Alma Mater Studiorum · Università di Bologna

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS
in
Machine Learning

TIME SENT OPTIMIZATION: THE DIENNEA CASE

Supervisor:
Prof. Claudio Sartori

Candidate:
Petru Potrimba

Co. Supervisor:
Dr. Andrea Pagliarani

Academic Year 2020-2021
Session II

Alla mia famiglia, che mi ha
accompagnato in questo lungo cammino
e non mi ha mai fatto mancare niente.

Contents

Abstract

Every marketer, at one time or another, has wondered when the perfect time to send emails is. Are recipients more likely to open messages in the morning or late at night? What about on Tuesdays at the lunchtime hour?

This problem is called Time Sent Optimization (TSO). Why is it so important? Because it allows marketers to send emails at the optimal time for each contact. Also, it helps marketers engage more effectively with contacts, gaining contacts' attention when they are historically most attentive to their emails. Moreover, suppose that you want to start an advertising campaign. The more emails will be opened, the more the company will gain in terms of money and it will be easier for them to advertise some product. Not only, we do not want just to increase the open ratio, we also make sure that the user reads it. This translates is clicking the email, like scrolling it or clicking on links present in it.

Most of the medium and big companies already use Time Sent Optimization for the reasons explained before. However, they do not publish how they did that because of business reasons. If someone would publish a way of doing this, then everyone will copy it and then the purpose of doing this will be useless because it is likely that at a certain time, each user, will receive a lot of emails from several companies and this would lead ruin the benefit of TSO.

In this thesis, we take into account the case of a company called Diennea which gave us their data to improve their open and click ratio on the communications they send to their users. At the end, through online A/B test results, we show the effectiveness of our proposed approach and how to further improve it.

Business understanding

The most important part of a project is to understand, from a business perspective, what a customer really wants to accomplish. This is essential to analyze because this business understanding will let us to understand which will be our profit and the impact of our product. The objective of this thesis project is to understand, for each client, which is the right moment to send him an email such that he will open and read it. What would be the advantages for a company that sends emails from a business perspective? The advantages are related to marketing and advertising products. The more users open and read emails, the better the marketing campaign for that company will be, and this translates almost automatically in gaining more money.

So, we need to improve the open rate of the sent emails. Of course, the more I can improve this open rate, the better. But what happens if this open rate is already high? Maybe it will be useless to spend time and money to build such tool, just because we will have a negligible impact. In our case, the company that asked us for such tool, had a pretty low open rate and click rate. So there is margin for improvement.

The resources that we start from are the data the company gave us. Before starting the project, we need to understand whether these data are good or bad. This is essential to do in order to compute the risks of failure. We can model a perfect algorithm, but if the data are not good, there is a probability that we will fail and therefore we would have spent a lot of time and money just for nothing. Fortunately, after some data analysis, we concluded that the data the company gave us where well structured and therefore we assumed that there were margin of improvement.

After analysed that the project is feasible, we structured the following pipeline:

- data analysis: in this phase we need to perform several statistics on the data in order to understand patterns and insight useful for the next phases.
- feature engineering: once understood our data, we need to create the

dataset that will be fed to our model. In particular, we will adopt a classic machine learning pipeline where we manually create the feature of our interest based on what we discovered in the data analysis phase.

- Time Sent Optimization algorithm: in this phase we will create several models that given in input a contact, they will return which is the best time to send to the user the email such that he will open it.
- testing: the testing phase is composed of two parts, the model evaluation through ad hoc metric and an A/B test. These two parts are crucial because they will tell us whether we are actually improved the open rate and click rate or not.

1 Exploratory data analysis

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

In particular, in this chapter, we will see how the dataset we have is structured and we will perform several statistics that will help us in the next phases.

1.1 Dataset

The raw dataset is composed is this way: each row of the dataset represents an *event*. The event can be of several types:

- *sent*: a row in the dataset that has the type sent means that the company sent an email to a specific user.
- *open*: a row in the dataset that has the type open means that a user has opened that specific email.
- *unsubscribed*: a row in the dataset that has the type unsubscribed means that a user unsubscribed from the email campaign.
- *complaint*: a row in the dataset that has the type complaint means that a user sent an email to the company writing them about the issues the user experienced.
- *bounced*: a row in the dataset that has the type bounced means that an email that has been sent to the user, has not arrived for some technical issues.

Each row is accompanied by other columns. Here we are going to outline the most important ones:

- *Event date*: indicates when the event has occurred.

- *HashMessaggio*: indicates a user unique identifier .
- *HashContatto*: indicates a email unique identifier.
- *Campaign name*: indicates the name for that email campaign.
- *Communication name*: indicates the communication name for a specific email.
- *Communication subject*: indicates the email subject.

The dataset is composed by 120k events and 2942 distinct users. Each user has received roughly 550 emails.

1.2 Statistics

In this section, we are going to collect some statistics on our data in order to get to better know them. This section is important because it will define which will be the features of our dataset. Moreover, we need to make statistics keeping in mind our goal: improve the open rate and click rate.

1.2.1 Current open rate and click rate

First of all, let us check which is the current open rate and click rate of our data. This is the main step to do before starting the project because if this rate is already high enough, maybe there is no need to apply any Time Sent Optimization technique on it. Otherwise, if it is low, we can estimate just by looking at the number how much we can improve it (namely, if it is very low, we can assume that the margin of improvement could be high).

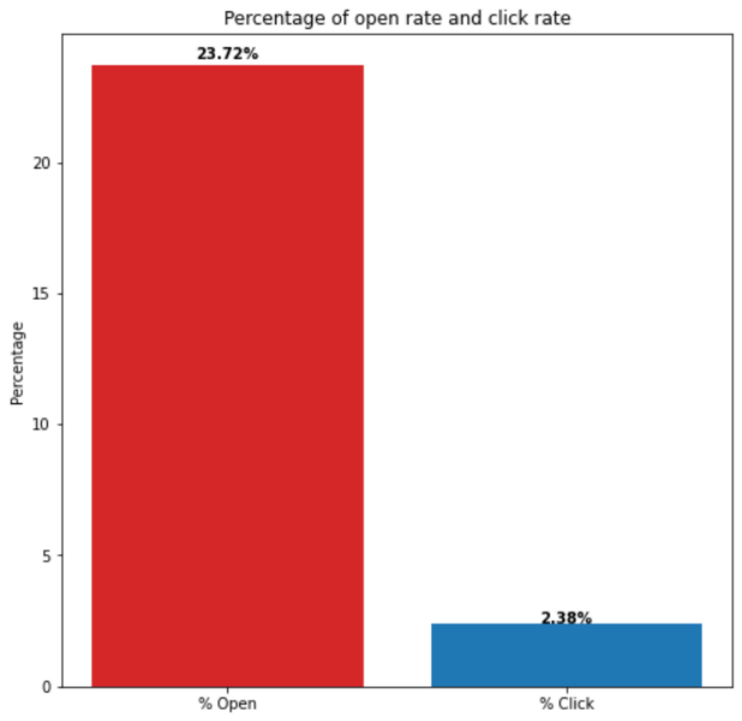
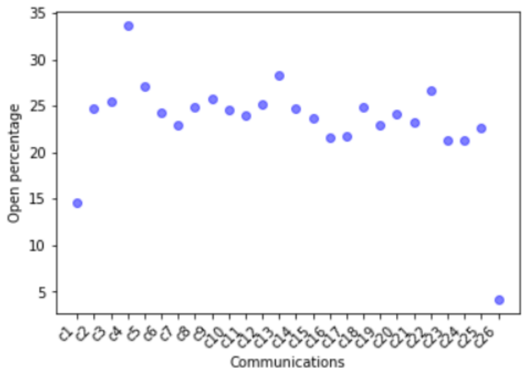


Figure 1: Percentage of open rate and click rate.

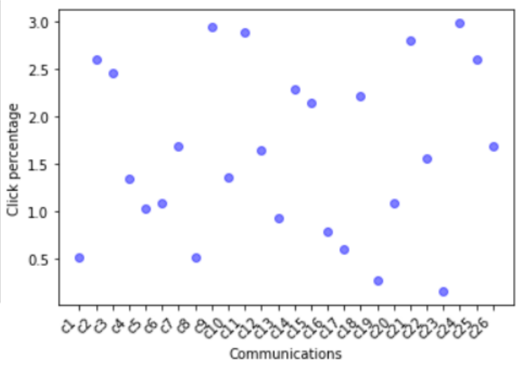
From the bar chart, it can be noticed that we have two percentages that are pretty low, in particular the percentage of click rate. So we can assume that we can build a model that can significantly improve both the open and click rate.

1.2.2 Communications open and click ratio

We know that each email is related to a specific communication. Here we want to check if some communication has a higher or lower open and click rate. If so, we can conclude that the type of communication would be a feature that can influence significantly the performance of our feature predictions. The following chart represents the percentage of opened emails among all the sent emails for each distinct communication.



(a) Open rate per communication.



(b) Click rate per communication.

From the left-most chart, it can be seen that most of the open ratio per communication is the same on average but the first and last communication which have respectively a percentage of $\sim 15\%$ and $\sim 5\%$. Also, it can be seen that the fourth communication has a peak of $\sim 35\%$. But, overall, the general trend of the data is similar.
On the other hand, the click rate per communication is very different. There is not any trend to notice. They are quite ”random”.
Since there are so many different fluctuations, specifically in the click rate per communication, we can conclude that the communication is directly depend on the fact that the user opens or clicks the email. So, we are going to consider the types of communications in the next phase when we will do feature engineering.

1.2.3 Average temporal distance between Sent and Open - Open and Click

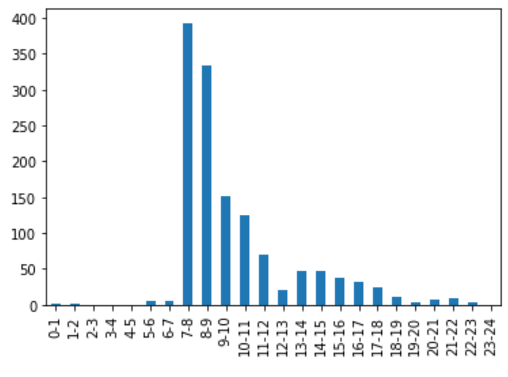
We want our users to open and click the emails as soon as possible. If we have a high open and click rate but these emails are opened, say, one month later, maybe this is not so useful for us. Maybe the communication is outdated.
So we need to check the average temporal distance between the sent and open email and between open and click. The values are computed following the format days hours:min:sec, and they are:

- *Distance between Sent and Open*: 1 days 00:13:54.
- *Distance between Open and Click*: 0 days 14:40:25.

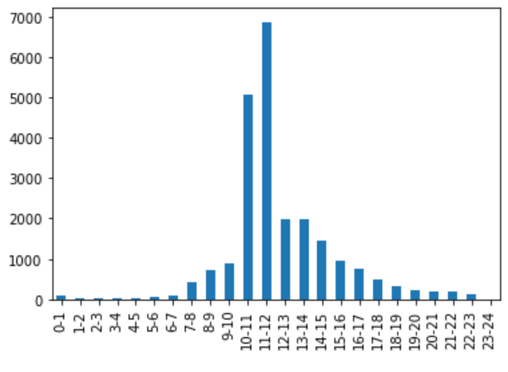
The elapsed time between both is very small. This is a good news since we are already satisfied with these values so we do not have to optimize them.

1.2.4 Distribution of time slots of open messages

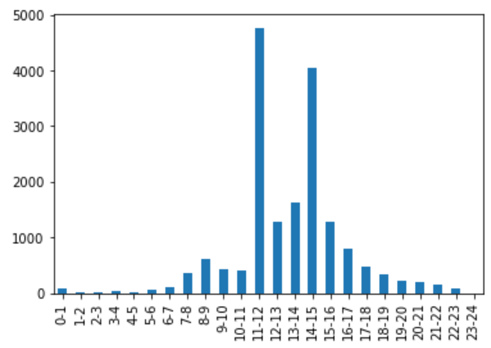
In this section we want to check *how many emails are opened for each time slot among all the emails sent at a specific hour.* This should give us a hint about which are the best time to send an email.
Subsequently, you can find the distributions of the time slots of the opened emails when the emails are sent in a specific time slot.

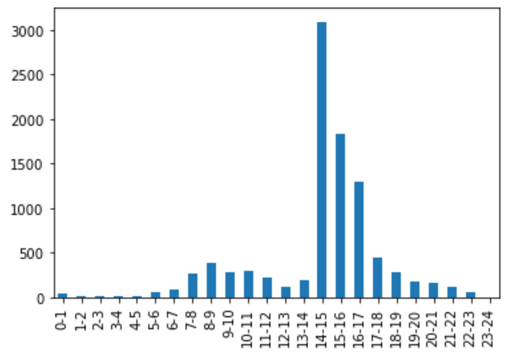


(c) Time slots of opened emails of emails sent from 5 to 8.



(d) Time slots of opened emails of emails sent from 8 to 11.


(e) Time slots of opened emails of emails sent from 11 to 14.


(f) Time slots of opened emails of emails sent from 14 to 17.

Looking at the charts, a significant pattern emerges. The pattern is that, if an email is sent in a specific time slot (like from 5 to 8), the majority of opened emails are right after the sent.
Furthermore, we can see that from 23pm to 7am, very few emails are opened. So we can deduct that this time slot is not a good one to send the emails. We are going to consider these patters too in the feature engineering phase.

1.2.5 Users' lifetime

Another important aspect to observe is how our users behave. There could be users that ”are interested” in every communication of the company and therefore they will open the emails every time, regardless the time at which the emails have been sent.
On the other hand, there could be users that open the communications only if they are sent at a specific time or users that do never open the communications, regardless the time at which the emails have been sent.
So it could make sense to look at the *users' lifetime* which in this context is defined as ”the elapsed time between the first email opened by a user and the last one”.
In the histogram below, we are going to plot the users' lifetime taking into account that each bin represents 10 days.

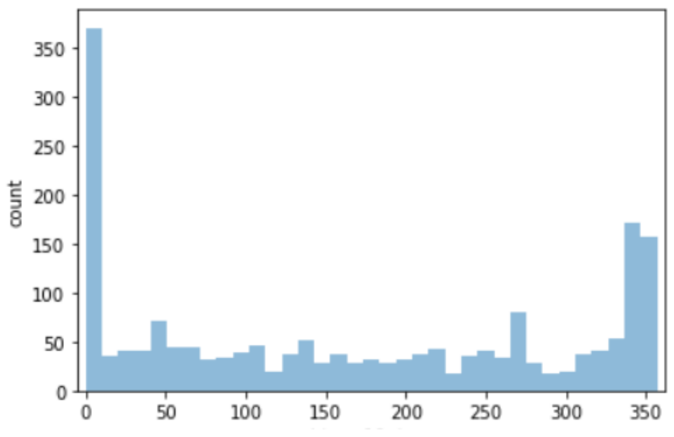


Figure 2: Users' lifetime.

From the above chart we can notice mainly two patters:

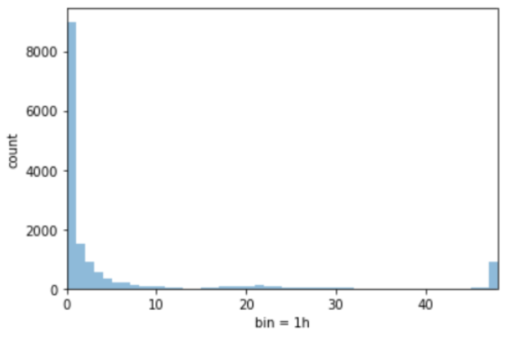
- the first one is that roughly ~ 360 users have a lifetime between 0 and 10 days. This means that a lot of users have opened the last email a lot of time ago and they did not open any email recently because they are not interested in them.
- the second one is that roughly ~ 300 users still open the emails since they subscribed, so this means that this slice of users is interested in the email's content.

and in between we have several users with different lifetime. Since there is a significant behaviour between users, this is something we will need to take into account. In particular, this information will be useful in the evaluation part which we will see in the next chapters.

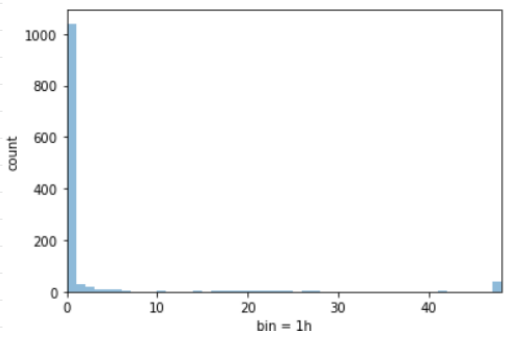
1.2.6 How many communications are opened and clicked in the first hours

Now we are going to analyze how good is the sent time that the company currently uses for their users. How to do that?
One thing we can do is to check the open ratio and click ratio that we already did before. Also, we can plot a chart that says *how many emails, from those*

sent, are opened in the first hour, how many are opened in the second hour until 48 hours.
We do this both for the opened emails and clicked ones using a bin of 1 hour.



(a) How many email are opened per hour.



(b) How many email are clicked per hour.

We can observe that the vast majority of the emails are both opened and clicked in the first hours and some of them are opened over 48 hours.
Also, we can notice that the histograms follows a decreasing exponential function.
This trend will be useful in the feature engineering phase where we will create feature based on that trend.

1.2.7 Open rate based on the day of the week

Here we want to check how good is the day of the week that the company has chosen to send the emails.
The day of the week could be crucial and it could influence a lot the fact that the user opens or not the email.
One way to check that is to draw a plot that indicates *among the sent emails on the day of the week x, how many of them in percentage are opened in the future?*. We do this in the subsequent chart.

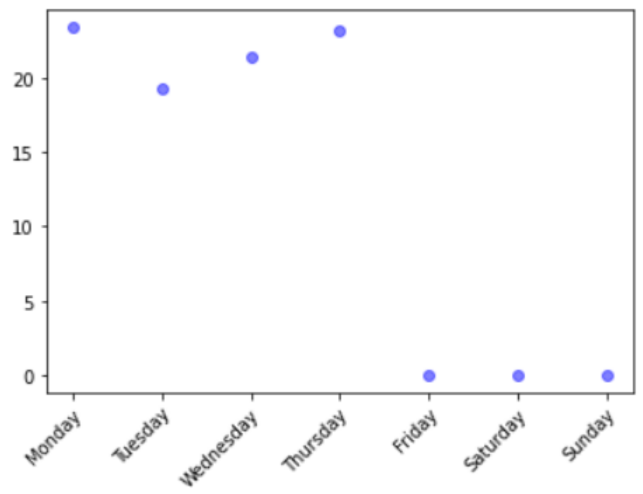
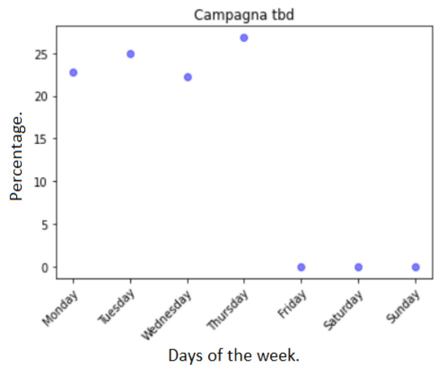


Figure 3: How many emails in percentage are opened in the future based on the day of the week.

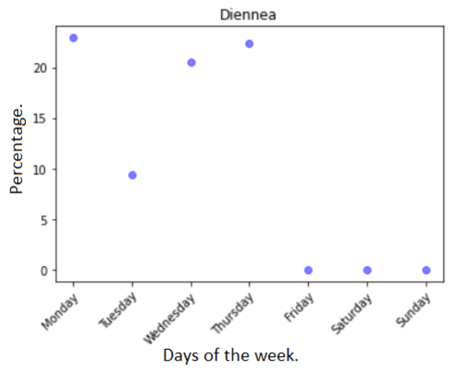
At first glance, we can directly observe that the email sent on Friday, Saturday and Sunday will never be opened in the future.
Actually, this is not true. Those percentage are zero simply because the company never sends emails on these days.
Another thing to notice instead is that if we send the emails on Tuesday, there are less chances that the emails will be opened in the future compared to the emails that are sent on Monday, Wednesday and Thursday which have roughly the same percentage.

1.2.7.1 Open rate based on the day of the week for each campaign

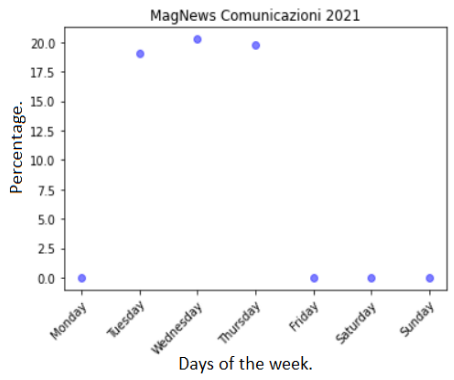
Each email belongs to a specific campaign. Here we want to understand whether the emails sent within a specific campaign can affect the open rate. To verify whether this is true, we are going to make the same plots as before but subdividing them by each campaign.
So, among the sent emails on the day of the week x, how many of them in percentage are opened in the future *for each campaign*?
We have three distinct campaigns, so we are going to make three plots.



(a) How many emails in percentage are opened in the future based on the day of the week and the campaign *tbd*.



(b) How many emails in percentage are opened in the future based on the day of the week and the campaign *Diennea*.



(c) How many emails in percentage are opened in the future based on the day of the week and the campaign *MagNews Comunicazioni 2021*.

As already discussed before, we have 0 % in correspondence of the days of the week Friday, Saturday and Sunday because the company did not send any emails on those days. Also, the company decided to not send emails on Monday for the campaign ”MagNews Comunicazioni 2021”, this is why we have a 0 % there.
No significant differences emerge between the three campaigns, a part for the campaign ”Diennea” on the day Tuesday which has an open rate halved with

respect to the other days and campaigns.
We have gone deeper in order to understand why the day Tuesday on the campaign Diennea and we discovered that on that day and that campaign, all the emails are sent at the exact hour for the all the users. While, on Monday, Wednesday and Thursday, the variance of the hours at which the emails has been sent is high, this means that the emails one these day are not sent at the same hour, unlike the emails sent on Tuesday.
In particular, on Tuesday, all the emails are sent at the hour 10:31. Since the open rate is very low, we can assume that sending the emails to all the users at that hour is not a good choice.

2 Feature engineering

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.
Machine learning algorithms learn a solution to a problem from sample data. In this context, feature engineering asks: what is the best representation of the sample data to learn a solution to your problem?
It is deep. Doing well in machine learning, even in artificial intelligence in general comes back to representation problems. It is hard stuff, perhaps unknowable (or at best intractable) to know the best representation to use, a priori.

The features in your data will directly influence the predictive models you use and the results you can achieve. You can say that: the better the features that you prepare and choose, the better the results you will achieve. It is true, but it also misleading. The results you achieve are a factor of the model you choose, the data you have available and the features you prepared. Even your framing of the problem and objective measures you are using to estimate accuracy play a part. Your results are dependent on many interdependent properties. You need great features that describe the structures inherent in your data.

So, in this chapter, we are going to discuss which are the features that we have chosen among the possible set of features that we experimented along with the choice of the label.

2.1 Contact open and click rate

Probably, the most important feature, is the open and click rate for each contact. Since we want to predict which is the best time to send the communication, we need to have the information of *when the users mostly open and click the emails.*
In particular, this feature is not "just a column in the dataset" but multiple columns that says, for each hour, the open and click rate for every user. So,

since the hours are 24 and we need to encode both the open rate and click rate, we end up with 48 columns that encode this features (24 columns for the open rate + 24 columns for the click rate).

2.2 Communication open and click rate

In the exploratory data analysis conducted in the first chapter, we have discovered that the type of communication influences the opening and click of the email. So we want to add this feature to our dataset.
As in the previous feature, here we do not have just a single column, but we have a column for each time slot, so 24 columns.
Each time slot represents *which is the open and click rate for that communication in that specific time slot.* As before, since we are encoding both the open rate and click rate, we will end up in having 48 columns for this feature.

2.3 Fitness Send-Open

This feature, differently from the previous ones, is just a single number and with this feature we want to encode *how good or bad was the real sent time that the company used to send a specific email to a specific user.* In particular, the idea is that: if the difference between the time at which the email has been sent and the time at which the user opened the email is small, then the sent time adopted is good. Otherwise, if it is large, then the sent time adopted is not a good one. So, the model needs to learn to optimize upon this feature if the time adopted is not a good one.

How to represent this into numbers? Well, we just compute the difference in minutes that elapses between the send time and open time and we pass this number to a decreasing exponential function that squashes the obtained number into a number into the range 0 and 1. The used decreasing exponential function is represented below.

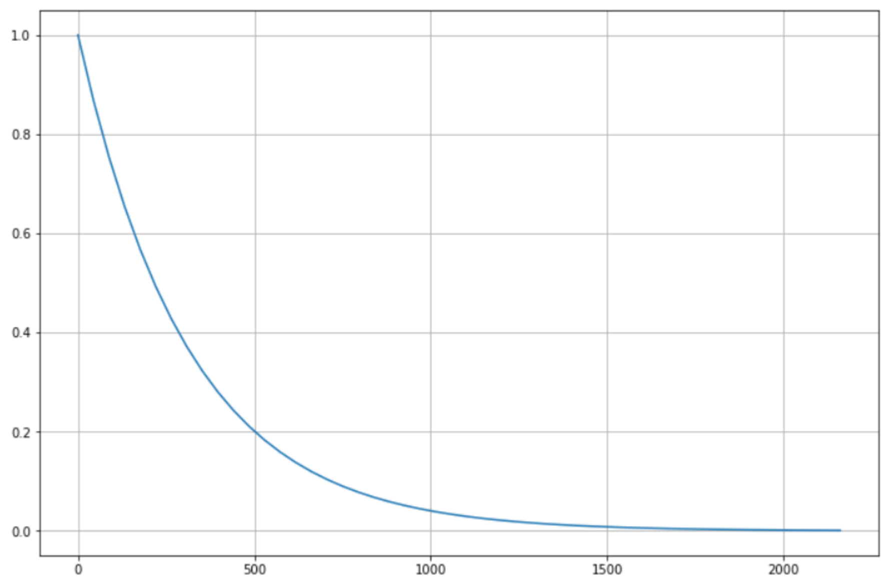


Figure 4: Fitness Send-Open.

We have shaped this function such that in correspondence of 0 on the x axis, it gives us the maximum, namely 1. This means that, if the email has been opened within a minute after its send, we have a 0 as elapsed time, so we want the maximum fitness for that email. Otherwise, if the email has been opened one minute later, then the exponential decreasing function gives us the relative value on the y axis. We have encoded also that, if the email is not opened within 36 hours, then we assign directly 0 as fitness. Also, this function has another hyperparameter which is the slope of the decrease. We have tested a range of slopes, even a slope which is approximated to a straight line, and we come up with the one shown in the above figure because, at the end, gave us the best results.

Everything seems fine, however there is a hidden problem with this feature. The problem is that, when we run a model on this dataset, this feature is overwhelmed by the other 96 features and it gives us a very few contribution. To solve that problem, we just need to give more weight to this feature and tune that parameter accordingly. Unfortunately, not all the already existing models allow to do that, but luckily enough the models that allow that worked pretty fine. If someone wants to use a specific model that allows to specify a weight for the features, he/she needs to build the model on his/her own.

2.4 Fitness Open-Click

This feature is exactly the same as the previous one, with the difference that the minutes we use to compute it are the minutes that elapse between the open time and click time of the email. As before, if this number is low, we will have a high number returned from the exponential decreasing function. If it is high, the function will return a low value. The chosen hyperparameters are the same as before but the slope decrease. In this case, we want a higher slope because the clicks are far rare compared to the openings, so whenever we have one, we want to give them a higher contribution. Otherwise, this feature would end up in having mostly zeros or very low numbers. The exponential decreasing function we used for this feature is depicted in the following chart. Note the difference of this slope with respect to the slope of the previous function.

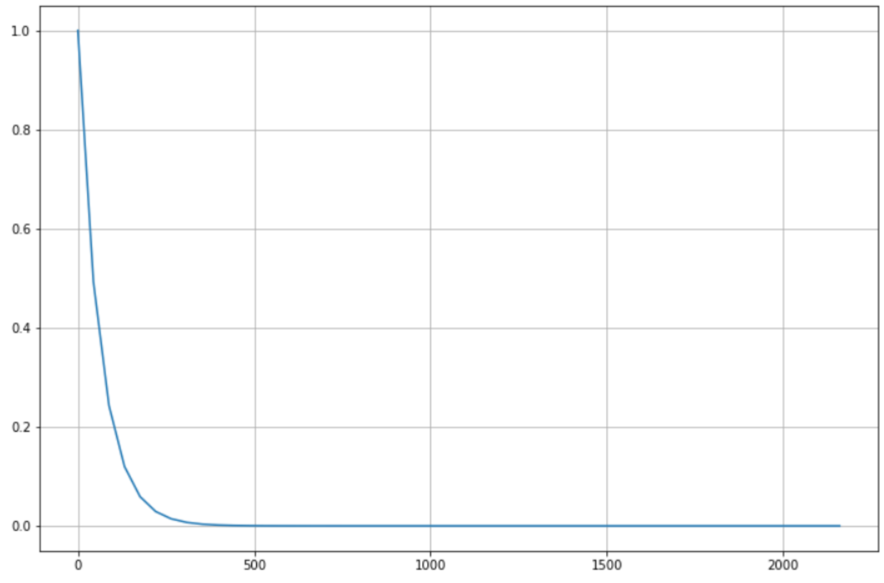


Figure 5: Fitness Open-Click.

Moreover, the hyperparameter that controls how much this feature contributes with respect the other features is tuned accordingly.

2.5 Building the dataset

Now that we have engineered the features, we can proceed to build the dataset. This dataset that we are going to build is not structured as the dataset we had at the beginning where each row represented an event. Here, each row contains for each contact and each message, what are the values of the all the previous features computed before.

We did several experiments and we come up actually with two different types of dataset where each one is applied to a specific algorithm we will describe in the next chapters.

2.5.1 Dataset 1 and label choice

The first dataset we used is just a stack of the engineered features. What really changes is the choice of the label to attach (supposing we will use a supervised learning approach). In particular, with this dataset, we can have three types of labels:

- the first thing that comes in mind is to use as label the *opening time* of the email. Actually this is a good idea, but there is a problem, a big one actually. The problem is that in the dataset, each row represent the email sent to a specific user and the info about the opening and click about that email. But we have several emails that are sent to the same user, so we will have several opening times and therefore we cannot assign a specific label to every user. Remember that our goal is to learn, for each user, which is the best time to send him the communications. We do not want our model to output several times for a specific user. So this label cannot be used.

- the second label that can be used is the time slot of the sent time. this means that if the email has been sent at 11:30, then we assign as label 11. If the email has been sent at 13:01, we assign as label 13. Also here, the hour at which the emails are sent to a specific user are different, but here there are less problems because we take into account the values of the fitness Send-Open and Open-Click that suggest us how good or bad the attached label is. However, this is not what we used because the performance where not so good.

- the third label we tried, that we also used, consists in attaching the exact sent hour as label. So if the email has been sent at 11:30, we assign as label exactly 11:30. But how to attach a specific hour a minute? We computed the time elapsed in minutes from midnight and squashed the resulted number into the range 0 and 1. This solution turns out to work much better with respect to the second label choice.

2.5.2 Dataset 2 and label choice

Here we talk about the second dataset we created which is actually that one that gave the best results. We have introduced previously also the dataset 1 because it makes sense to give it a try since depending on the type of data one could have, it could work better compared to this one. This dataset is composed by the only features *contact open and click rate* and *communication open and click rate*. Here we removed both fitness. What really changes is the label. Instead of having one number in the label, we have all the possible time slots from 00:00 to 00:00 of the next day, which are 24 time slots. Not only, we take each time slot and divide it further into a properly tuned hyperparameter, which is 4 in our case. So we will end up in having time slots where each one represents a quarter of an hour. What values each quarter of an hour has? To understand it, let us make an example: if the email is opened at 11:30, we start by computing how many minutes are elapsed between the first quarter which is 00:00. Supposing these minutes are x. We take this x and pass it to a decreasing exponential function (as before with the fitnesses) that gives us a number. This number is that goes into the quarter 00:00. The we proceed doing the same for the next quarter, which is 00:15. And so on and so forth. In this way, we are giving as label a histogram which indicates which *is the users' fitness for all the possible hours*. We are kind of taking the previous fitnesses and injecting them into the label and therefore the goal now translates into learn this fitness distribution. Let us plot how one of the label look like:

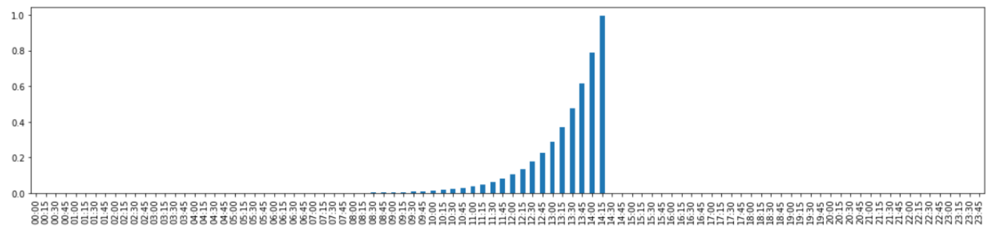


Figure 6: Label of dataset 2.

As it can be seen, the function that is drawn is what we expect, namely and exponential because the function we used to encode the label's numbers is a decreasing exponential. The more we get close to the open quarter hour, the more the fitness is high. Note that after the open quarter hour, we have zero everywhere because we want to minimize only in one direction. This means that we want our send prediction to be before the real open, not after because if, for instance, our real open is at 12:00 and our prediction for sending the email is at 13:00, then our error is no by 1 hour, but is by 23 hours. Therefore, to avoid this, we just set our fitness to zero on all the quarters after the open quarter hour.

It could also make sense to plot the mean of all the labels we have in our dataset to get an insight about which are the quarters where the emails are mostly opened.

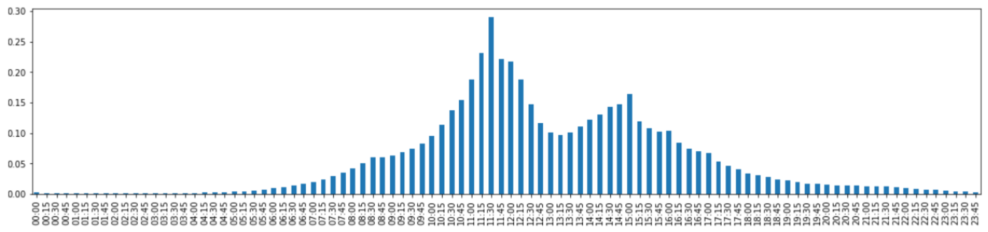


Figure 7: Mean of the labels of dataset 2.

From the above chart we can directly notice that there are two hours that are particularly preferred to open the emails. They are: 11:30 and 15:00. They could also be interpreted: 11:30 is a hour before lunch break, while 15:00 is a hour after lunch break. So we do expect that hour model will, on average, suggests hours that are close to those two. On the other hand we can see that from 00:00 to roughly 06:00 the users almost never open the emails.

3 Time Sent Optimization algorithm

In this section we are going to describe the algorithms that we implemented and tested. We start by showing the least performing algorithm till the most performing one.

3.1 Unsupervised learning

The first attempt to solve this problem is by using an unsupervised algorithm. But what is unsupervised learning? Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

Unsupervised learning models are utilized for three main tasks: clustering, association, and dimensionality reduction. Below we will define each learning method and highlight common algorithms and approaches to conduct them effectively.

3.1.1 Clustering

Clustering is a data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by structures or patterns in the information. Clustering algorithms can be categorized into a few types, specifically exclusive, overlapping, hierarchical, and probabilistic.

3.1.2 Exclusive and overlapping clustering

Exclusive clustering is a form of grouping that stipulates a data point can exist only in one cluster. This can also be referred to as “hard” clustering. The K-means clustering algorithm is an example of exclusive clustering.

3.1.2.1 K-means K-means [19] clustering is a common example of an exclusive clustering method where data points are assigned into K groups, where K represents the number of clusters based on the distance from each group's centroid. The data points closest to a given centroid will be clustered under the same category. A larger K value will be indicative of smaller groupings with more granularity whereas a smaller K value will have larger groupings and less granularity. K-means clustering is commonly used in market segmentation, document clustering, image segmentation, and image compression. Overlapping clusters differs from exclusive clustering in that it allows data points to belong to multiple clusters with separate degrees of membership. "Soft" or fuzzy K-means clustering is an example of overlapping clustering.

3.1.3 Hierarchical clustering

Hierarchical clustering [7], also known as hierarchical cluster analysis (HCA), is an unsupervised clustering algorithm that can be categorized in two ways; they can be agglomerative or divisive. Agglomerative clustering is considered a "bottoms-up approach." Its data points are isolated as separate groupings initially, and then they are merged together iteratively on the basis of similarity until one cluster has been achieved. Four different methods are commonly used to measure similarity:

- *ward's linkage*: this method states that the distance between two clusters is defined by the increase in the sum of squared after the clusters are merged.
- *average linkage*: this method is defined by the mean distance between two points in each cluster.
- *complete (or maximum) linkage*: this method is defined by the maximum distance between two points in each cluster.
- *single (or minimum) linkage*: this method is defined by the minimum distance between two points in each cluster.

Euclidean distance is the most common metric used to calculate these distances; however, other metrics, such as Manhattan distance, are also cited in clustering literature.

Divisive clustering can be defined as the opposite of agglomerative clustering; instead it takes a “top-down” approach. In this case, a single data cluster is divided based on the differences between data points. Divisive clustering is not commonly used, but it is still worth noting in the context of hierarchical clustering. These clustering processes are usually visualized using a dendrogram, a tree-like diagram that documents the merging or splitting of data points at each iteration.

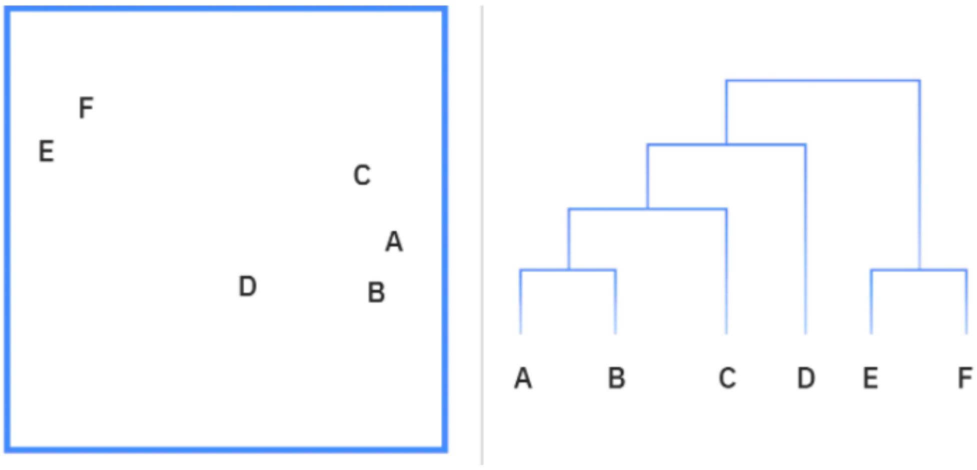


Figure 8: Hierarchical clustering.

3.1.4 Probabilistic clustering

A probabilistic model [4] is an unsupervised technique that helps us solve density estimation or “soft” clustering problems. In probabilistic clustering, data points are clustered based on the likelihood that they belong to a particular distribution. The Gaussian Mixture Model (GMM) is the one of the most commonly used probabilistic clustering methods.

Gaussian Mixture Models are classified as mixture models, which means that they are made up of an unspecified number of probability distribution functions. GMMs are primarily leveraged to determine which Gaussian, or normal, probability distribution a given data point belongs to. If the mean or variance are known, then we can determine which distribution a given data point belongs to. However, in GMMs, these variables are not known, so we assume that a latent, or hidden, variable exists to cluster data points appropriately. While it is not required to use the Expectation-Maximization (EM) algorithm, it is a commonly used to estimate the assignment probabilities for a given data point to a particular data cluster.

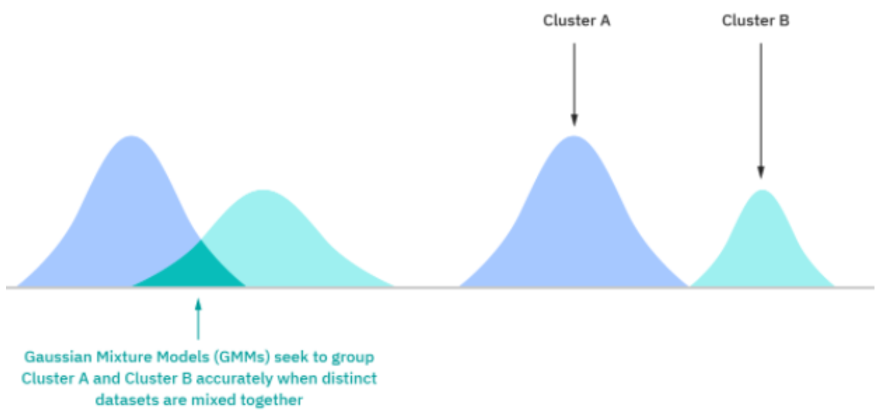

Cluster A
Cluster B
Gaussian Mixture Models (GMMs) seek to group Cluster A and Cluster B accurately when distinct datasets are mixed together

Figure 9: Gaussian mixture models (GMMs).

3.1.5 Association rules

An association rule [14] is a rule-based method for finding relationships between variables in a given dataset. These methods are frequently used for market basket analysis, allowing companies to better understand relationships between different products. Understanding consumption habits of customers enables businesses to develop better cross-selling strategies and recommendation engines. Examples of this can be seen in Amazon's "Customers who bought this item also bought" or Spotify's "Discover Weekly" playlist. While there are a few different algorithms used to generate association rules, such as Apriori, Eclat, and FP-Growth, the Apriori algorithm is most widely used.

3.1.6 Principal component analysis

Principal component analysis (PCA) [3] is a type of dimensionality reduction algorithm which is used to reduce redundancies and to compress datasets through feature extraction. This method uses a linear transformation to create a new data representation, yielding a set of "principal components." The first principal component is the direction which maximizes the variance of the dataset. While the second principal component also finds the maximum variance in the data, it is completely uncorrelated to the first principal component, yielding a direction that is perpendicular, or orthogonal, to the first component. This process repeats based on the number of dimensions, where a next principal component is the direction orthogonal to the prior components with the most variance.

3.1.7 TSO Algorithm 1

The first Time Sent Optimization algorithm attempted is based on unsupervised learning. The motivation for doing that is the following: users that have similar features (based on our previous engineered features) should receive the email at the same time. So, if we use some unsupervised learning algorithm and find some distinct clusters, we can set up an algorithm do to so. The general algorithm is depicted in the next figure.

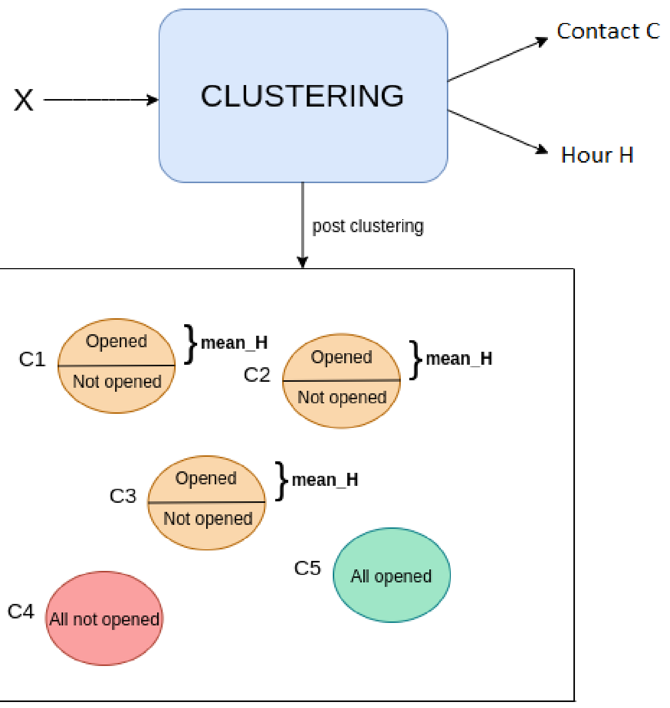


Figure 10: TSO algorithm 1.

It works in this way: we pass our dataset X to a clustering algorithm and for each contact C we want to output the optimal time to send the email H. Post clustering, we have three possible types of clusters the can be formed:

- the orange cluster: in this cluster are present some users that opened the majority of the emails and some that did not.
- the green cluster: in this cluster there are users that opened all the emails that have been sent to them.
- the red cluster: in this cluster there are users that did not open any email.

The idea of the algorithm is the following: for each type of cluster, we do an action:

- case orange cluster: here we compute the mean of the sent hour for all the users of this cluster that opened the vast majority of the emails and assign the mean hour (called *mean_H* in the above diagram), which will correspond to the sending time of the emails to all those users belonging to this cluster that did not opened the emails.

- case green cluster: here, since all the users are already opening all the emails, it means that the time to send the email for this type of users is already good enough, so we will continue to use the same identical sending time.

- the red cluster: the problem arrives here: which sending time should I use for these users? First, we need to analyse what are the sending time of these users and conclude that that sending time is not a good one. Then, one possible thing to do, is to compute the Euclidean distance between this cluster and all the other clusters and take the into account the minimum distance of the first cluster that is not as the same type of this one (namely not a cluster that have all the users that did not open the emails). Here we have two cases:
  - we encounter a orange cluster: then we assign to the red cluster the *mean_H* of the orange cluster.
  - we encounter a green cluster: then we assign one random sending hour among the sending hour of the users belonging to the green cluster.

This type of algorithm is very simple and is based on rules. There is no learning. We decided to start with the simplest algorithm that came into our mind because it can give us a baseline. Moreover, starting with simple algorithms is always better because in the worst case it would not work and then we can move to more complex algorithms. But if it works, we spent very few time and we have a baseline in our hands that we can improve.

Unfortunately, this algorithm did not work well in our case. We tested it with all the aforementioned algorithms but no one gave us decent results. The problem is due to the dataset that is very scattered, so we cannot find good clusters. More precisely, we are finding a lot of scattered clusters each one composed by few users.

We decided anyway to describe this simple algorithm because is easy to implement and maybe for someone could work better. It is worth to give it a try at least.

3.2 Supervised learning

Since the unsupervised learning algorithms proved not be effective for our type of problem and dataset, we decided to switch to the supervised learning world.

Supervised learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately, which occurs as part of the cross validation process. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox.

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining: classification and regression:

3.2.1 Classification

Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below. If we want to approach our problem using a classificator, we need to convert our previous labels into classes. In this case, we can just assign as class the time slot at witch the email has been sent (so we would end up in having 24 classes). The problem with this approach is that, after this conversion, the classes are completely uncorrelated each other. But, we want them to be correlated. Suppose that the classification decision boundary lays between the classes 12:00 and 04:00. It could happen that, due some noise, the datapoint belonging to the class 12:00 fall down into the adjacent class which is the 04:00 class. It means that could happen that the model can suggest hours that are completely uncorrelated. What we want is that if our class is 12:00 and there is some noise, the algorithm suggest something close to it, like 11:00. So, due to this problem, classification is not the right approach for our type of problem. So we moved to regression.

3.2.2 Regression

Regression is used to understand the relationship between dependent and independent variables and therefore allows us to train models that keep the correlation between the send time. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms. Now, we are going to describe the different regression algorithm we tried and then we describe the second TSO algorithm we implemented.

3.2.2.1 Random forest Random forest [2], like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction, see the figure below.

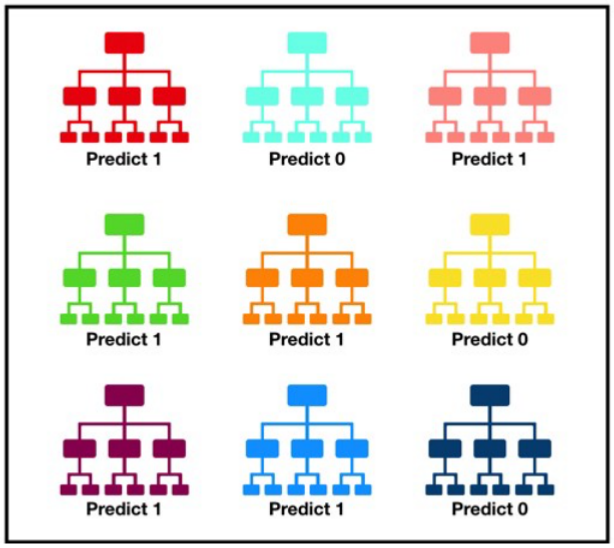


Figure 11: Random forest.

Random forest is a supervised learning algorithm. The ”forest” it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems.

Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there is no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm’s regressor.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

3.2.2.2 Linear regression Linear regression [18] analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a "least squares" method to discover the best-fit line for a set of paired data. You then estimate the value of X (dependent variable) from Y (independent variable).

The objective of linear regression is to minimize the loss function:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where the hypothesis h_θ is given by the linear model:

$$h_\theta = \theta^T x = \theta_0 + \theta_1 x_1$$

The model's parameters are the θ_j values. These are the values that need to be adjusted to minimize cost $J(\theta)$. One way to do this is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update:

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update θ_j for all j). With each step of gradient descent, the parameters θ_j come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

3.2.2.3 Bayesian linear regression

Bayesian linear regression [17] is an approach to linear regression in which the statistical analysis is undertaken within the context of Bayesian inference. When the regression model has errors that have a normal distribution, and if a particular form of prior distribution is assumed, explicit results are available for the posterior probability distributions of the model's parameters.

3.2.2.4 Least angle regression

Least-angle regression (LARS) [13] is an algorithm for fitting linear regression models to high-dimensional data. Suppose we expect a response variable to be determined by a linear combination of a subset of potential covariates. Then the LARS algorithm provides a means of producing an estimate of which variables to include, as well as their coefficients.

Instead of giving a vector result, the LARS solution consists of a curve denoting the solution for each value of the L1 norm of the parameter vector. The algorithm is similar to forward stepwise regression, but instead of including variables at each step, the estimated parameters are increased in a direction equiangular to each one's correlations with the residual.

The basic steps of the Least-angle regression algorithm are:

- start with all coefficients β equal to zero.
- find the predictor x_j most correlated with y.
- increase the coefficient β_j in the direction of the sign of its correlation with y. Take residuals $r = y - \hat{y}$ along the way. Stop when some other predictor x_k has as much correlation with r as x_j has.
- increase (β_j, β_k) in their joint least squares direction, until some other predictor x_m has as much correlation with the residual r.
- increase $(\beta_j, \beta_k, \beta_m)$ in their joint least squares direction, until some other predictor x_n has as much correlation with the residual r.
- continue until: all predictors are in the model.

3.2.2.5 Neural networks Neural networks [9], also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

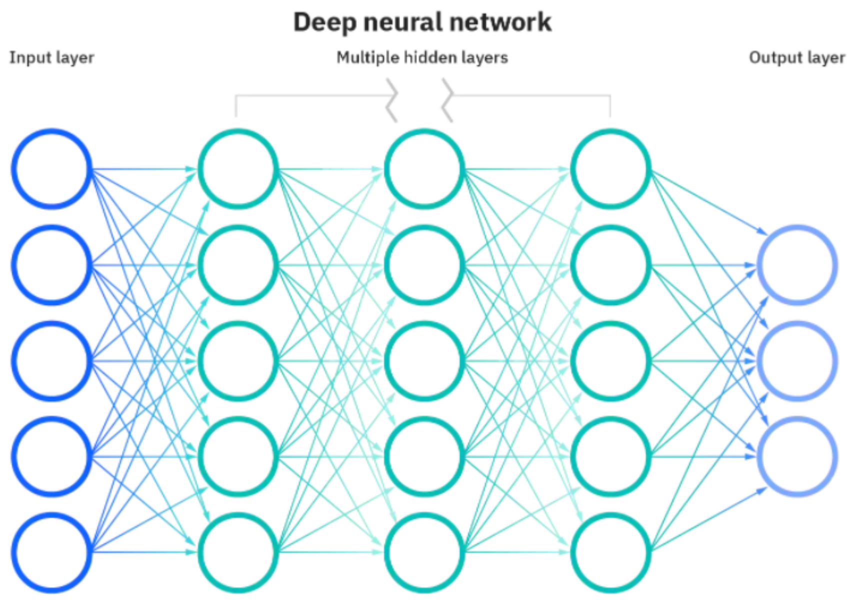
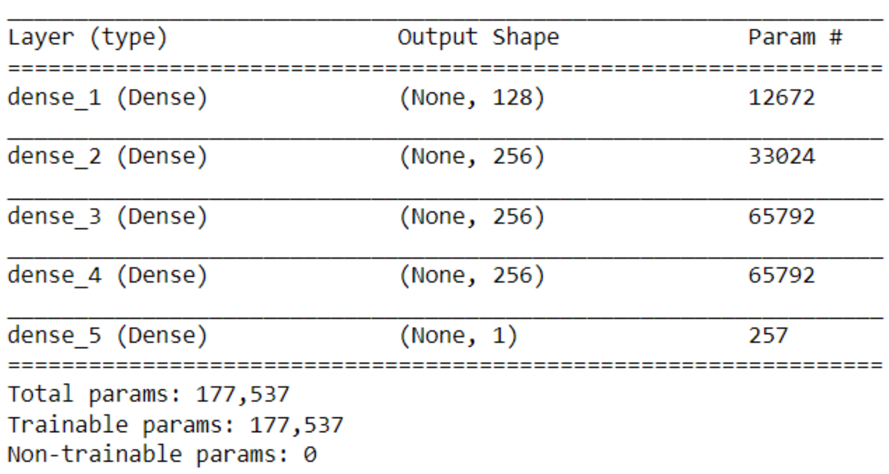


Figure 12: General representation of a neural network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts.

The architecture of the neural network used in our case in the following:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               12672
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 257
=================================================================
Total params: 177,537
Trainable params: 177,537
Non-trainable params: 0
```

Figure 13: Our neural network architecture.

It is composed by 5 dense layers. The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The activation functions used in all the layers but the last one is *relu*, while in the last layer, to perform a regression, we use a *linear* activation function. The loss function to minimize is the *mean absolute error* and the optimizer used is Adam.

3.2.2.6 Convolutional Neural Networks A Convolutional Neural Network (ConvNet/CNN) [8] is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of neurons in the human brain and was inspired by the organization of the visual cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. A collection of such fields overlap to cover the entire visual area.

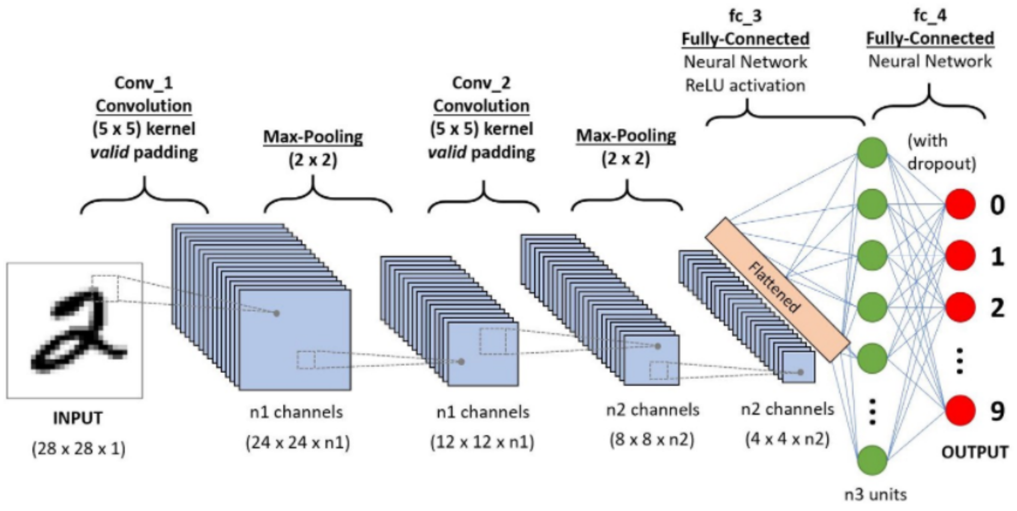


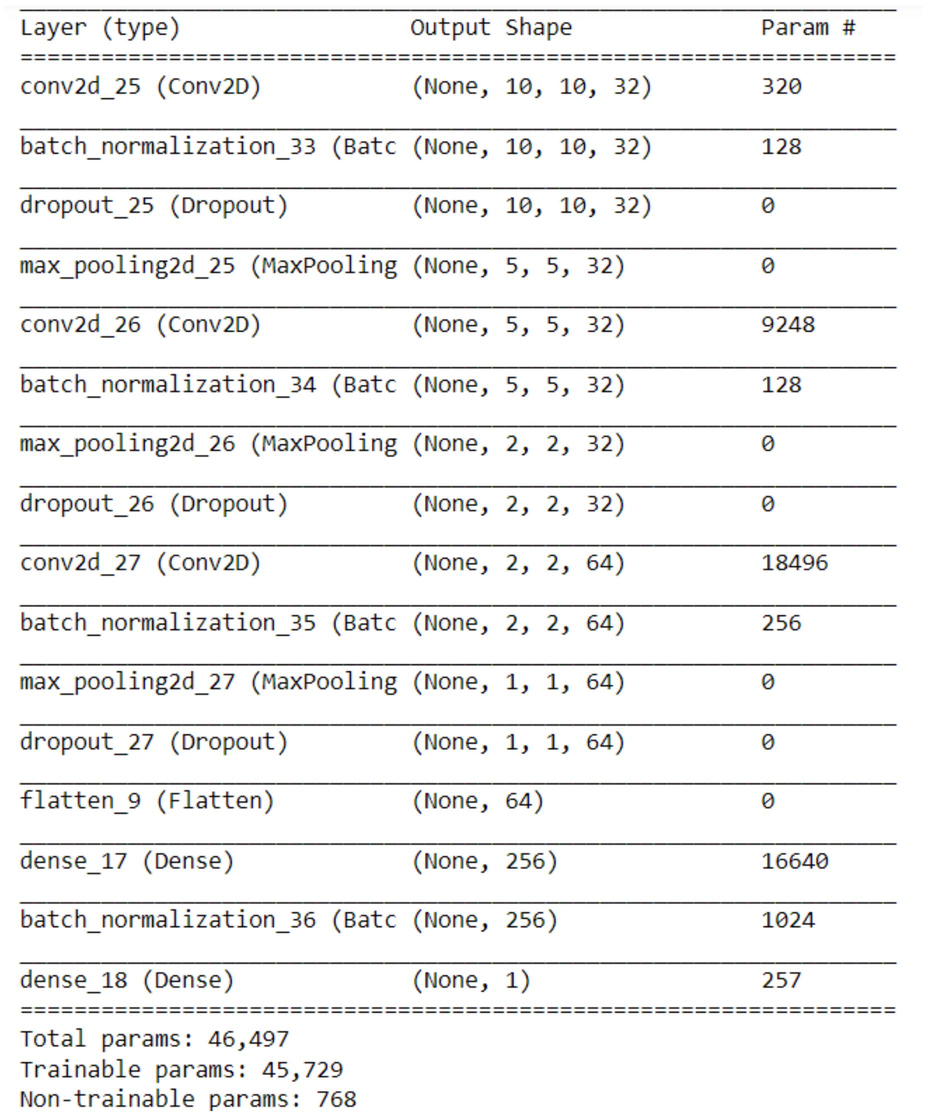
Figure 14: A CNN model to classify handwritten digits.

However, we do not have images, we have tabular data. How to use therefore a CNN? It simple, we just need to take the rows of our dataset and reshape them into a $n \times n$ matrix. However, length of our dataset are 1×98 so we cannot directly create a $n \times n$ matrix. We can proceed in two ways:

- we can perform *feature importance* on our dataset (for instance, random forests allow us to do so) and then take the first number that has as a squared root an integer. In this case this number is 81, since its squared root is 9. Then, this 9 will become the width and height of our row data converted into a grayscale image. However, doing so, we will discard $98 - 81 = 17$ features, and we do not want to remove features, we prefer to retain them.

- the second way to do so, and the way that we actually used, is to increase the columns of our dataset by copying already existing features until we arrive at a number that has as a squared root an integer. In this case, we just need to add two features and we obtain 100 columns and therefore the reshaped rows will became of size 10×10. In this way we are adding redundant information but this is better than removing 17 features.

The architecture of the our convolutional neural network is the following:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_25 (Conv2D)           (None, 10, 10, 32)        320       
_________________________________________________________________
batch_normalization_33 (Batc (None, 10, 10, 32)        128       
_________________________________________________________________
dropout_25 (Dropout)         (None, 10, 10, 32)        0         
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 5, 5, 32)          0         
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 5, 5, 32)          9248      
_________________________________________________________________
batch_normalization_34 (Batc (None, 5, 5, 32)          128       
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 2, 2, 32)          0         
_________________________________________________________________
dropout_26 (Dropout)         (None, 2, 2, 32)          0         
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 2, 2, 64)          18496     
_________________________________________________________________
batch_normalization_35 (Batc (None, 2, 2, 64)          256       
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 1, 1, 64)          0         
_________________________________________________________________
dropout_27 (Dropout)         (None, 1, 1, 64)          0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 64)                0         
_________________________________________________________________
dense_17 (Dense)             (None, 256)               16640     
_________________________________________________________________
batch_normalization_36 (Batc (None, 256)               1024      
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 257       
=================================================================
Total params: 46,497
Trainable params: 45,729
Non-trainable params: 768
```

Figure 15: Our CNN architecture.

In the architecture we can see that there are some layers called *batch normalization* [12], what is that layer? It is a layer that tries to solve the *internal covariance shift*. To understand the internal covariate shift let us make an example: say that I have a 2 layer neural network and we want to learn representations. So, when we train, imagine we are looking at the second layer, this layer is learning taking in input a representation r which is not fixed/completely learnt, this r is changing while training. This means that is like we have layers that are training together, but in reality what we would like to do is to come up with the best representation for the first layer and then train the second one on this best representation, not together.

Think about that a layer learns to capture edges, then we can give this representation to the next layer that from the edges extract the corners. This is what we are hoping to do, however is practice this does not happen. What happens is that the layers are learned together so the second layer receives in input edges that are not stable because the first layer has not finished to learn them. So this problem is actually very bad, because when we want to train something, we want the training set to be fixed, we do not want our distribution to change. But as it is not like that, here each layer sees the input distribution of the things it has to work on that changes.

Batch normalization idea is to counter the previous problem in this way: since the distribution r is changing, I will try to normalize the output of the first layer such that the distribution r does not change too much. We are still learning, but we are constraining what the distribution of r may look like. In particular we will make r to follow a Gaussian distribution, so zero mean and unit variance.

Batch normalization behaves differently at test time: at test time, we do not want to stochastically depend on the other items in the mini-batch, namely we want our output to depend only on the input deterministically. So to counter this problem at test time, I kind of arbitrarily say that the mean and variance will be constants and their value will be a running average of the values seen during the training time. So during the training I will keep a running average of the mean and variance and when training ends I will freeze the values obtained and I use them to compute the test predictions.

The pros of batch normalization are:

- it allows the use of higher learning rates.
- the careful initialization of the weight is less important.

- training is not deterministic, and this is good because it acts as regularization.

While the cons are:

- it is not clear why is so beneficial.

- more complex implementation since need to distinguish between training and testing time.

Another layer we can see in the CNN architecture on the *dropout* layer [6], what is this layer? The idea of dropout is that whenever we have a minibatch and we forward it to the network, we do not use the full network but we use a random subset of it. In particular, according to a random probability p (hyperparameter), some activations are set to zero, so in the end we end up with a very sparse subset of the initial network. Why is this a good idea since we are dropping away a lot of informations? Because dropout prevents feature detectors to co-adapt. Let us make an example to understand it better: say we want to detect a face, in general the faces do not always have all the face attributes like nose, eyes, mouth, etc.. In general something could be missing. So, for example, I would like to train my network also for faces with an occluded eye. So it is a good idea to not having all the features that co-adapt but randomly learn a subset of these features to be able to generalize better on unseen data, so to force my face detector to work also when only some of the attributes are present.

3.3 Metrics

Before building and running our models, a question arises. When we have trained our model, how do I know *how good my model is*? We need a metric to compare the different models we are going to test. Not only, this metric need to reflect the reality. When we will put this model in production, if our metric say that we will improve the open ratio by, say, 15% and the click rate by 5%, this need to be as close as possible to what will happen in the future when the model will work in practice. If we do not build a metric, the basic solution is to train a model, put it in production and wait for n month to collect the new data. And only after the n month we can see how our model performed. This is time-consuming and unfeasible.

In literature, there is not a standard metric to use for this type of problem, so we need to invent one. The idea is the following: if my model predicts a hour to send the email which is closer to the hour at which the user opened the email with respect to the true sent hour, then we have done a good job. If the hour is the same, then we are not improving anything. While, if the predicted hour is before with respect the true sent hour, then we are not doing a good job. So we can create a metric that can tell us on how many samples we are improving, being equal or getting worse. Moreover, we would like to know how much in terms of minutes we are doing better or worse. Because, for instance, we can improve the sent time by 80%, but if we are improving each sending time by 1 minute, we are actually not doing a good job. Also, we added other metrics that facilitates us to compare the models, and they are:

- when I am not doing a good job with the sent prediction, what is the percentage of predictions that are worse by at least 10 minutes with respect the true sending time?
- percentage of prediction when I am doing a good job.
- which is the average improvement in minutes.
- which is the average worsening in minutes.

Another aspect need to be taken: on what set of users do we need to compute this metric? On all users? Maybe there are users that always open the emails, regardless at which time the email has been sent. So maybe it does not make sense to include these users in the test set because I do not care if I improved the sending time for these users since they will always open the emails. So, in the test set, I need to exclude these users. In particular these users are chosen following these constraints. A user will be contained in this category if:

- this user opened all the emails that have been sent to him.

- the 80% of the emails sent to this users have been opened within 15 minutes.

If a user respect these two conditions, then it will be removed from the test set. The numbers you read before (80% and 15 minutes) are hyperparameters that have been properly tuned.

3.4 TSO Algorithm 2 applied on dataset 1

In this section we are going to describe the Time Sent Optimization algorithm 2 solved as a regression problem applied to the dataset 1.

The algorithm follows the following steps:

- choose one regression model and train it on the training set.
- take the trained model and assign 1 (the maximum value) to both features Fitness Send-Open and Fitness Open-Click for all the rows of the test set. This is the key-point. Doing so, we are constraining the model to return us the sending time where these features are at their maximum value, namely the best sending time the model can output.
- get the prediction of the test set after have removed the users that respected the conditions explained before.

When the model will be in production, what will happen when a new user subscribes to the newsletters? We need to send to the user a pre-specified number of emails and then retrain the model again. The same applies for new communication types. Since our dataset is composed by features regarding the communication type, if we have a new communication, how do we need to handle this? Since we have already learn a lot from the users and considering that the communication is not so crucial, the first step is to take the mean of the features of all the other communication types and assign the values we obtain for the new communication type. And, as before, after have sent a pre-specified number of emails for each user for the new communication, we can remove the average computed before and substitute with the communication open and click rate for that communication (like the engineered feature of the second chapter).

3.4.1 Results

In this section we are going to show the results of the TSO Algorithm 2 applied on dataset 1 of the models we have cited before taking into account the metric described before. The results on the test set of the dataset 1 are the following:

Model	% Worse $\geq$ 10'	% Better	Avg worse (min)	Avg better (min)
Random Forest	0.005	0.180	**3.030**	**348.24**
Linear Regression	0.004	0.191	3.098	324.12
LARS	0.004	0.220	3.122	322.09
Neural Networks	0.004	0.231	3.287	312.11
CNN	**0.003**	**0.274**	4.010	322.98

We need to look at these results in this way: we want % Worse $\geq$ 10' to be small, % Better to be high, Avg worse to be small and Avg better to be high. Among the model tested, two models shines particularly, the Random forest and the CNN. Also, there is a general trend: if Avg worse gets higher, then Avg better gets lower. Moreover, if % Worse $\geq$ 10' gets lower, then % Better gets higher.

Even though the CNN has not the best results on all the metrics, what really interests us is the % Better. The CNN has a very high percentage and this should reflect into an improvement of open rate in the future by that value. Remember also that we used a test set excluding the users that open always the emails regardless the time at which they have been sent. So the % Better is a lower bound.

Noteworthy is that the Avg worse is very low, this means that when we are doing worse. We are just anticipating the sent time by a couple of minutes and this if fine for us. What should be very low is the metric % Worse $\geq$ 10', which is so in all our cases. Also, we can notice that we have a very high Avg better. This means that we are taking a big percentage of sending time and moving them by a lot. So, for instance, if the previous sending time was 10:00, now we are moving it by several hours getting closer to a time where is more likely that the user will open the email.

However, this is only our guess. We need to employ a test A/B to see which are the real results.

3.5 TSO Algorithm 3 applied on dataset 2

Here instead, we are going to describe the Time Sent Optimization algorithm 3 solved as a regression problem applied to the dataset 2.

The algorithm follows the following steps:

- choose one regression model and train it on the training set.
- get the prediction of the test set after have removed the users that respected the conditions explained before.

This algorithm is similar to the previous one. What changes here is that we do not need to set the features Fitness Send-Open and Fitness Open-Click to 1 since we do not have them anymore as input, but we are predicting them. This algorithm turns out to work much better than the previous one and is the one we will use in production.

3.5.1 Results

In this section we are going to show the result of the TSO Algorithm 3 applied on dataset 2 of the models we have cited before taking into account the metric described before. The results on the test set of the dataset 2 are the following:

Model	% Worse $\geq$ 10'	% Better	Avg worse (min)	Avg better (min)
Random Forest	0.004	0.192	**2.652**	327.13
Linear Regression	0.004	0.201	2.777	319.22
LARS	0.004	0.257	2.765	**355.05**
Neural Networks	0.003	0.271	3.002	342.27
CNN	**0.003**	**0.310**	3.010	334.52

Looking at the results, compared to the previous ones, we can notice that we have the same patters as before so the same considerations made before applies here too. Also, it can be noticed that almost all the models performed better with this algorithm with respect the previous model. Thus, the final model that we will choose will be this one. In particular we are going to choose the CNN model because has a very high percentage of the Better metric.

3.6 Training and testing

As already said, the final model we have chosen is the CNN trained with the TSO Algorithm 3 applied on dataset 2. To obtain its results, we have accordingly used a training set, validation set and a test set. We have tested several split of these three sets, like: 60 train / 20 validation / 20 test, 60 train / 10 validation / 30 test, 70 train / 15 validation / 15 test. Also, we have experimented with k-fold cross validation [1] with k equals to 3 and 5. At the end, we have chosen as result the split the gave us the worst result to have a realistic measure (a lower bound). We have trained the CNN for 30 epochs using Adam optimizer with β_1 equals to 0.9 and β_2 as 0.999, we used the *mean squared error* loss and we applied also learning rate schedule with step decay of 10. Also, we added into the network some layers to counter overfitting, in particular: batch normalization and dropout layers.

3.6.1 Regularization

Moreover, we applied some advanced regularization techniques to improve our results as much as we can.

3.6.1.1 Ensembles At test time, more often than not, one is used to just take the trained model and run in on the test set and get an accuracy. This is the standard way to test your model. However, we can do better squeezing out the last drop of juice from our trained model using ensembles [11]. Ensembles follow the following pipeline:

- train multiple (randomly initialized) models on the same dataset.
- run each model over a test image.

- average the results (eg. take average of softmax outputs, then take the argmax).

This usually increases the overall performance by 1-2%. Why this should be a good idea? Because if networks have similar error rates, different networks tend to make different mistakes.

3.6.1.2 Snapshot ensembling The previous method works well. However we have to train a lot of networks and this could take a lot of time. To counteract this issue, snapshot ensembling [15] comes to aid. By using a cyclic learning rate schedule, we can simulate M trainings in the time span of one by taking snapshots of the parameters reached at the end of each cycle. We can use a cyclic cosine learning rate which will look like this:

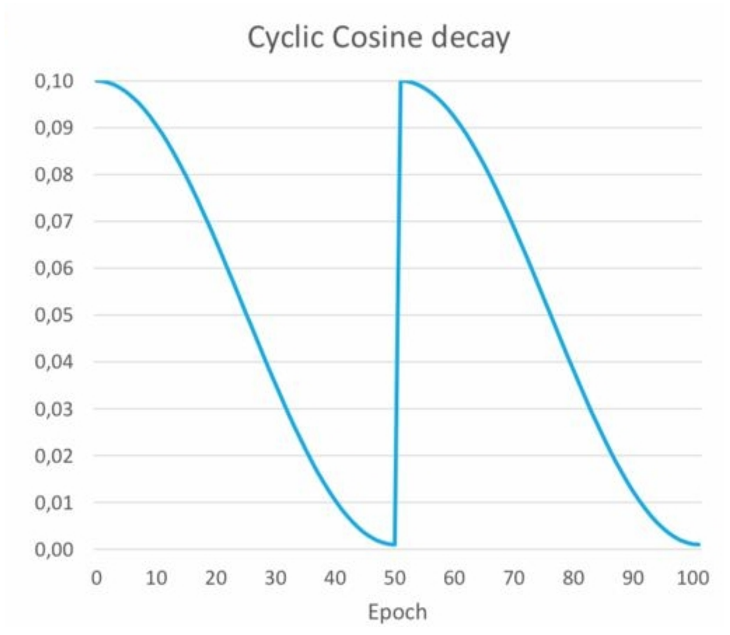


Figure 16: Cyclic cosine learning rate.

So, whenever the cyclic cosine learning rate is decreased, we take a snapshots of the parameters of the model and we save them. Then proceed in doing this until the training has finished. At the end we will end up with M models. By ensembling those models at test time to get the predictions, we usually gain better performance that just doing one cycle of training.

3.6.1.3 Polyak average If we do not want to pay the price of the ensemble at test time (because at runtime we would need to run M models), we would want to get the benefit of ensemble without running ensembles. How to do so? One thing we can do is Polyak average [10] which is basically the idea of updating the parameters in the same way we are doing so far but then we keep another copy of them ($\theta^{(test)}$) where we accumulate every new version of our parameters which we have in a running average fashion. So actually the parameters used at test time will be a running average of the parameters we have seen at training time. Why this is a good idea? Because we will see that the loss will be super noisy. It will bumps a lot between different minibatches. So this Polyak average of the parameters is actually taking the mean of the crazy movements of the loss and on average will give us better performance. The update and the copy of the parameters $\theta^{(test)}$ are shown below:

$$\theta^{(i+1)} = \theta^{(i)} - lr\nabla_\theta L(\theta; D^{(train)})$$
$$\theta^{(test)} = (1-\rho)\theta^{(i+1)} + \rho\theta^{(test)}$$

ρ is an hyperparameter which represents to who give more weight. Being ρ usually very high, we give more weight to the past. $D^{(train)}$ represents the training set, L is the loss, ∇_θ is the gradient and lr is the learning rate.

3.6.1.4 Stochastic Weight Averaging A variation of Polyak average is the so called Stochastic Weight Averaging [16]. Its idea is to combine the multiple annealing, namely the downside curve of the cyclical learning rate. We take only the snapshots when the learning rate in the downside and small and, instead of doing an ensemble of that, we average these weights like we did in Polyak average, but instead of doing it for all the parameters as Polyak does, we average only the good ones (the ones where the learning rate is already decreased and so it could have reached a good minima or something like a minima). This has shown to generalize well. The parameters are calculated in this way:

$$\theta^{(test)} = \frac{\theta^{(cycle)} + n_{cycles}\theta^{(test)}}{n_{cycles} + 1}, n_{cycles} += 1$$

4 Test A/B

We have already have a metric that tell us how well our model perform. However, to have a concrete result, we need to deploy the model and collect the results and compute the new open rate and click rate. A way of doing this is through a A/B test [5].

A/B test consists of testing two different strategies to target a group.

For example, suppose you have a newsletter subscription button - it's called control (the existing element) - and you want to know if something can be improved by changing your newsletters to get more subscribers.

Then, you create another newsletter with a different subscription button: the positioning, the color, the copy of the call to action, the shape, etc. This new element is called a variant.

Now, you show them both to a 50/50 split of your entire audience.

This way you will collect analytic data from both the control and the variant you use as a deciding factor. Which of the two versions attracts the most new subscribers? The most successful version of the newsletter should be your choice.

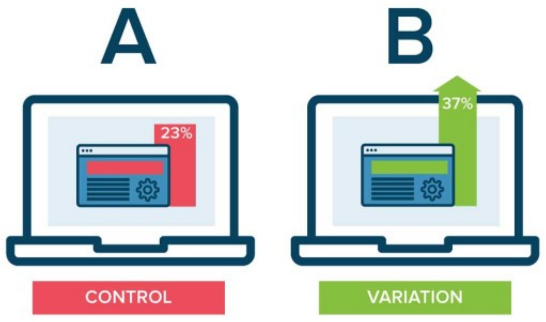


Figure 17: Users' subdivision for A/B test.

4.1 Why A/B testing is important

In short, doing A/B testing will serve as a method of solving problems based on data and statistical measurement. This will allow companies to be better informed and make decisions about their marketing strategies, websites, apps, and so on in a more analytical way.

There are many areas that businesses and marketers can solve and improve by conducting A/B testing. Keep in mind that you have to match the right element that has the greatest impact on each statistic. These areas are as follows.

- conversion rate: doing A/B testing will allow you to see if the alternative converts more visitors into buyers than the original method.
- bounce Rate: with an A/B test you can try out changes like page navigation to see if your visitors stay longer or shorter.
- click-through rate: after making some changes to your site, you can measure whether your visitors are more likely to click on certain links.

4.2 Real results

We created two groups of people. To one group we suggest the sending time suggested by our model, to the other group we send the emails at the hours that the company was used to.

Importantly, we need to keep in mind that we discovered that not all the users are the same. Some of them will open the emails always regardless the time at which the emails have been sent and some of them will open the emails only if sent to a specific hour. So the groups need to contain the same percentage of these users. The chosen split is 50/50.

The A/B test has last three months and the company sent, for each user, one email per day. After the testing, we obtained the following results where the group A represents the group of users to which the model has not been applied where the group B is the group of users to which we applied Time Sent Optimization:

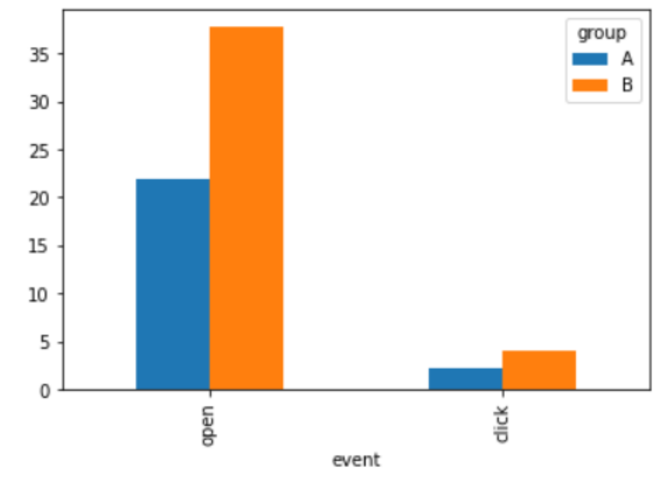


Figure 18: Open and click rate after test A/B.

In blue are shown the open rate and click rate of the group A of users to which the model has not been applied. The percentage are similar to those we computed on the full dataset. Previoulsy, we had 23.72 % as open rate and 2.38 % as click rate. Now, we have slighly less percentage, in particular we have 21.87 % as open rate and 2.12 % ad click rate. The comparison between the results of group A and those of the initial analysis, serve only as a verification that the conditions have not changed from the first data collection to the A / B test, it does not matter so much whether the values are higher or lower, what matters is that the difference is in the area of statistical fluctuations.

On the other hand, the green bars represent the open and click rate of the users of group B which are the users to which we applied Time Sent Optimization. We can directly notice the effectiveness of our model: we improved the open rate by roughly $\sim$ 16 % and the click rate by roughly $\sim$2 % (we almost doubled the percentage of group A's click rate). This is a very good result. The interesting point though is not this one, but is the metric we used. Our best model has predicted that we could have increased the open rate from $\sim$ 23 % to $\sim$ 31 %, but we did better! This is because, in every process of the evaluation, we have chosen to take the results in the worst cases. So that $\sim$ 31 % was a lower bound. This is very useful because if someone wants to try different approaches of this problem, he/she do not need to run the test A/B every time wasting a ton of time. You can just run this metric and obtain a lower bound of what will be the open rate of the model you want to deploy.

5 Conclusions

In this thesis we have shown some possible ways of solving Time Sent Optimization problem. From the business point of view, having this kind of a tool for a company is important because the companies can make their marketing campaigns more effectively since more users open and click the company's communications.

This task is not particularly difficult in itself. However, unlike the other domain, this task has no literature. So we need to invent the algorithm completely from scratch. In particular we have to understand how to create the dataset and which set of features to employ, how the label is structured and which metric for the evaluation need to be used.

Our first attempt was to solve this problem into the unsupervised learning domain. However, our dataset was too scattered to perform operations like clustering. So we moved the world of supervised learning. However, moving to supervised learning is not a straight step because we need to understand how our label will be structured since there could be many options. After several attempts, we decided to opt for a label which represents a histogram that indicated which is the users' fintness for all the possible hours. Then, we tried to solve the problem as classification. However this turns out to not be effective because in the classification world, the labels are uncorrelated each other, but for our problem we want them to be correlated. Therefore we switched to solve a regression problem. We build two different algorithms and tested several regression model but the best one was the model based on Convolutional Neural Networks with, what we called, dataset 2. However, this worked for our case. The variance of the distribution of the data is huge for for this specific problem, so maybe for someone this method could not work. Therefore, we suggest to test all the models we presented in this thesis.

5.1 Future works

This type of problem is really data-dependent. In particular, the more data we have the better. So, as a primary future work, is necessary to continue to collect new data from the users and retrain the model again and test it with the metric proposed in this thesis.

In spite of we reached a higher open rate compared to the based one, we improved by a little the click rate. This is because we concentrated mainly on improving the open rate assuming that if I increase the open rate, then the click rate will increase accordingly. This is true in part but not in general. So, as a future work, it could make sense to work directly also on increasing the click rate. This could be done by creating a model that writes emails' subjects and email content that are attractive to the user. This is a Natural Language Problem which is not related with Time Sent Optimization, but we think that combining these solution could lead into even better results, mostly on the click rate.

Another interesting aspect that one can explore is to solve this problem as a classification problem. We said that classification is not suited for this kind of problem because the labels must be correlated each other and in classification the labels are not correlated. But classification has a lot of pros and we think that it could work better than regression. The thing that one may explore is how to make the labels correlated each other.

References

[1] Daniel Berrar. Cross-Validation. *Reference Module in Life Sciences*, 2014.

[2] Leo Breiman. Random Forests. *arXiv*, 2001.

[3] Sidharth Mishra; Uttam Sarkar; Subhash Taraphder; Sanjoy Datta. Principal Component Analysis. *research gate*, 2017.

[4] Seyed Esmaeili; Brian Brubach; Leonidas Tsepenekas; John Dickerson. Probabilistic Fair Clustering. *arXiv*, 2020.

[5] Cyrille Dubarry. Confidence intervals for AB-test. *arXiv*, 2015.

[6] Nitish Srivastava; Geoffrey Hinton. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014.

[7] Sakshi Patel; Shivani Sihmar; Aman Jatain. A study of hierarchical clustering algorithms. *IEEExplore*, 2015.

[8] Jiuxiang Gua; Zhenhua Wangb; Jason Kuenb. Recent Advances in Convolutional Neural Networks. *arXiv*, 2018.

[9] B. Mehlig. Machine learning with neural networks. *arXiv*, 2021.

[10] Damien Scieur; Fabian Pedregosa. Universal Average-Case Optimality of Polyak Momentum. *arXiv*, 2021.

[11] M.A. Ganaie; Minghui Hu; M. Tanveer Suganthan. Ensemble deep learning: A review. *arXiv*, 2021.

[12] Sergey Ioffe; Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*, 2015.

[13] Bradley Efron; Trevor Hastie; Iain Johnstone; Rob Tibshirani. Least Angle Regression. *research gate*, 2004.

[14] Foxiao Zhan; Xiaolan Zhu; Lei Zhang Xuexi Wang. Association Rules. *arXiv*, 2019.

[15] Gao Huang; Yixuan Li; Geoff Pleiss; Zhuang Liu; John E. Hopcroft; Kilian Q. Weinberger. Snapshot Ensembles: Train 1, get M for free. *arXiv*, 2017.

[16] Pavel Izmailov; Dmitrii Podoprikhin; Timur Garipov; Dmitry Vetrov; Andrew Gordon Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv*, 2019.

[17] Bruna Wundervald. Bayesian Linear Regression. *research gate*, 2019.

[18] Khushbu Kumari; Suniti Yadav. Linear Regression analysis study. *research gate*, 2018.

[19] Shi Na; Liu Xumin; Guan Yong. Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm. *IEEExplore*, 2010.