifications, as well as push-model based content distribution services like email and push notifications. Conventionally, members can subscribe to receive email communication regarding news, connection requests, membership and event notifications. At times the social networking service may also use the email communication channel for the purpose of promotions or product marketing. Such email communication is of special interest to a networking service because of its potential in providing valuable information to its members without requiring them to be actively logged in to the mobile or web applications of the networking service. Email communication, if used judiciously, can provide an enormous value to members thereby keeping them engaged. This engagement is in turn highly valuable to the networking service. However, sending an email for every instance of news, connection request, or the like can result in a member receiving an overwhelming number of email messages. This can hamper the effectiveness of communication if the email messages are not sufficiently relevant to the member’s interests, as the member might start to indiscriminately disregard all messages received from the networking service (including the important ones).

For a social networking service, email communication can help drive mid and long-term member engagement. Member engagement often relates to members visiting the mobile or web applications of the networking service more often. However since optimizing for member visit frequency is often difficult due to long-term feedback loops, proxies that are readily measurable with little feedback delay are often optimized in practice. Most of the industrial research and development in the area of email optimization has been in the context of driving up click-through rates of email marketing campaigns [1]. These approaches were inadequate for our problem setting.

Email communication from LinkedIn to members comprises of several different types of emails, with each type corresponding to a product vertical as illustrated in Figure 1. Some sample email templates are shown in Figure 2. As highlighted in the email template of Figure 2(b), LinkedIn allows members to control their communication experience of a particular type of email through the change frequency or the unsubscribe options on any email of that type. Alternatively, members can personalize their communication experience of all types of emails through the communication settings panel under the user account settings tab on the web or mobile application, as shown in Figure 3. In this panel, for each type of email, a member can choose to turn off communication completely, or to receive communication at event-occurrence, daily, weekly or recommended cadence. The recommended setting is designed to enable the member to receive email communication regarding only the
most relevant information in a timely manner. LinkedIn optimizes the communication experience of the member when the member opts-in to this setting.

Optimizing the communication experience of a member entails optimization on several fronts, such as, volume optimization, delivery time optimization, frequency control and channel selection. Each of these problems pose unique challenges which merit careful attention to detail. One of these problems, namely volume optimization, is discussed at length in this paper.

Since email communication can drive member engagement for a networking service, naive reduction of email volume can have adverse effects on the growth of the service. Hence, intelligent algorithms are required to identify the emails which may be dropped. The decision to send or drop an email needs be based on the likelihood of the targeted user deriving a high value from the email. We can indirectly infer this value based on the user’s reaction to the email. The user may interact positively with the email by either viewing the email or also by clicking a link within the email which will typically bring her to the application. Alternatively, the user may react negatively to the email by either ignoring the email, or by clicking on the unsubscribe option within the email, or by reporting the email as spam to her email service provider. Ideally the networking service would like to cut down on email volume in a way such that the positive interactions are maximized and negative interactions are minimized. It is evident that this would require optimizing across multiple different objectives. This paper is about the challenges, approaches, and experiences relating to email volume optimization at LinkedIn. Our contributions are summarized below:

2. RELATED WORK

Whittaker et al. [2] published the first user experience study on email overload in 1996. The study exposed the problem of information management, with the subjects finding it challenging to organize their cluttered inboxes and identifying important emails. Sixteen years later, Grevet et al. [3] conducted a quantitative analysis of users’ inboxes on Gmail and concluded that large email volume and poor email organization continue to remain problems. In a unique study published in [4], the authors were able to quantitatively establish the advantages of cutting down emails in a workplace. These included improved work quality and lower stress levels, among others.

In light of the above, several ideas and tools have been conceived to help users manage email overload. Some of the most promising research has been targeted at automated classification of emails into different priority levels or content categories. Early attempts at email prioritization can be found in [5] where the authors train a linear classifier on a small email corpus, using only email features such as text, message length, etc. More recently, Yoo et al. [6] have proposed methods for a more personalized email prioritization where the feature vector is enriched with personal features inferred from the social graph. Lu et al. [7] on the other hand make use of explicit personal features in the form of a user’s interaction history with messages.

While email prioritization tries to address email overload from the recipient side, email volume optimization is an attempt to address email overload from the sender side. If the sender had a mechanism to estimate the priority of each email message for its targeted recipient then the sender could decide to drop the messages with low priority, as the recipient is likely to ignore those emails anyway. However the decision to drop messages in the recipient’s interest can conflict
with the sender’s interest. Hence, email volume optimization requires the sender to make a trade-off between multiple different objectives. We have not been able to trace any published literature on email volume optimization per se, but there is literature demonstrating effectiveness of multi-objective optimization in various problems pertaining to web applications. Agarwal et al. have employed multi-objective optimization for personalized click shaping in online content recommendation in [8] and for displaying promotional widgets on web applications under business constraints in [9].

3. COST-BENEFIT ANALYSIS

In order to empirically examine the cost and benefit of emails for LinkedIn, we set up a small user bucket as follows. For each member in this bucket, we selected a random number in [0, 1] as the probability with which each email message generated for this member was dropped. We denote this bucket as the random-drop bucket. The remaining users received all emails generated for them. We denote this bucket as the send-all bucket.

We collected data from this experiment over a period of one week. Over this period, a large number of unique users in the random-drop bucket received several different types of emails. Each type of email contained information corresponding to a particular product-vertical and contained at least one link, clicking on which would bring the user to a page under the product-vertical on the mobile or web application. Important user interactions with emails, such as email view, email click, unsubscribing from an email type and reporting an email as spam were tracked. In addition, user sessions were tracked and categorized into two categories, viz., downstream (email triggered) and organic, based on the following attribution model. Any session that a user started within 15 minutes of clicking a link within an email was attributed to that email, and categorized as an email triggered downstream session. Any page view and in-app action performed within a downstream session, such as profile view, job view, search, invitations-to-connect, feed like/comment/share, etc., was also categorized as downstream. All remaining sessions were categorized as organic.

3.1 Benefit of Sending Email

Emails were found to drive a significant percentage of in-app engagement. As tabulated in Table 3.1, we observed 2.6% less page views from members in the random-drop bucket compared to the members in the send-all bucket. In other words, members who received all email messages generated for them performed 2.6% more page views than the members who randomly received about half of the email messages which were generated for them. Similarly, a substantial loss in page views was observed for pages under various product verticals.

<table>
<thead>
<tr>
<th>Product-Vertical</th>
<th>Delta in Page Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>-2.6%</td>
</tr>
<tr>
<td>Homepage</td>
<td>-1.4%</td>
</tr>
<tr>
<td>Jobs</td>
<td>-4%</td>
</tr>
<tr>
<td>Profile</td>
<td>-4.5%</td>
</tr>
<tr>
<td>PYMK</td>
<td>-4.5%</td>
</tr>
<tr>
<td>Search</td>
<td>-4%</td>
</tr>
</tbody>
</table>

Table 1: Delta in page views for various product-verticals. random-drop vs. send-all

Figure 4: % increase in active members with increase in emails sent to member

Figure 5: % increase in downstream page views by a member with increase in emails sent to member

To understand this better we performed a fine grained analysis of members in the random-drop bucket. We divided the members into four segments based on the frequency of their visits to the mobile or web application before the start of the experiment. These four segments were: daily-active (visited everyday), weekly-active (visited once a week), monthly-active (visited once a month) and dormant (remaining). This segmentation is also meaningful from another point of view: about the same number of emails were being generated for members within a given segment. No emails were being generated for a vast majority of members within the dormant segment. As shown in Figure 4, it was found that the number of active members (who visited the application at least once over the course of the experiment) increased with an increase in the number of email messages sent to members. For example, from the plot for monthly-active members, we can say that if \( x \) of these members are active when no emails are sent to these members, then more than \( 2x \) members will be active if a large number of emails are sent to these members. This trend is evident for all member segments except the daily-active segment, in which 100% of the members visit organically without the need for any email triggers. However even within this segment members were found to perform more downstream page views with an increase in the number of emails sent, as shown in Figure 5.

To analyze the impact of downstream page views on in-app actions we computed the contribution of downstream
in-app actions towards the overall in-app actions. We found that downstream in-app actions can constitute 10% to 40% of the total in-app actions for certain important actions, such as viewing a job, sending an invitation to connect, etc. These findings are shown in Figure 6. Also shown in the same figure is the percentage of members who performed specific in-app actions only as downstream actions. For example, of all the members who performed action1 on the mobile or web application, about 12% of those performed that action only after clicking a link within an email which brought them to the application. This implies that a significant number of members rely on the receipt of email to carry out certain tasks on LinkedIn.

![Figure 6: Contribution of downstream in-app actions to overall in-app actions](image)

### 3.2 Cost of Sending Email

We found that excessive email communication could have several negative consequences. We observed 45% more number of negative responses to emails in the send-all bucket compared to the random-drop bucket. These negative responses were in the form of clicking the unsubscribe option within an email, or reporting an email as spam to the email service provider. This was not surprising since sending more number of emails is likely to result in more number of responses, positive and negative. However, unlike a positive response to an email, the scope of a negative response to an email does not end at that particular email. If a member clicks the unsubscribe option within an email, we lose the ability to send any emails of that type to that member in the future. If a large number of members report emails from a particular sender as spam to their email service providers, then this can result in an email service provider blocking and filtering all emails from that sender [10]. In general, such deliverability issues are not easy to resolve.

Excessive email communication was also found to result in the members missing important emails. Owing to limited time and attention spans in humans, user attention was found to get divided across important and less important emails. Here is how we discovered this. We picked four types of emails that were designed specifically to serve an important purpose for LinkedIn. We selected members in the random-drop bucket who received an equal number (=4) of these important emails but different number of total emails. For this set of members we plotted the average number of clicks on these important emails by a member against the total number of emails received by that member. This plot is shown in Figure 7 for two groups of members, viz., members for whom a small number of emails were generated, and members for whom a large number of emails were generated. A downward trend is readily visible in both these groups.

Irrelevant email communication (irrelevant to the recipient) can at times also affect the brand perception of a corporation.

### 4. PROBLEM FORMULATION

As established in the previous section, on the one hand email is an important driver of in-app engagement, but on the other hand excessive email can also have severe negative consequences. It goes without saying that the sender needs a way to identify those emails which are important for the sender, and at the same time relevant to the recipient. Then the sender can send only these emails, dropping the rest. In other words, the sender would like to minimize the number of emails sent, in a way so as to maximize the positive outcome and minimize the negative outcome.

In this section, we describe how we formulate email volume optimization at LinkedIn as a Multi-Objective Optimization (MOO) problem in order to obtain optimal trade-off points involving positive and negative outcomes of sending emails. We also describe how the MOO problem is solved and how the solution to the problem is actually used in a production system. But first, we will explain why we think MOO is the right approach.

#### 4.1 Why MOO

For the purpose of simplicity of exposition, let’s consider only one kind of positive consequence and only one kind of negative consequence of each email. Let’s consider a downstream session as the positive outcome and an unsubscribe or reporting of an email as spam (denoting either one as a complaint) as the negative outcome. Then, on one hand we would like to maximize the number of downstream sessions and on the other hand we would like to minimize the number of emails sent and the number of resulting complaints. For each generated email $e$, let’s assume we can obtain estimates for

$$P_{sess}(x_e) = \Pr(\text{downstream session from } e \mid e \text{ is sent})$$

and

$$P_{comp}(x_e) = \Pr(\text{complaint from } e \mid e \text{ is sent})$$

by using email features $x_e$ in the utility prediction models.
from Section 4.3. Since each email \( e \) is targeted to a particular user, we are assuming that \( x_e \) will also include features of the targeted user.

One simple and intuitive approach for achieving our goal could be to rank-order emails based on \( P_{\text{sess}} \) and only send the top emails for which \( T_{\text{sess}} \) is greater than some threshold \( T_{\text{sess}} \). Another simple approach could be to only send those emails for which \( P_{\text{comp}} \) is smaller than a threshold \( T_{\text{comp}} \). In Figure 8 we plot the fraction of sessions and complaints obtained at various threshold values, using these two naive approaches. The points on the curves are computed by simulating the thresholding process with various values of \( T_{\text{sess}} \) and \( T_{\text{comp}} \). Both x and y axes are normalized by the simulation result at the point where no threshold is applied, i.e., the point where no email is dropped.

As we can see from Figure 8(a), thresholding on \( P_{\text{sess}} \) achieves a very nice trade-off between sessions and sends: we could choose a point on the trade-off curve where the send volume is reduced dramatically with little loss in sessions. For example, we could send only 60% of the emails and still maintain roughly 93% of the maximum achievable sessions. However, this thresholding approach achieves a poor trade-off between complaints and sends: complaints are proportional to sends, which is no better than dropping emails at random. On the other hand thresholding on \( P_{\text{comp}} \) achieves a good trade-off between complaints and sends but a poor one between sessions and sends, as shown in Figure 8(b).

Clearly, none of these naive thresholding approaches yields a satisfactory trade-off point where the send volume is reduced with a minimal loss in sessions and a significant drop in complaints, simultaneously. Loosely speaking, an ideal solution would achieve a sessions-sends trade-off similar to the blue curve in Figure 8(a), and a complaints-sends trade-off similar to the red curve in Figure 8(b). More formally, we would like a solution to have a mechanism that can output optimal combinations of sends, sessions and complaints.

4.2 MOO Formulation

In this section we develop on our running example of finding an optimal trade-off between sends, sessions and complaints and present our MOO methodology for email volume optimization. While the following discussion is based on the example of three utilities it should become clear that the formulation presented here readily generalizes to an arbitrary number of utilities of similar nature.

\[ \min \ z \] \[ \text{s.t.} \sum_{e \in E} z_e \geq \alpha_{\text{global}} \times \text{maxGlobalSess} \] \[ \sum_{e \in E} P_{\text{sess}}(x_e) z_e \geq \alpha_t \times \text{maxLocalSess}_t, \ \forall t \] \[ \sum_{e \in E} P_{\text{comp}}(x_e) z_e \leq \beta_{\text{global}} \times \text{maxGlobalComp} \] \[ \sum_{e \in E_t} P_{\text{comp}}(x_e) z_e \leq \beta_t \times \text{maxLocalComp}_t, \ \forall t \] \[ 0 \leq z_e \leq 1 \ \forall e \]

In this formulation:

• The objective \( \sum_{e \in E} z_e \) in (1) is the expected number of emails sent under serving plan \( z \), across all email types.

• The first constraint (2) specifies the target for the global session count. On the left hand side, \( \sum_{e \in E} P_{\text{sess}}(x_e) z_e \) is the expected session count under serving plan \( z \) across all email types. On the right hand side,

\[ \text{maxGlobalSess} = \sum_{e \in E} P_{\text{sess}}(x_e) \]

is the expected session count if all emails in \( E \) are sent and \( \alpha_{\text{global}} \in [0, 1] \) is the fraction of this maximum achievable session count that we aim to retain.

It is worth pointing out that in expressing the total session count as a summation over the session probabilities of individual emails we are making an independence assumption between emails. In other words, we are making an assumption that the send/drop decision for an email does not affect the probability of a session from any other email. This may be a strong assumption to make in certain scenarios and our future research is targeted at solutions for such scenarios.

• The second constraint (3) specifies the targets for the local number of sessions originating from each email type \( t \). On the left hand side, \( \sum_{e \in E_t} P_{\text{sess}}(x_e) z_e \) is the expected number of sessions originating from type \( t \) emails under serving plan \( z \). On the right hand side,

\[ \text{maxLocalSess}_t = \sum_{e \in E_t} P_{\text{sess}}(x_e) \]
is the expected number of sessions originating from type $t$ emails if all those type $t$ emails are sent, and $\alpha_t \in [0, 1]$ is the fraction of this maximum achievable number of sessions originating from type $t$ emails that we aim to retain.

- The next two constraints (4) and (5) specify the global and local tolerances for complaints, similar to those for sessions.
- The last constraint (6) enforces that $z_e$ is a probability.

It is worth noting here that:

1. Not all constraints (2)-(5) need to be specified. Only the ones corresponding to actual requirements need to be specified. For example, let us say that we have a global sessions target, a strict global complaints tolerance and a strict local sessions target for email type $t = 1$. Then we only need to include the two global constraints and one local constraint corresponding to $t = 1$. In an extreme case if we only care about a global sessions target, then only one constraint (2) needs to be included, and the problem (and its solution) degenerates to the naive approach of thresholding on $P_{sess}$.

2. Although in the example LP formulation above we consider the send volume as the objective and treat sessions and complaints as constraints, this is not a requirement. Depending on the use case any one of the utilities of interest can become the objective and the rest can be specified as constraints. For our example, it is okay to change the objective to maximizing the total sessions $\sum_{e \in E} P_{sess}(x_e)z_e$ and have the total send volume as a constraint: $\sum_{e \in E} z_e \leq \omega_{global} \times \text{maxGlobalSend}$, where $\omega_{global} \in (0, 1]$.

4.2.2 Handling Unseen Emails With Lagrangian Duality

At first glance, solving the above LP may seem trivial: at the beginning of the upcoming week, gather all emails $E$ which will be generated, solve the LP with any standard LP solver, and use the optimal solution obtained to serve the emails in the week. However, this is unrealistic in practice due to the following challenge: we do not know in advance the set of emails $E$ that will be generated in the upcoming week.

Fortunately, we observe that the distribution of our generated emails does not change significantly week over week. So we use the set of emails generated in the past week as a forecast for $E$. However, similarity in distribution does not provide us the solution $z_e$ for every email that will actually be generated in the upcoming week. To that end, we make use of the primal-dual technique introduced in [8].

We first add a quadratic regularization term to the objective to make the problem strongly convex. This allows easy conversion from dual solutions to primal solutions.

$$\min \mathbf{z} \quad \sum_{e \in E} [z_e + \frac{\gamma}{2}(z_e - q)^2]$$

where $q \in [0, 1]$ is some prior on the send probability, and $\gamma > 0$ is a regularization parameter.

We now solve the dual of the primal problem above for the set of emails generated in the past week. Standard QP solvers are unable to handle a problem of our scale due to the large number of variables in the dual problem corresponding to the probability constraints of (6) (of the order of millions at LinkedIn). We employ an in-house large scale implementation of the Operator Splitting algorithm [11] to solve this QP. Let $\mu_{\text{global}}, \mu_t, \nu_{\text{global}}$ and $\nu_t$ be the solutions to the dual problem corresponding to the global session constraint (2), the local session constraints (3), the global complaint constraint (4) and the local complaint constraints (5) respectively. Then $z_e$ for any newly generated email can be efficiently computed on-the-fly without the dual solutions corresponding to (6) using Algorithm 1 from [8]. That algorithm can be condensed into the following equation for our binary (send or drop) decision case. For an email $e$ of type $t$:

$$z_e = \Pi_{[0, 1]} \left( \frac{\mu^*_t P_{sess}(x_e) - \nu^*_t P_{comp}(x_e) - 1}{\gamma} + q \right)$$

where $\mu^*_t = \mu_{\text{global}} + \mu_t$, $\nu^*_t = \nu_{\text{global}} + \nu_t$, $\Pi_{[0, 1]}(\cdot)$ stands for the projection onto $[0, 1]$. If we set $\gamma \rightarrow 0^+$ then the send/drop decision is further simplified to the following deterministic rule. For an email $e$ of type $t$:

$$\text{Send } e \iff \mu^*_t P_{sess}(x_e) - \nu^*_t P_{comp}(x_e) - 1 > 0.$$ (7)

We use this simplified decision rule in our current implementation. This rule requires us to maintain a set of just a few coefficients ($\{\mu^*_t, \nu^*_t\}$) to make a send/drop decision for each individual email. We call these the MOO coefficients.

4.3 Utility Prediction

The solution to the MOO problem above depends on the availability of downstream session prediction $P_{sess}(x_e)$ and complaint prediction $P_{comp}(x_e)$ for each generated email. In this section we outline our process for training response prediction models. This includes data collection, large-scale logistic regression model training and model evaluation.

4.3.1 Training Data Collection

Collecting unbiased training data for our use case is a non-trivial exercise. A naive approach for collecting training data is to reserve a small fraction of user base to always receive all generated emails, bypassing any volume optimization logic. The responses (downstream session/complaint/no-interaction) thus collected could then form the training dataset. However, this method of data collection can suffer from serving bias due to the following reason. A user’s response to an email depends not just on the attributes of that particular email but also on the past experience of the user with similar emails. For example, we have observed that a user’s propensity of clicking on an email is correlated with the number of emails received by the user per week over the past few weeks. The click probability is high if the user received few emails in the previous week, but the correlation becomes weaker if the user consistently received few emails each week over the past four weeks. If the serving scheme is unable to produce instances with varying number of emails sent over the past few weeks then we will not be able to capture user behavior in such scenarios. At LinkedIn, at least one email is generated for a vast majority of active members every week which implies that the naive serving scheme will rarely, if ever, create a scenario where an active member received zero emails in the previous week, although
it is likely to result in a higher probability of engagement with a candidate email this week.

We developed the following serving scheme to explore the feature space and overcome the aforementioned sparsity problem. For every member, we choose a random number in [0, 1] which is the probability with which a message is dropped for this member, denoted as $P_{\text{drop}}$. We refresh $P_{\text{drop}}$ every 4 weeks so as to capture both novelty as well as burn-in effects. As an illustration, suppose that 10 emails are generated for Alfred every week and that the random number generator produces $P_{\text{drop}} = 0.8$ for Alfred, then this would mean that Alfred will receive 2 messages per week on average from week 1 to week 4. If the random number generator produces $P_{\text{drop}} = 1.0$ for Alfred at the end of week 4, then Alfred will receive 0 messages per week from week 5 to week 8. We do not refresh the random numbers for all members at the same epochs (beginning of week 1, week 5) but rather spread the refresh epochs as described in Algorithm 1. This provides us with the flexibility of choosing any time frame (say one week) for training data collection with adequate coverage of feature space.

Algorithm 1: Choosing $P_{\text{drop}}$ for each member

1. if memberId % 4 == weekOfYear % 4 then
   2. choose a new $P_{\text{drop}}$ for memberId
4. else keep using the previous $P_{\text{drop}}$ for memberId

We deploy this model for 1% of our members.

4.3.2 Model and Features

Once we have collected appropriate training data, we train two logistic regression models with $\ell_2$ regularization, one for each of the two utilities, viz., downstream session and complaint. Let $y$ denote the response to an email $e$ described by feature vector $x_e$, then the corresponding Bernoulli random variable $Y$ is modeled as

$$E[Y] = \frac{1}{1 + \exp(-\theta^\top x_e)}$$

where $\theta$ is the parameter vector which we want to learn. Once $\theta$ has been estimated from the training data, for any new example $x_e^{\text{new}}$ we simply use the mean of the Bernoulli distribution as the predicted response, i.e.,

$$\text{Prediction} = \Pr\{Y = 1\} = \frac{1}{1 + \exp(-\theta^\top x_e^{\text{new}})}$$

We include 4 broad categories of features in the feature vector $x_e$, viz.,

1. Targeted member’s profile features such as member’s locale, age, etc.
2. Targeted member’s in-app activity features such as member’s last visit time, number of sessions over the last week, etc.
3. Targeted member’s past experience with emails such as number of emails received in the past week, number of emails clicked, etc.
4. Email message features such as type of message, length, etc.

In addition, we also include interactions between the above features.

We train these logistic regression models on Spark [12] using in-house optimization libraries which implement a distributed version of TRON [13], a trust region Newton method. The distributed optimizer enables model training with a large number of features and training examples on the Spark framework which allows rapid iterations.

4.3.3 Model Validation

We compute 2 metrics to measure the performance of our response prediction models:

1. Area under the receiver operating characteristic curve (AUC): This is to verify whether the model is directionally correct, i.e., larger predictions for positive examples.

2. Observed to expected ratio (O/E ratio): This is to verify the scale of the model. This metric is computed as the number of positive test examples divided by the sum of predicted probabilities for all test examples. An O/E ratio close to 1 is desired. It is important to verify the scale of the model because the sum of the predicted utilities is used to approximate the actual number of sessions and complaints in the constraints of the MOO formulation.

The AUC and O/E ratio metrics for our best utility prediction models are tabulated in Table 2.

<table>
<thead>
<tr>
<th>Utility</th>
<th>AUC</th>
<th>O/E ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>0.86</td>
<td>0.99</td>
</tr>
<tr>
<td>Complaint</td>
<td>0.81</td>
<td>1.05</td>
</tr>
</tbody>
</table>

Table 2: Validation metrics for prediction models

4.4 Offline Evaluation

As described in Section 4.2.2 we use the set of emails generated in the past week to obtain the MOO coefficients, and then use those for making send/drop decisions for emails which will be generated in the upcoming week. Since there is an element of uncertainty here, we use a simple replay methodology to evaluate the effectiveness of our volume optimization models (response prediction + MOO coefficients) before deploying them to the online serving system.

We collect one week of data from our send-all bucket, starting from a point in time after the period over which the training data was collected. The emails $e$ thus collected are annotated with the observed user responses, namely, observed complaints ($oc = 0/1$) and observed downstream sessions ($os = 0/1$). Now Algorithm 2 is used to compute the replay results. Note that we are only maintaining counters for the global utilities in this algorithm, but its extension to include local level utilities should be obvious.

The sends, sessions and complaints ratios thus produced should closely match the expected ratios as per (1)-(6). For example, at the global level the following should hold true:
Algorithm 2: Replay algorithm for global utilities

Data: \( d = (x_e, oc, os) \)

1. complaints = sessions = sends = 0
2. maxComplaints = maxSessions = maxSends = 0
3. foreach \( d \) do
   4. maxComplaints = maxComplaints + oc
   5. maxSessions = maxSessions + os
   6. maxSends = maxSends + 1
   7. if type\( (e) = t \) then
      8. \[ \text{score}(e) = \mu^t P_{\text{treat}}(x_e) - \nu^t P_{\text{comp}}(x_e) - 1 \]
   9. if score\( (e) > 0 \) then
      10. complaints = complaints + oc
      11. sessions = sessions + os
      12. sends = sends + 1
13. sendsRatio = sends/maxSends
14. sessionsRatio = sessions/maxSessions
15. complaintsRatio = complaints/maxComplaints

\[
\begin{align*}
\left| \text{sendsRatio} - \frac{\sum_e z_e}{\sum_e 1} \right| & \rightarrow 0 \\
\text{sessionsRatio} & \geq \alpha_{\text{global}} \\
\text{complaintsRatio} & \leq \beta_{\text{global}}
\end{align*}
\]

Although this replacer is a simple and efficient tool for obtaining an estimate of a model’s performance in an online A/B test, it is not always accurate due to a shortcoming of this replacer. This replacer need not capture all the dynamics of the population on which the model is going to be deployed. For example, say we are going to deploy our new model to a population which is being served by another model which drops emails with a fixed probability of \( P_{\text{drop}}=0.9 \). Then the distribution of \( P_{\text{treat}}(x_e) \) and \( P_{\text{comp}}(x_e) \) for this population is likely to be different from the send-all population used in the replacer.

5. SYSTEM ARCHITECTURE

In this section, we outline the important components of our system, as illustrated through a block diagram in Figure 9. The system consists of an online serving system and an offline training system.

Depending on a member’s message subscription settings, the message generator will produce a message for the member. For example, if the member subscribes to network updates, then a message is generated once a week which contains important updates pertaining to the member’s connections, such as job changes, work anniversaries, profile updates, etc. This message is passed down to the utility prediction engine along with the ID of the target user. The utility prediction engine extracts the message features from the message to create a partial feature vector and appends member features of the targeted user from a tracking data store. This feature vector is used in the utility prediction models to predict the expected utilities as per (8). These predicted utilities are passed to the Volume Optimization (VO) decision engine which makes the final send/drop decision based on the MOO coefficients as per (7). The member’s interactions with the received emails, along with member profile and activity data are recorded in the tracking data store.

Hourly snapshots of this database are loaded into Hadoop. The snapshot data is used in the utility model trainer for training response prediction models on Spark. These models are fed into the utility prediction engine as well as the MOO solver. The MOO solver employs these models on the training data for predicting expected utilities, which are used in the optimizer to produce optimal MOO coefficients. Note that the beauty of the design lies in the ability to make a send/drop decision for each individual email independently, based on the learned response prediction models and a few MOO coefficients. Further, this simple but effective design is what enables us to scale the online decision making to handle millions of message requests every week.

6. EXPERIMENTS & RESULTS

6.1 Experimental Setup

We set up an online A/B test experiment to evaluate the performance of our volume optimization approach. Members in the treatment bucket received emails based on the output of the volume optimization decision engine. The MOO coefficients used in the decision engine were dual solutions to a MOO problem with constraints listed under the ‘Constraint’ column of Table 3. The experiment covered several email types which constitute about 70% of the total volume of emails sent by LinkedIn. We collected several metrics of interest from this bucket and compared them against the control send-all bucket in which members received all emails generated for them.

6.2 Results & Analysis

Table 3 summarizes the offline replay and A/B test results. The ‘Contribution’ column shows the contribution of each email type towards the total sends, complaints and downstream sessions in the send-all bucket. The ‘Constraint’ column lists the complaint tolerances and session targets at global and local levels. In this experiment, at the global level we are willing to tolerate no more than 60% of the maximum possible complaints (when all emails are sent) and target at least 98.5% of the maximum achievable sessions. It is quite obvious from Figure 8 that it is impossible to achieve this kind of a trade-off through the naive threshold-
We draw the following observations from the table:

1. The MOO coefficients $\mu_1$ and $\mu_2$ corresponding to the session constraints for email type 1 and 2 respectively are equal to 0. This means that the corresponding local level constraints on session targets were inactive at the point of optimality. This in turn implies that we could achieve more local sessions for these email types than the target. For example, we have specified a session target of 95% under the Constraint column for email type 1 but have achieved 101.06% sessions for this email type in our A/B test. Similarly, we achieve more sessions (93.92%) in the A/B test than our target of 80% for this email type.

2. The MOO coefficient $\nu_2$ corresponding to the constraint violation for email type 2 is orders of magnitude larger than the others. This is because we have specified a very strict constraint on the complaints tolerance (50%) for email type 2. In general, the stricter a constraint the higher are the chances of its corresponding dual solution being larger. This is because a large MOO coefficient will ensure that the corresponding utility prediction plays an important role in the send/drop decision making as per (7). For example, the large $\nu_2$ will ensure that an email $e$ of type 2 is sent only if its predicted complaint probability $P_{comp}(xe)$ is very small.

3. All constraints are being satisfied in the A/B test results, except for a minor violation of the global sessions constraint. This violation is not surprising since we solve the MOO problem for emails generated in the previous week and use its solution to make send/drop decisions for emails generated in the following week.

4. The offline replay is making reasonable predictions about the A/B test results but it is not always accurate.

Weekly trends for the global metrics that we directly optimize for, viz., sends, sessions and complaints are shown in Figure 10. The red line in each plot corresponds to the A/B test result while the blue line corresponds to the constraint. Table 3 above was a detailed presentation of results from week 8.

We also saw an indirect impact of volume optimization on certain metrics that we did not explicitly optimize for. The impact on one such metric, namely, the number of unique users who visited a page is tabulated for some of the product pages in Table 4. A '-' entry in the table indicates that the impact was not statistically significant. We do not observe any statistically significant loss in the total number of unique users visiting the mobile or web applications. This was expected because of our strict constraint on global session targets. However, we do observe a reshaping of user visits across various products. This is a nice byproduct of volume optimization which eliminates less relevant and poor quality emails. If poor quality emails are dropped, the total email volume received by a member is reduced allowing the member to devote more time and attention to the good quality emails. So although certain products lose out on unique users, the more relevant products gain. This is consistent with our observations in Section 3.2 where we had seen that user attention gets divided between important and less important emails. Here we would like to point out once again that this kind of interaction between emails is ignored in our MOO formulation where we assume that the send/drop decision for an email does not affect the probability of a session or complaint from any other email. Relaxing this independence assumption is the primary target of our future research. On a lighter note, the large drop in unique users visiting the HelpCenter pages is rewarding and also amusing, for reasons which should be obvious to the reader!

7. CONCLUSION

To the best of our knowledge, this is the first work on email volume optimization for a large online social networking service. We have introduced the challenging nature of the problem through a comprehensive cost-benefit analysis of email communication. It has been found that on the one hand emails deliver significant value to the members thereby driving in-app engagement, but on the other hand excessive email volume results in increased negative responses such as
a member unsubscribing from an email type or reporting an email as spam to her email service provider. Excessive email has also been found to result in the user’s attention drifting away from important emails. We have illustrated why naive approaches to volume optimization result in sub-optimal solutions, and presented our formulation of the problem as a multi-objective optimization problem, with the objective of minimizing the number of emails sent in a way so as to maximize the positive outcome and minimize the negative outcome of sending emails. We have discussed the challenges faced in solving the multi-objective optimization problem and our workaround. We have described how we evaluate our volume optimization models offline before deploying them in production. The major components of our volume optimization system, comprising of an online serving system and an offline training system have been outlined. Finally, we have demonstrated the effectiveness of our approach through online A/B test experiments.

8. ACKNOWLEDGMENTS

This work would not have been possible without the invaluable insights from our team at LinkedIn comprising of Ankan Saha, Carl Cummings, Deepak Agarwal, Kinjal Basu, Liang Zhang, Rishi Jobanputra, Shaunak Chatterjee and many others.

9. REFERENCES


