Università degli studi di Padova

Department of Mathematics "Tullio Levi-Civita"

Master's degree in Data Science

Federated Learning for an Autoencoder Based Recommender System for Look-Alike Modeling

Supervisor: Professor Michele Rossi
Co-supervisor: Mirko Polato

Candidate: Giovanni Vedana

21st September 2022

Abstract

User segmentation based on browsing history is a cornerstone of online advertising: it allows companies to reach a targeted group of subjects that could be interested into their product. Furthermore, the look-alike modeling is a powerful tool that can enlarge the audience while including meaningful potential customers.

The upcoming end of third-party cookies and privacy regulations limit the companies' power to freely access user data: this makes online targeting more and more challenging and it calls for the development of a way to understand user's interests without the need to access their data.

This work offers a solution in this direction, suggesting a "zero-trust" process for the segmentation of the users by means of the look-alike modeling: this process uses well-known data science fields, as recommender systems, and it applies them in a whole new context, combined with groundbreaking techniques such as Federated Learning, an innovative privacy-preserving way of training machine learning models. It allows to build a procedure fully compliant with user privacy and with the end of third-party cookies.

Contents

List of Figures

List of Tables

List of Acronyms

GDPR General Data Protection Regulation

FL Federated Learning

RS Recommender System

AE Autoencoder

VAE Variational Autoencoder

ML Machine Learning

PPML Privacy Preserving Machine Learning

RTB Real Time Bidding

NLP Natural Language Processing

NN Neural Network

RNN Recurrent Neural Network

NN Neural Network

CF Collaborative Filtering

CB Content-based

NN Neural Network

SGD Stochastic Gradient Descent

1 Introduction

The main focus of this thesis is the creation of a zero-trust process for the segmentation of the users by means of the look-alikes, in the context of digital advertising. It is based on a recommender system that aims at suggesting new categories to users based on their previous browsing activity. This model was trained with Federated Learning, a privacy preserving technique for training machine learning models: this allows to build an entire segmentation process without collecting the user data, in a full accordance with the GDPR (General Data Protection Regulation) and with the end of third party cookies.

The research of a proper recommender system, trained and refined in a federated environment, is intended for ID Ward Ltd, a company that works in the field of digital advertising with a focus on the data privacy issue. The content of this work will be deployed by the company with the aim of providing for each user values of appreciation for categories of websites.

The work is structured as follows: in Section 2 we expose the motivations that led to this project, with a deep insight on the field of digital advertising, explaining the reasons that require the use of a privacy preserving method for data-driven models.

In Section 3 we present the main previous works related to this thesis: firstly, we provide an overview of the different types of recommender systems, with a brief insight on the state-of-the-art approaches. After that, we go through some previous application of federated learning for recommendations. In addition to this, we focus on the Google's attempt to overcome the end of third party cookies (FLoC), dwelling on the issues brought by this solution.

In Section 4 we present the main contributions of this work, that consist firstly in the proposal of a simple but innovative model for this context, that performs well this task. Furthermore, among the metrics taken into account we propose a specific metric based on a taxonomy: it turned out to represent well the overall performance of the model. Besides, we provide a solution for the continual learning of the model in a federated setting: it allows to integrate the information about the new data into the model without losing the knowledge acquired during the previous training sessions.

In Section 5 we focus on the dataset we worked with, presenting how the data was collected and analyzing its distribution. Then, we present the data processing pipeline, which analyzed the data prior to feeding it to the recommender system.

Then, in Section 6 we focus on the method used for training and evaluating the model. We start by presenting the specific training procedure adopted for the learning of the model. After that, we dive deeper in the explanations

of the metrics used for the evaluation: we considered ranking metrics (HIT), threshold metrics (Precision and Recall) and a taxonomy-based custom metric.

In Section 7 we present the architecture of the model adopted for the recommendations: it consists in an autoencoder-based network with the addition of textual information (the titles of the URLs or the entities). We show the results obtained by tuning the hyperparameters and changing the starting features; furthermore, we show how the textual information affects the final suggestions. We then propose a simple approach for the refinement of the model on new data in a centralized setting. Finally, we present how the processing pipeline can be implemented in a distributed setup.

Section 8 is dedicated to Federated Learning: we start by presenting the optimization problem we aim to solve and then we compare three different federated algorithms (FedSGD, FedAdam and FedAVG) noticing a clear advantage of FedAdam over the others. Then, we propose a solution for the continual learning of the recommender system in a federated setting, based on the idea of reconstructing the original training dataset, since the actual data is not available with federated learning.

Finally, in Section 9 we conclude our work and present future developments.

2 Motivations

This work was possible thanks to a collaboration between the University of Padua and ID Ward Ltd, that is a data privacy company that acts in the context of digital advertising.

This field grew rapidly in the last 20 years due to the massive growth of the Internet, that required a brand-new way to show advertisements to users. Indeed, at the beginning of digital advertising ad spaces were bought through direct deals by *advertisers*, namely the company who want to promote their brand, and ads were traded in the same way of magazine ads: the advertiser and the *publisher* (the company who sells the inventory) reached an agreement for a specific time period for a particular website. This means that the ads were showed indistinctly to all the users that browsed the web-page of a publisher, without taking into account information about user preferences.

This method was very inefficient for two main reasons: first, the majority of the targeted people were not interested in the product showed, because usually the potential customers are only a small slice of the entire audience that browses a web-page. Second, the number of websites grew faster than the number of companies willing to advertise on them, and the manual trades did not keep up with this pace, with the consequence that many ad spaces remained unsold.

As a solution to these issues was created a procedure that manages the market of digital advertising automatically, without the need of human negotiations: it is called *Programmatic Advertising*. Several different factors handle this process, with the aim to guarantee an advantage to both the sides of the trade. With programmatic advertising trades usually take place in real-time when the user clicks a web-page, through a bid that involves the companies interested in targeting that particular user (called *Real-time Bidding*, RTB). This allows to overcome the problem of unsold inventories together with the targeting issue.

Here, machine learning and AI algorithms are useful because they allow to understand the characteristic of the visitor starting from their data, allowing the proper targeting of a group of users.

Programmatic Advertising

Programmatic Advertising is the process that aims at automating the market of online advertising. In order to use as minimum human involvement as possible there are different components that take part to this procedure:

- **Ad server**: it creates, manages and runs the advertisement campaign. It is the server that decides the ad to show on the website and records the metrics regarding the campaign (e.g. impressions and clicks)
- **SSP** (Supply-Side Platform): it is the platform through which publishers manage and sell their inventories.

- **DSP** (Demand-Side Platform): it is the platform through which advertisers can buy multiple inventories from different websites.
- **Ad network**: it is the intermediary between publishers and advertisers. It handles the RTB.
- **DMP** (Data Management Platform): it collects and processes the data coming from different websites. It is responsible for the segmentation and for the look-alikes modeling. An example of DSP is ID Ward Ltd, who creates user segments based on their browsing histories.

There are two main ways in which advertisers can purchase ad spaces from publishers: through programmatic directs or through the RTB. In the first case the trade is managed by the Ad network and once the inventory and the CPM (cost-per-thousand impressions) are set, the procedure is handled automatically by the Ad network. In the second case the inventory is sold through a bidding that is done among all the advertisers interested at targeting a particular subject, depending on the user data and on the website. This allows to reach a proper group of users, whose characteristics are hopefully similar to those of their most profitable existing customers.

The offer to the bid depends on the type of target policy adopted by the company. With *Contextual Targeting* the advertisers target users depending exclusively on the data of the website the ad space belongs to, such as the keywords. It allows to not use the visitor data, overcoming data privacy issues, but at the same time it brings several drawbacks. First of all, some websites are too broad for a precise targeting and, moreover, others could show an interest that cannot be translated into a real purchase interest without the help of other websites previously clicked (i.e. news websites).

On the other hand, *Behavioral Targeting* focuses on the visitor data, targeting a user depending on its profile, which is created based on their browsing history. For this targeting method typically three different types of information can be used: user's interests, the demographic data and their purchase intent. As far as this work is concerned, we focused on the first case, and more in particular at how companies can target a proper group of users (*segment*) and how they can enlarge it through the *look-alike modeling*.

Segmentation and Look-alike modeling

In digital advertising the users are usually clustered into several groups, called *segments* (or *cohorts*), composed of users that share similar characteristics. Once understood what are the attributes that distinguish its most profitable customers, a company aims to target only people with similar characteristics, in order to make the advertising campaign as effective as possible. A mountain clothing company, for instance, could be interested in targeting only users that like to hike and with age between 20 and 30.

Even if already effective, this process has some limitations and can be improved. First of all, the segment created by a company might be too small

for the company's needs and a company might want to enlarge the audience while maintaining a group of users that could be interested into their product. Then, this procedure takes into account only people who are known to have those characteristics, while due to a lack of data it could be that some other people still reflect those attributes even if they were not recorded in a page view. The *Look-alike modeling* expands the segmentation process by solving these issues.

With this method, indeed, advertisers are able to enlarge the segments also including users that look alike their customers and have characteristics correlated with those the company is looking for, and as a consequence they could be interested also in what the company has to offer. In the previous example, for instance, the mountain clothing company could decide to also target people interested in athletics, swimming or travels: by doing this the overall audience still is composed by meaningful users and it contains a larger number of users.

In this work we implemented a model that provides the look-alike modeling starting from the browsing activity of the users, using information such as the date of the click view and the time spent on the website along with data regarding the content of the website (the title and the entities). For this task we built a recommender system, whose purpose is to suggest new categories to the users depending on the previously browsed categories. As output, the model provides a value that can be seen as the probability that the user is interested into a particular category: by doing this the advertisers can choose a "look-alike threshold", establishing which users belong to the segment choosing only those whose probability is above the threshold.

Cookies

Nowadays Behavioral Targeting is possible through information that is stored in the local storage of the user and that allows to keep track of their activity across different websites. These pieces of information are called "cookies", and they have been a cornerstone in the history of digital advertising. They correspond to snippets of code that run when the user clicks a web-page and record information about the click view, such as the time spent on the page, the location, email, passwords and possibly clicks on ads. Among many other things, they allow the user to have a customized experience on the site, since they store data about the items in shopping list and maintain the users in their account for future sessions. There are essentially two types of cookies: first-party cookies and third-party cookies.

With first-party cookies we mean that information that is stored in the user's device directly by the website the user is browsing. This type of cookies have the main purpose to provide a personalized service to the user, recording their activity only for what concerns that particular domain.

On the other hand, third-party cookies are created by other domains, different from the one visited by the user, hence the name third-party cookies.

They are mainly used for tracking and online-advertising purposes; they allow to track the user activity among multiple websites, allowing to obtain an accurate user profile. Typically this is done by a request sent from the web page to the third party's server. The request is usually done by files that are stored in the page, such as ads, images, buttons or tracking pixels. Usually these elements do not need to be clicked for firing the request. A social-media site, for instance, can place a button in an external site, allowing to record the activity on that particular domain: the consequence is that the next time that the user will navigate on their social account they will see an ad regarding a topic previously seen on another domain.

An important turning point in this scenario will be in 2024, when Google, Apple and Mozilla will turn off the third-party cookies, with the goal of strengthening online privacy by giving users more control, choice, and transparency when it comes to personalized online advertising. This will clearly also change the way digital advertising works, forcing advertisers to find a new way to target people based on their previous behavior.

Here, ID Ward Ltd proposes a solution, that consists in a product totally compliant with the end of third-party cookies. After having obtained the user's consent, it collects the data in a central system, where it creates the segments based on the whole user activity, without the need of third-party cookies. After the audience is created, to each user is associated a set of indices, that are the indices of the segments the user belongs to. From these numbers it is not possible to obtain the browsing history of the user, since only the company that created the segment knows the meaning of the segment. On the other hand, the company can decide to target all the people belonging to that particular segment, without the possibility to access the users IDs.

Clearly, this solution has an important limitation, namely the fact that the user needs to trust that the company will not share their data with third parties. This can be overcome by moving the segment creation on the user device, without the need for it to leave the device: this is possible with Federated Learning.

GDPR

The General Data Protection Regulation, or GDPR, is a European regulation that came into force in 2018, and whose goal is to regulate all industries and companies dealing with personal data, giving to data subjects more control on their data.

First, it states that the companies must implement measures to protect personal data (with encryption and pseudonymisation). With "personal data" the GDPR refers to "any information relating to an identified or identifiable natural person (*data subject*); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental,

economic, cultural or social identity of that natural person" (GDPR, Art.4).

Furthermore, it regulates the creation of users' profiles based on personal information obtained without any consent, the use of that data in automated decisions and the unsafe storage of it. In particular, it requires that the company obtains personal data only after a clear, unambiguous and explicit consent from the data owner for every type of data-processing activity. This affects digital marketing since usually user profiles are created based on information obtained by third parties, but this is punished by GDPR with significant fines (up to 20M€ or 4% of the previous year's turnover for serious infringement).

Despite this, it is not easy to verify that the requirements of the GDPR are met, and so that the data is not sold or exchanged to third parties. Indeed, there are still companies that share and obtain data without the user consent since the gain they receive is higher than the punishment they would receive for the violation. This problem can only be overcome by reducing companies' ability to collect personal data and preventing them from exchanging it, still providing a tailored experience to the user.

Zero-trust process

In this work we propose a solution for a proper segmentation of the users based on behavioral targeting in a cookieless environment, in full compliant with the GDPR and with the end of third-party cookies. For this task we used Federated Learning, a machine learning technique for training data-driven models that does not require the direct collection and centralization of data. Indeed, with a federated approach it is possible to train a model without the need to access the user data, and so without the possibility for the companies to share the data with third-parties. For this reason this type of process is called "zero-trust", because the users do not have to trust that the company will not share the data with third parties, and this simply because they do not have it.

In this process the recommender system for look-alike modeling is trained in a federated environment on data about the whole browsing history of the user that is stored completely on the user device. By using information on the device, we insist, there is no possibility to exchange data with third-party cookies, in accordance with the future end of third-party cookies.

In addition to this, the inference part takes place in the user device as well, computing the look-alike values for the users and computing to which segments they belong to. After this step, the only information that leaves the device are the indices of these segments: to each segment, indeed, is assigned a random segment ID that can change over time and only the company that creates the segment knows what it refers to. This limits the possibility to fingerprint a user, that means that it is difficult to identify the user starting from the segments it belongs to.

3 Related Work

3.1 FLoC

FLoC (Federated Learning of Cohorts) is in attempt made by Google that has the aim to supply a cookieless alternative for the profiling of the users. FLoC is part of the Privacy Sandbox, a family of technologies developed by Google that aim at protecting online privacy.

According to Google's tests, FLoC achieves at least 95% of the conversions per dollar spent compared to cookie-based advertising, but the way in which it is built hides several issues and limitations that do not make it a feasible solution for the future cookieless world.

Starting from the browsing history of the user, FLoC maps them into a particular cohort, consisting of users that share similar characteristics. This association is done through an unsupervised learning algorithm called *SimHash*, that takes as input a d-dimensional vector x and maps it in a p-bit vector $H_p(x)$, called "hash vector", representing the cohort ID. This is done by taking p random unit-vectors $w_1, ..., w_p$ and computing the i-th entry of $H_p(x)$ as

$$H_p(x)_i = 0 \text{ if } w_i \cdot x \leq 0 \qquad H_p(x)_i = 1 \text{ if } w_i \cdot x > 0.$$

SimHash allows similar vectors to be mapped in the same cohort, since the probability that this happens is $Prob(H_p(x_1) = H_p(x_2)) = (1 - \frac{\theta(x_1,x_2)}{\pi})^p$, where $\theta(x_1, x_2)$ corresponds to the angle between the two vectors. The computation of the hash vector is done independently to the other users, and so SimHash can be computed locally on each user's machine, with no need for a central server to collect behavioral data.

In order to ensure privacy and prevent fingerprinting, FLoC requires that each cohort verifies a k-anonymity condition, meaning that each cohort needs to be composed by least k different users: this indeed makes the derivation of individual behaviors across the web harder and allows to a user to hide in the crowd. In particular, Google decided to use k equal to a thousand of users.

Unfortunately SimHash does not allow to set a threshold on the minimum number of users for each cohort so the choice of p becomes important, since a too large value of p would create cohorts that does not comply with the k-anonymity, meaning that advertisers learn more about each user's interests and have an easier time fingerprinting them. On the other hand, a too small value of p would not allow to properly catch the peculiarities of each group.

Google's experiment used 8-bit cohort identifiers, meaning that there were only 256 possible cohorts, leading to the first drawback of FLoC: it is very blunt and not specific, reducing very detailed behavior into broader interests that does not represent well the user.

In addition to this, the cohort IDs are fixed, and this because the vectors $w_1, ..., w_p$ do not change over time; otherwise it would be impossible for the companies to target the right group of people since they would not know to

which cohort their past customers belong to, given the new vectors. This allows trackers to easily fingerprint the users, because they can keep track of the segment they are part of and finding out the behaviors that identify them. This could not be possible if the cohort ID would change randomly over time, without the possibility to reconstruct the user's behavior starting from it - which is what ID Ward Ltd does.

Another relevant problem of FLoC is that the cohort ID could be used as identifiers: since the cluster algorithm used does not follow any specific rule for the creation of the clusters, it could happen that a cohort is composed by people that browsed a particular website, or it could reveal other general information such as the interests or demographic information. FLoC, so, may exacerbate many of the worst non privacy problems of behavioral advertising, such as discrimination and predatory targeting. Indeed it can happen that a company identifies a group that consist exclusively of people belonging to a particular social class or have a specific sexual orientation and decides to target them only (discrimination). On the other hand it could happen that a company leverages vulnerabilities or weaknesses of potential customers in order to turn them into actual ones (predatory targeting).

This issue can be overcome by totally discarding sensitive categories and making a categorization only based on the remaining categories, building cohorts exclusively based on them. In particular, starting from the classic taxonomy used for digital advertising, in this work we dropped categories about medical issues, gambling, politics and religion.

3.2 Recommender Systems

A Recommender System (RS) is an algorithm that aims at suggesting to users new items that are probably interesting for them, based on different aspects such as their previous behaviors, their demographic information or their needs. This type of model is becoming more and more popular thanks to a lot of different applications that need to provide to a subject a personalized recommendation. This could be due to several reasons: firstly, it is useful for increasing sales since a good recommendation could make the difference between a normal click-view and an actual purchase. Secondly, it could be beneficial for the users, since in many contexts it happens that the overall catalogue of items is so large that for the users it is difficult to find by their own what they could like. That is, for instance, the case of companies such as Netflix, Amazon and YouTube, where the number of items is huge and it increases every day.

Basically the types of data that can be exploited in a recommender system are divided in two main categories:

- *Explicit* data, that refers to the information that is directly asked to the users, such as scores of appreciation for particular items or even textual feedbacks.

- *Implicit* data, that is the data that is collected based on indirect behaviors of the users, without the need of addressing them. For example, data on whether users clicked or not a particular item or the amount of time they spent on a particular web-page.

Recommender Systems are divided into many categories depending on the type of information they use, but for the aim of this work we focus on Collaborative Recommender Systems, Content based Recommender System and Hybrid Recommender Systems, since the model developed in Section 7 will fall into these.

Collaborative Recommender System

This type of RS is widely implemented since it aims at catching the similarities of the users based on their behaviors and it wants to generate new recommendations depending on inter-user comparisons. This means that if a first user is interested in the items A, B and C while a second user is interested in A and B, the algorithm will suggest the item C to the last user. This aspect is called *Collaborative Filtering* (CF) and it refers to the fact that users cooperate with each other to provide proper suggestions. Collaborative filtering is based on the assumption that people who agreed in the past will agree also in the future and that they will like objects similar to those they liked in the past.

Content based Recommender System

With Content based (CB) Recommender Systems we mean those RS whose suggestions are based on the user's profile using features extracted from the content and the attributes of the items that the user liked in the past. Unlike the CF case, here the recommendations are not influenced by the other users' profiles. This is because with this type of RS we recommend those items that are mostly related to the items already seen in the past, without taking into account the relations between the users.

Hybrid Recommender System

Moreover, there are numerous RS that combine the positive aspects of different recommenders in order to improve the quality of the suggestions. These algorithms are called Hybrid RS and many companies nowadays use these models: while the huge amount of users could suggest to use a CF approach, at the same time the users could look for items similar to those seen in the past, thus suggesting a CB approach. The model developed in Section 7 belongs to this category, since we took advantage of the information about the content of the pages and we used a collaborative approach.

3.2.1 Deep Learning Recommender Systems

The importance of Deep Learning increased abruptly in the last years, not only in fields such as Computer Vision (CV) and Natural Language Processing (NLP), but also in the context of recommendations. This is due to some advantages carried by neural networks: first of all in CV and in NLP usually the tasks have a clear temporal or spatial structure that can be easily handled by appropriate neural networks such as Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

In the same way, also for the recommendation task often the data has a temporal structure: this is for example the case of Click-Through Rate Predictions, where the aim is to predict the next click of a user (that could correspond to a video, advertising, purchase etc...) depending on their previous clicks. In this context, indeed, it could be more appropriate to face this task with a model that takes into account the sequential behavior of the data.

Moreover, Deep Learning is becoming increasingly more important in this context since it allows to exploit the Collaborative Filtering approach for recommendations, exploiting a sort of people-to-people correlations. This behavior is present in neural networks because in the training process the weights are able to catch the correlations between the items, allowing to suggest to users items that are already interesting for other users that have similar characteristics.

While CF is able of accurately recommending a wide range of products without the need of understanding the essence of a particular item, Deep Learning provides a whole new level of sophistication: DL indeed allows to further improve the recommendations by combining the CF aspect with content-based methods. This can be done by also exploiting side information about the items or the user itself (the textual information, in our case). In [10] for instance, *Karamanolakis et al.* use a Hybrid RS combining ratings with explicit text feedback from the user in a Variational Autoencoder (VAE), one of the state-of-the-art recommender systems. Usually also additional demographical features are able to provide better recommendations ([6]): here, for example, *Covington et al.* exploit a geographic embedding in order to provide different recommendations depending on the location of the device.

3.3 Federated Learning for Recommender Systems

In the last decade the increasing interest in user privacy has led to the need of privacy preserving machine learning models. A solution in this direction was made by Google in 2016, when proposed Federated Learning, a technique for training data-driven models where the learning takes place in the users machines. A central system, then, has the purpose to update the global model by agglomerating the models obtained from the training on different devices. This consents to learn an overall model without the need to collect and inspect the data.

The interest for this paradigm, in contrast to the traditional way of collecting, storing and processing user data on a server beyond the user's control, brought to several researches regarding the federated optimization problem. The first and most straight-forward attempt was to apply the Stochastic Gradient Descent in a federated environment. Following this way, *McMahan et al.* [13] tried to improve this algorithm by proposing a more communication-efficient approach based on a simple mean aggregation in the server side, combining models trained locally with several iterations (FedAVG). In addition to this, *Reddi et al.* in [18] proposed federated versions of adaptive optimizers (AdaGrad, Adam and Yogi). This last attempt had the purpose to solve the drift data problem to which federated learning is subject.

Even if it is still a not very explored area, in the last years were made many attempts to adapt recommender systems into a federated environment, since this would solve many limitations resulting from the direct access to the client data. An early milestone was the federated version of the matrix factorization recommendation model, proposed by *Ammad-ud-din et al.* in [2]. The matrix factorization model is a particular recommender system that aims at reconstructing the ratings of the users to the items by means of proper embedding representations of users and items. In particular, if $x_u \in \mathbb{R}^k$ is the vector representation of user u and $y_i \in \mathbb{R}^k$ is the representation of item i the prediction of the rating of user u to the item i is given by $\hat{r}_{ui} = x_u^T y_i$. The learning is done by minimizing a proper loss function between the matrix of the predicted ratings $\hat{R}$ and the actual ratings R. The optimization algorithm used in this context is usually the ALS (Alternating Least Squares), that aims at finding the optimal values of the users and items matrices X and Y by updating one matrix at the time alternatively.

The first federated solution proposed for this model in [2] consists in the simple intuition that the item vectors are placed on a central server and the user vectors reside locally on each edge device. The user factor vectors are updated locally on the client machine using the user's own data and the item representation from the server. The main drawback of this approach is that it requires clients to retain the state, specifically their user embeddings, across rounds. In cross-device FL contexts, indeed, stateful federated algorithms are less appropriate because the population size is typically much bigger than the number of clients who engage in each round, and a client normally participates only once during the training process [22]. In this sense an improving of the technique described above was proposed, leading to the Federated Reconstruction algorithm [21]: it is basically a stateless version of the previous algorithm, where the main modification lies in the storing of the user representation. In this case, indeed, the user vector is not stored in the local machine but is reconstructed every time a user is selected for the training. This is done through steps of the SGD once the item matrix I is sent to the client.

As far as the recommendation purpose is concerned, several research works focused on how to improve the privacy preserving aspect of federated learning,

stating that this technique protects the user data only partially, since private information can still leak with FL. More precisely, after local computations, clients should transmit their computed parameters to the coordination server for aggregation and looking at the update brought by a user it is possible to understand the behaviors of the client. To overcome this problem several privacy aggregation methods were proposed, such as the use of homomorphic encryption, the use of a Secure Multiparty Computation aggregation based on a variant of FedAVG [19] and through the use of differential privacy, in which an artificial noise is added to the parameters of the clients [25]. Another solution was suggested by *Perifanis et al.* in [16], where they propose an enhancement by decomposing the aggregation step into matrix factorization and neural network-based averaging for an item recommendation purpose based on NCF (Neural Collaborative Filtering).

Another privacy risk related to federated learning regards the fact that if the recommender system has full access to all the contextual attributes then it might have a serious consequence if an attacker can get or infer this information collectively. This issue is then partially overcome in [1], where *Waqar et al.* proposed a federated learning-based privacy-preserving model for context-aware recommender systems that works with a user-defined collaboration protocol to ensure users' privacy. This idea lets the user decide how much data share with the local model, preventing at least in part privacy risks.

4 Main Contributions

The main contribution of this work lies in the creation of a zero-trust process for the user segmentation through look-alikes values: as far as we know this is the first attempt to overcome the end of third party cookies with the use of a federated learning recommender system.

Other than the design of the entire process, this work brings some research results to the fields of recommender systems and privacy-preserving machine learning. Here we provide innovative aspects in both these areas, combining well-known data science techniques in a new context.

In the context of recommender systems, we tested and implemented a particular model that to our knowledge is unique in this context. This is because of the specific training procedure we decided to adopt and for the addition of textual information about the web-pages browsed by the users, rather than only the information of the category the site is about. Besides, we provide insights on how this additional information affects the final suggestions.

Moreover, we give an example of how to measure the value of interest (*score of interest*) for a specific category, based on the recency, frequency and time spent by a user on websites of that category.

In addition to this, we provide a suitable way of evaluating the model, considering a particular metric that takes into account how far in the taxonomy our suggestions are from the suggestion we actually want to recommend. This metric turned out to well represent the overall performance of the model and we think it can be helpful for other recommendation tasks based on a taxonomy.

With regard to the privacy-preserving machine learning our contribution is mainly in the field of federated learning.

We trained the recommender system in a federated environment with algorithms implemented by us. We give a comparison of the different algorithms and provide an example where FedAdam performs better than FedSGD and FedAVG, providing a model whose performance is similar to that obtained with a classic centralized training.

In addition to this, we propose a solution for the continual learning of the model with federated learning: it allows to repeatedly train the model without losing the knowledge about the previous training sessions. This is done without the need of collecting and storing the data about the previous trainings, in full accordance with federated learning: this solution is based on the idea of generating the data looking at the output of the recommender system. With this procedure we obtained comparable performance with that obtained with all the data available.

5 Dataset

5.1 Data Collection

The data we worked with consists in browsing histories of users: we will refer to these also as *click* histories, since they record the sequence of the pages the user clicked. This data, however, covers only a limited part of the total URLs browsed by the users: ID Ward Ltd, indeed, collects data only when a user visits a website where the company script is active. In addition to this, the collection of the data starts only if the user agrees the conditions that appear once they browse the page. These terms, if agreed, activate a tracking pixel that allows to register the user activity across multiple websites. Once the pixel is activated it starts to collect information about the browsing activity, such as the time spent on the page, the type of device used, the IP address and the content of the page.

Before April 2022, for a period of one year, ID Ward Ltd was active mainly on websites about video gaming and technology, thus the data collected was quite monothematic. Later on, the company started collecting data from other domains, including several news websites that have the benefit to contain information about several different categories, leading to a more heterogeneous dataset. In this work we will refer to these two collections of data as "first period" and "second period" dataset.

After collecting the data we divided both the datasets into train (0.5), validation (0.3) and test (0.2) set: we used the train set for the training of the models, the validation set for monitoring the loss function during the training while the test set was used for the final evaluation.

5.2 Data Categorization

After the collection of the data, the text of each URL is passed to a Natural Language Processing (NLP) model that aims at understanding to which category the site belongs to: it is a classification algorithm that has the purpose to label the content of the URL into a category of a taxonomy. This algorithm has an accuracy of 91%, bringing a considerable amount of noise in the data, since for the remaining 9% of the websites the predicted category could be totally different from the actual one.

The taxonomy used for this task is the IAB Taxonomy (see [9]), that is a well-known categorization for online advertising. The IAB enables the whole advertising ecosystem (advertisers, publishers, and platforms) to use a uniform and easy to understand vocabulary to describe content in order to make choices about ad relevance.

In this taxonomy each category is divided into four different levels of specificity (or *tiers*) where the deeper the level, the more detailed the category is. For example the category *Healthy Living/Wellness/Alternative Medicine/Herbs and Supplements* is a very specific category, since it has information about all the different levels. Anyway, the URLs can be categorized

	First period	Second period
Number of users	497599	190900
Total categories	304	433
Interests per user	2.57	2.95

Table 1: Information about the first and second period data.

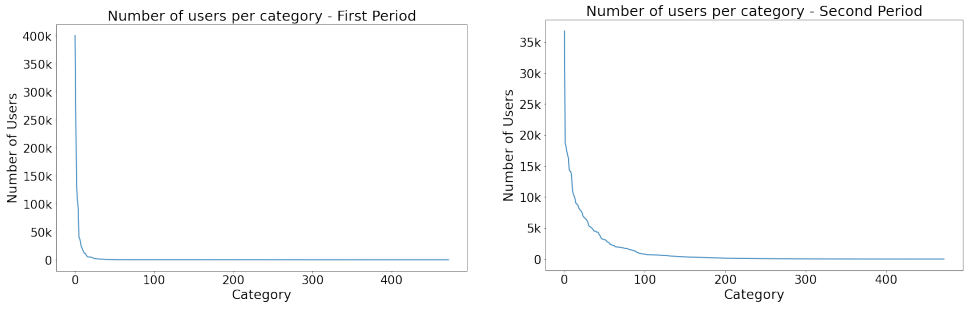
even with less tiers (e.g. *Pets/Dogs* or *Food & Drink/Food Movements*).

This original IAB taxonomy is composed by 533 distinct categories, divided into 27 main tiers: however, we neglected the sensitive categories in order to completely avoid discrimination and predatory targeting problems mentioned in Section 3.1. In particular, we did not considered categories about health and medical services, alcoholic beverages, smoking, political elections and gambling.

By doing this we obtain a taxonomy composed by 473 different categories, even if not all of them are present in the data used for this project: the first period data covers 304 categories, while the second period data covers 433 categories (see Table 1).

Furthermore, looking more closely at the categories that appear in our data, we noticed that only a small part of them is really significant for the users. In fact, there are numerous categories that are not interesting for the majority of the subjects even if they appear in the data. The restricted number of popular items can be seen in Figure 1a and 1b where we clearly see that, both in the first and second period, there is a small number of categories that is really interesting for the users. Furthermore, we underline that the tail of this distribution is larger for the second period (Figure 1b): this means that this dataset contains a larger number of relevant categories, as we expected considering the type of websites collected in the second period.

In addition to this, we show in Figure 2a and Figure 2b the histogram of the number of distinct interests per person in the two periods. Here,



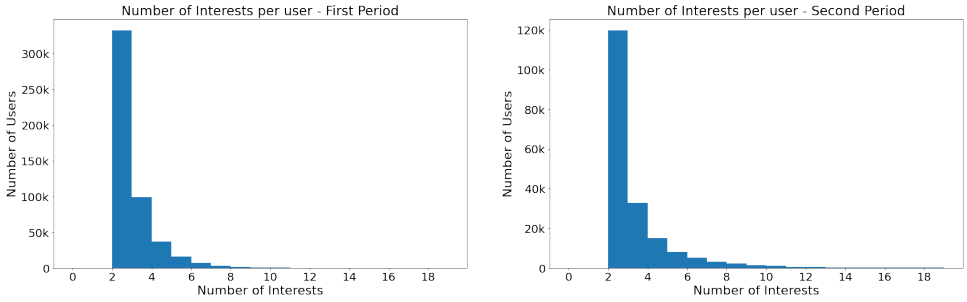
(a) (b)

Figure 1: Distribution of the categories in the two datasets.

Category (First period)	Users
Video Gaming / PC Games	400901
Video Gaming / Video Game Genres	252152
Hobbies & Interests / Games and Puzzles	134456
Technology & Computing / Computing	106602
Technology & Computing / Computing / Search	92601
Video Gaming / Video Game Genres / Action Video Games	39928
Hobbies & Interests / Games and Puzzles / Card Games	37957
Technology & Computing / Computing / Cloud Computing	30343
Hobbies & Interests / Games and Puzzles / Roleplaying Games	20683
Video Gaming / Video Game Genres / Simulation Video Games	16775
Hobbies & Interests / Radio Control	13647
Pets / Dogs	11517

Table 2: The most popular categories in the first period.

we considered only those users with minimum 2 different interests because this will be crucial for our training procedure (see Section 6.1). We noticed that each user has on average a small number of interests (2.57 for the first period and 2.95 for the second): this is clearly a limitation that makes the recommendation task quite hard because we need to suggest items starting only from a small number of categories. On the other hand this can allow us to catch more easily the correlations between the main categories, since a larger number of interests could also lead a large amount of noise.



(a) (b)

Figure 2: Distribution of the number of interests per user.

In Table 2 and 3 we report the most popular categories in the two dataset: from here we clearly see that the first period mainly includes data about games and technology. The most popular category of a different topic is *Pets/Dogs*, represented by only 11K users (against the 400K users interested in *Video Gaming/PC Games*). On the other hand, the second period includes more heterogeneous data, allowing to learn information about more categories.

Lastly, we point out that in order to have a fair evaluation we neglected

Category (Second period)	Users
News and Politics	36811
Music and Audio	18602
Real Estate / Real Estate Renting and Leasing	18150
Real Estate	17311
Style & Fashion / Street Style	16865
Real Estate / Houses	16172
Business and Finance	14411
Business and Finance / Business	14145
Family and Relationships	14101
Food & Drink / Dining Out	13345
Pop Culture	11118
Home & Garden / Home Improvement	10526

Table 3: The most popular categories in the second period.

the users of the validation and test set that have interests that do not appear in the train set: this is because a recommender system suggests only items that were already seen by at least one user.

5.3 Data Processing

After the data collection and categorization our aim is to process the data in order to create two different pieces of information:

- a *score* of interest for each category
- a vector - called *embedding* - for each website. It carries the information of the textual data (titles of the URLs or their entities).

These two types of data, described in detail in the next sections, will be the inputs of the recommender system.

5.3.1 Scores of interest

The score of interest is a value that measures the interest of a user into a particular category. This value is computed starting from the click history of the user and it depends on three different quantities:

- *time*, that refers to the total amount of time spent on pages of a particular category (in seconds)
- *recency*, that refers to the last time the user browsed a page of that category (in days)
- *frequency*, that refers to the number of times that the user has seen a URL of that category in the last month.

Among these three quantities, the time spent is the least significant since it could not reflect the real interest of the person. This is because it can happen that the device is active even if the user is not really interested in the page in that particular moment. Time, in fact, has been used only as a threshold: if the total amount of time spent on pages of a category falls under a particular value, the score of interest for that category is 0. Specifically, we took a minimum thresholds of 10 seconds: by doing this we discard all the categories that a user could have browsed accidentally, without a real interest.

On the other hand, recency and frequency are more relevant for our task. In order to create the scores, the values of these two quantities have been divided into 5 different levels of magnitude, using the thresholds showed in Table 4, obtaining a value between 0 and 5 for each of them.

Then, we considered as overall score the sum of the recency score and the frequency score, in such a way to obtain a value in the range $[0, 10]$. The formula for the computation of the score is shown in equation 1, where $s_u(c)$ is the score of user u for category c while r, f and t correspond to recency, frequency and time. $\mathbb{1}$ is the indicator function.

$$s_u(c) = (r_u(c) + f_u(c)) \cdot \mathbb{1}_{\{t_u(c)>10\}} \tag{1}$$

This way of computing the scores fits for our purpose because it includes a sort of decay for the interests: indeed, if the user stops to browse pages of a category, the interests falls to 0 (since the recency and frequency values gradually decrease).

5.3.2 Embeddings

Titles

With the aim to improve the starting information about the click history, we took advantage of the textual information of the title of each URL by creating vectors that carry that information. These vectors (called *embeddings*) are obtained through an NLP model that takes into account the semantical information contained in the text. This means that titles that share a similar meaning will be mapped in vectors closer than titles that are completely different.

This was done by exploiting two different architectures: at first we used the multilingual version of the Universal Sentence Encoder [8], a Convolutional Neural Network developed by Google in 2019 trained on sentences of

Score	0	1	2	3	4	5
Recency	> 30	[10, 30)	[6, 10)	[3, 6)	[2, 3)	< 2
Frequency	< 1	[1, 5)	[5, 10)	[10, 15)	[15, 20)	> 20

Table 4: The thresholds used for computing recency and frequency scores.

different length in 16 different languages, among which Italian. In addition to this, we tried another model, more specific for Italian, called GilBERTo [7], that is a pretrained language model based on BERT and trained on Italian text.

After having obtained the embeddings we created for each user a matrix of shape $(m, 512)$, where m is a parameter that corresponds to the maximum number of URLs to consider, while 512 is the embedding size. We did this with the aim to obtain an embedding history of fixed length to give as input to the RS. For all the users whose click history is shorter than m we padded the embedding matrix with zeros. We set parameter $m = 5$.

Entities

In addition to the titles we also leveraged another type of textual information, starting from the websites browsed by the users. In particular, we considered also the entities contained into the websites. This model was not developed in this project: we just focused on the creation of the word vectors, given the set of entities for each URL.

After the entity extraction we created the embeddings by means of a deep learning language model called *Word2Vec* [15]. It consists in a neural network that aims at learning a continuous representation of a word starting from its context. In our case the context of the word is represented by the set of words that appear with it in the same document (website).

A *Word2Vec* model can learn the word representation in two different ways, with a Continuous Bag-of-Words model (CBOW) or with the Continuous Skip-gram model, whose architectures are shown in Figure 3 and that work as follows:

- the CBOW aims at predicting a word starting from the surrounding context words: the one-hot encoding of the words in the same document are summed together and the resulting vector is taken as input of a linear layer, whose weights are the learning parameters of the model. We underline that this network does not take into account the order of the words in input, since the sum is permutation invariant.

- the Skip-gram, on the other hand, aims at reconstructing the context from a single word. Unlike the previous case, this model considers the order of the words in the document and gives more importance to the close words, since they are usually more related to the current word.

In particular, we decided to focus on the CBOW model, because the set of words for each URL is unordered.

For the training of the model we kept only the words that appear in at least 50 different websites, with the aim to remove the unpopular words that have only the side effect to add noise to the learning. As far as the size of the vector is concerned we fixed it to 100: decreasing the embedding size from 512

Coronavirus	Barcellona	Polizia
Omicron (0.94)	Bayern Monaco (0.99)	Il giudice (0.95)
Tribunale (0.92)	Bayern (0.98)	Tribunale di Taranto (0.95)
Comitato (0.91)	Real Madrid (0.98)	Polizia di Stato (0.95)
Arma (0.91)	Fenerbahce (0.97)	Vigili del Fuoco (0.95)
Francesco Acquaroli (0.91)	Real (0.97)	Mazzini (0.94)
Covid (0.91)	Manchester City (0.96)	Scienze (0.94)
Centro (0.90)	Villarreal (0.96)	In particolare (0.94)
Federica Nardi (0.90)	Reds (0.96)	spaccio (0.94)
Pfizer (0.90)	Liverpool (0.95)	Protezione civile (0.94)
Green Pass (0.89)	Ancelotti (0.95)	Giulio Silenzi (0.94)

Table 5: Examples of entities and their most similar words. The similarity measure is the cosine similarity.

(for the titles) to 100 allows to reduce the size of the recommender, providing a lighter model.

The representation of a URL is then obtained by averaging the vectors of all the entities contained in it, obtaining for each URL a single vector of size 100. Finally, we create for each user a matrix of size $(m, 100)$ that contains the embeddings of the last m websites browsed by the user, as we did with the titles.

After the training of the *Word2Vec* we took a look at the resulting model, checking if it was able to catch the similarities between the entities. For this task we used the *cosine-similarity* between two entities, defined as the dot product between the two vector representations. In Table 5 we reported some examples of words and their most similar entities.

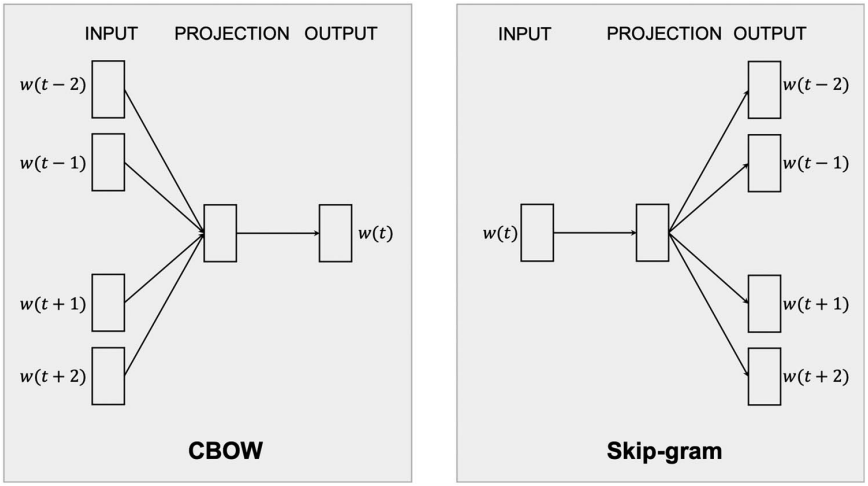


Figure 3: The Word2Vec model.

6 Method

In this Section we present the methodology used for training and validation of our recommender system. We start with the explanation of our specific training procedure and then we focus on the choice of the metrics for the evaluation.

6.1 Training Procedure

In our particular case the recommender system aims at suggesting new categories starting from the click history. So, in order to properly train and validate our model we put the recommender in an appropriate setting, building a training procedure that fits for this task.

We decided to recreate a realistic scenario, training the model in a situation similar to the application where the recommender will act. Specifically, we did this by hiding for each user a category at random among the ones they are interested in. Then, we try to reconstruct the full set of categories starting from the remaining ones.

At first, we identify which are the categories that are actually relevant for the users. These are not necessarily all the browsed categories, because it could be the case that a category was interesting only in the past or that the page was clicked accidentally. In order to do this we transformed the initial scores of interests into binary values, taking a threshold that is considered as the minimum level for considering a category interesting for the user. We set the threshold equal to 3.

After the binarization of the interests, at first we hide one category (we call this procedure *masking*) and then the resulting vector is given as input to the RS. The masking process occurs at random sampling from a uniform distribution among all the possible categories with score different from 0. The complete process is represented in Figure 4.

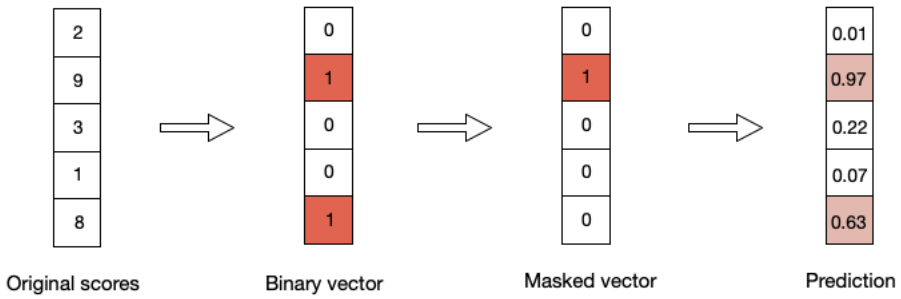


Figure 4: The adopted training procedure.

Moreover, the masking occurs at each new epoch in such a way that the hidden category possibly change at each iteration. This allows to exploit as much information as possible from the initial scores, creating from each user

as many train samples as the number of categories that could possibly be masked.

In order to recreate the same situation that will occur in the real application, after the sampling of the hidden category we hide also all the URLs that belong to the masked category. These URLs will not be taken into account in the embedding matrix: this is because we assume that those URLs will be seen in the future (as the masked category will).

6.2 Metrics

For the evaluation of the recommender system we used several metrics, considering both ranking metrics and threshold metrics. For the particular application of this RS it is more appropriate to look at how the model performs given a threshold and comparing the results by looking at the items that stay above this value (that will be the recommendations): this is because, depending on the quantity of possible clients that the publishers want to reach, they can adjust the threshold and obtain different values of confidence for the audience they send the advertise to.

In addition to this, we compared the performance also in terms of a ranking metric, HIT, with the aim of checking if the model behaves as we desire.

Moreover, we evaluated our models also considering how far are the top recommendations to the real one (the masked category) in the taxonomy: we did this because in a recommender system it could be the case that the suggestion is meaningful even if it is different from the item we want to suggest.

We underline here that all the metrics are computed considering only the set of items that have score equal to zero in the starting vector, since we expect that to high scores in the input are associated high scores in the output (interesting categories in the input will remain interesting also in the output).

6.2.1 Ranking metrics

Given the way in which we structured the training, it is worth considering where the masked category is positioned in the ranking of the predicted items. Ideally, we would like the masked category to be always the category with the highest score in output, among the ones that are not observed in input. At the same time, however, we are not interested in the ranking of all those categories that are not relevant for the user, so for this reason we decided not to focus on metrics that consider the whole ranking, such as the NDCG [24].

In particular, the ranking metric that we chose is the HIT@k: for each sample it corresponds to 1 if the hidden category is positioned within the top k categories, while it is 0 otherwise, as follows:

$$HIT@k = \begin{cases} 1 & \text{if } c \in \text{TOP-k} \\ 0 & \text{otherwise,} \end{cases}$$

where c is the hidden category. By formulating the HIT in this way, its mean over all the users corresponds to the average number of times in which the masked category is within the top k items.

6.2.2 Threshold metrics

As far as the threshold metrics is concerned, we focused on the *Precision* and *Recall*. Once a threshold is fixed, the Precision corresponds to the fraction of suggested items that are actually relevant (i.e. interesting) for the user, where an item is *suggested* if its output score is higher than the threshold. On the other hand, the Recall is the fraction of relevant items that are suggested.

Reasoning in terms of positive and negative predictions (where the positive class is the class of items that we want to suggest), Precision and Recall are defined as follows:

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN},$$

where TP are the true positives, FP are the false positives and FN are the false negatives.

6.2.3 Taxonomy-based metric

Finally, we wanted to consider not only if a suggestion is right or wrong, but also how far the prediction is from the true suggestion (the masked category). In order to achieve this, we decided to focus also on a "taxonomy aware" metric, namely a metric that considers the different levels (tiers) of the taxonomy. Indeed, in the context of recommender systems, it could be meaningful to check if a suggestion lies in a field that is conceptually similar to the one we want to predict, even if the prediction itself is wrong.

So, assuming that the parent-child relation in the taxonomy tree is the closest possible, we formulated a distance between two categories as the number of steps in the path that connects the two categories in the taxonomy tree. We underline that the taxonomy graph is actually a tree, since all the categories have distinct sub-categories; so, it exists only one path that connects two nodes.

Figure 5, for instance, represents the taxonomy tree for the main tier *Technology and Computing*: in this case the distance will be equal to 2 for categories that are similar or very related, such as *Artificial Intelligence* and *Robotics*. The path [Artificial Intelligence, Technology and Computing, Robotics], indeed, is the path that connects the two categories and it has length equal to 2. For categories that are not so similar, instead, the same distance will be higher: for example its value for the categories *Artificial Intelligence* and *Web Design and HTML* will be equal to 3 since the path [Artificial Intelligence, Technology and Computing, Computing, Web Design and HTML] is the path that connects the two.

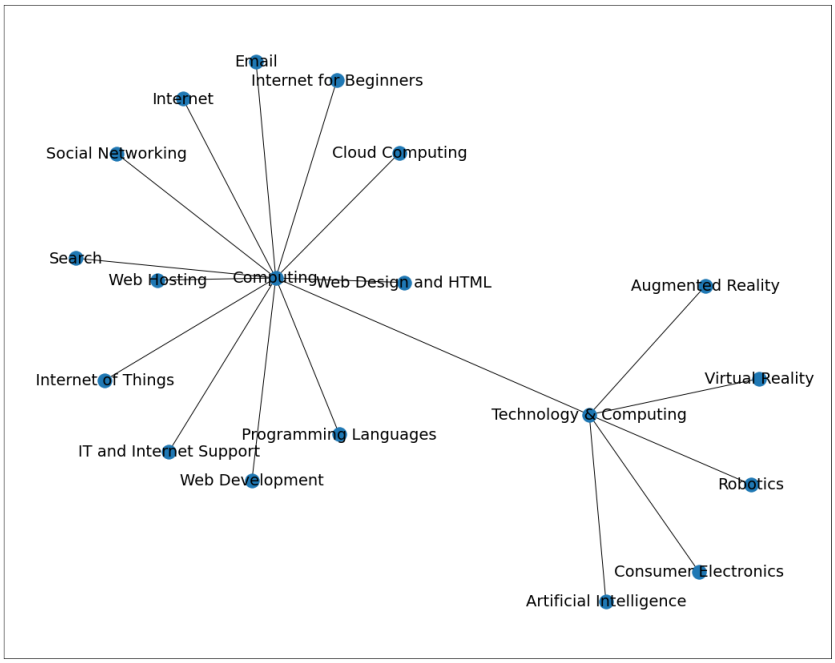


Figure 5: The taxonomy tree for the field of *Technology and Computing*.

We remind that since the taxonomy tree is 4 levels deep, the maximum distance between two categories is equal to 8.

Finally, as metric we considered the average distance between the right suggestion (the hidden category) and all the categories that were placed higher than it in the predictions, in terms of score. This means that the lower this metric is, the better the recommender works. If the right suggestion is exactly the top-1 suggestion, then the metric will be equal to 0.

For instance, if the category we want to suggest is *Artificial intelligence*, but the recommender places it in third position after *Web Design and HTML* and *Robotics*, then the taxonomy metric will be equal to 2.5.

7 The model

In this Section we focus on the architecture used for the recommender system: we explain the experiments done with the type of architecture and with its hyperparameters, with the purpose to find the model that performs the best for our task. We start by presenting a model that consists in a non-personalized RS: we consider this recommender as a baseline for the following models. Then, we present our deep learning recommender system, comparing the results obtained with different choices of the starting features. Furthermore, we propose an approach for the refinement of the model on new data. Finally, we show our implementation of the pipeline, from the data processing to the creation of the look-alikes.

7.1 Popularity-based model

As a first straight forward approach we tried to tackle the recommendation problem by suggesting items only based on the overall popularity of the categories. It basically uses the items that are in trend right now and suggests the ones that are most popular among those that have not already been seen by the user. For *popularity* we mean the number of times that a particular category appears in the train set.

Clearly this model is rough because neither it provides personalized recommendations nor it is able to catch the correlations between the categories. Moreover, this algorithm favors those items that are already relevant, because the unpopular items will never be suggested. However, there are situations where this type of recommender outperforms several state-of-the-art models, such as for those problems where there are common traits between all the subjects, regardless of the features of the users and the items.

This is the case of our data regarding the first period: this data contains mainly information about video-games and technology and therefore an approach based on popularity leads to good results. Indeed, the popularity model in this case gave a value of HIT@1 of 52%. This means that 52% of the times we are able to suggest the right category to the user: we are able to reach this remarkable value because all the user have common interests.

Looking at how the popularity model performs considering all the available data (from both the first and second period) we see that the value of the HIT@1 decreases until 37% as expected, since there is a broader set of relevant categories. Despite this, the obtained value is still very large, denoting that there is a large disproportion in the data.

Now we restrict our analysis on the second part of data, with the purpose of checking how this model performs on data collected from heterogeneous websites: this guarantees that a large number of categories is well represented and that there is no a predominant category. In this case the value of HIT@1 returned by the popularity model is only 7%: from this we conclude that the popularity model performs well only for monothematic datasets.

Dataset	HIT@1
1^{st} period	52%
1^{st} and 2^{nd}	37%
2^{nd} period	7%

Table 6: HIT@1 values for the popularity model.

Besides, this model has the limitation that the number of suggestions has to be fixed a priori since there is not a value of confidence for the predictions, and so it does not make any sense to talk about Precision and Recall: for this reason it does not fit for our segment creation task, where the companies need an appreciation value for each category.

7.2 Autoencoder-based model

After the first trial with a popularity-based model, we decided to tackle the recommendation problem with an autoencoder-based architecture. In order to do this we started from the idea of the Variational Autoencoder (VAE), that is one of the state-of-the-art approaches for recommendations. In particular we kept its core autoencoder architecture because it fits well with our task and our data structure.

The main difference between an autoencoder and a variational autoencoder is that a VAE has the objective to learn the distribution from which the input vector is sampled from, while a simple AE only aims at learning a compact and deterministic representation (the *encoding*) of the input. In terms of architecture, the difference is in the layer before the encoding, where the VAE aims at learning the mean and the variance of the input distribution (it is not a goal of this work to focus on the VAE architecture: for more details see [12]).

First of all, considering a simple AE instead of a VAE allows to slightly reduce the size of the model: our recommender system, in fact, needs to be particularly light because it will be used in the user's device to provide suggestions without accessing their data. In addition to this, it has to be light because the training with federated learning will take place in the device, for privacy-preserving reasons (see Section 8). A light model, indeed, allows to save battery and memory in the user's device, along with egress costs which relate to sending the model back and forth from client to server. Moreover, it also allows to speed up the training process on the user's device and to save money due to the storing cost of the model in the server. For these reasons we decided to focus exclusively on models whose memory requirements did not surpass 1.5MB.

Furthermore, the choice of a simple autoencoder over a VAE can be explained due to the fact that a classic AE allows us to easily add other input information about the users' browsing histories. In our particular case we

decided to consider the information about the titles and the entities of the pages browsed by the users. We did this by vectorizing those information into *embeddings*, obtained through deep learning models. We called this model *PseudoAE* (PAE) because it has the structure of an autoencoder, but with added information in input. Other than that, we trained also a normal AE without any additional information, with the aim of understanding the actual contribution of the titles and the entities for the final recommendation.

7.2.1 The architecture

The architecture of the PseudoAE is composed by three main networks:

- the *Encoder*
- the *Embedder*
- the *Decoder*.

The *Encoder* is the network that, given the vector of the interests, aims at providing a compact vector representation of the user. It takes as input a binary vector of size 473, that corresponds to the number of interests in the taxonomy. It is composed by 2 fully connected layers followed by a batch normalization layer, as shown in Figure 7a.

The *Embedder* is the network that aims at processing the textual information of the titles: it takes as input a matrix of size (m, emb_size), where m is the maximum length of the browsing history, and emb_size is the embedding size (512 for the titles and 100 for the entities). By doing this the matrix contains the information of the last m pages browsed by the user. It is composed by a RNN and returns its final state, in such a way that this vector is representative for the whole click history of the user.

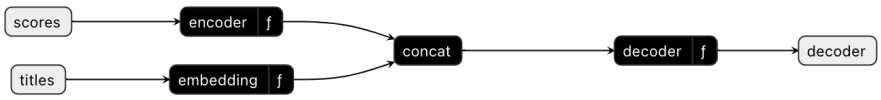
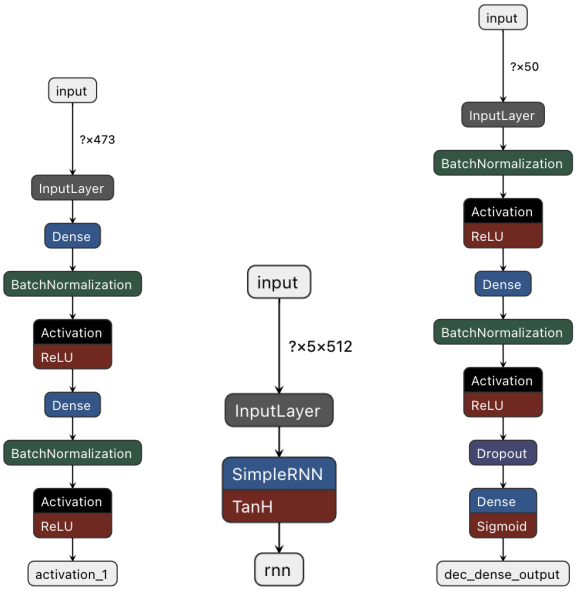


Figure 6: The overall structure of the PAE.

The *Decoder*, finally, is the part of the architecture that aims at providing the reconstruction of the scores (and so the recommendations) given the outputs of the Encoder and of the Embedder, as shown in Figure 6. It is composed by 2 fully connected layers, 2 batch normalization layers and a dropout layer. It returns a vector of size 473, whose entries are in the range $[0, 1]$ since they are the output of a sigmoid function, defined as $\sigma(x) = \frac{e^x}{e^x+1}$: in this way the i-th output of the model can be seen as the probability that the user is interested in the i-th category. The overall architecture of the three components of the PAE is represented in Figure 7.

Furthermore, we noticed that the network returns high values for those interests that appear in input, as expected, but it gives low values for the



(a) Encoder (b) Embedder (c) Decoder

Figure 7: The architecture of the *PseudoAE*.

interests we want to suggest, even when they are among the top-rated. This creates a significant gap between the items already present in the click history and the new ones, even when the suggestions are meaningful: this is a problem because in the creation of the segments the companies need to set a threshold that is a sort of probability, and we want this value to be reasonable. This is not what happens in our case because, for instance, a value relatively low, as 0.3, is actually very high for our model. This creates a relevant mismatch between the need of the company and the probabilities provided by our model.

In order to solve this problem, we applied in the inference process a concave monotone function after the application of the sigmoid. More specifically, we decided to adopt the equation of a circumference passing through the origin and the point $(1, 1)$, since we want that a score of 0 or 1 remain unchanged, and whose center lies in the straight line of equation $y = -x + 1$. The equation is the following:

$$f_a(x) = \frac{a+2}{2} + \frac{1}{2}\sqrt{(a+2)^2 - 4(x^2 + ax)}$$

Adjusting the value of a it is possible to control the slope of the curve, as show in Figure 8: for our purpose we decided to adopt $a = -2.5$.

7.2.2 Training

As loss function we used the Binary Cross-Entropy, defined as follows:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)), \tag{2}$$

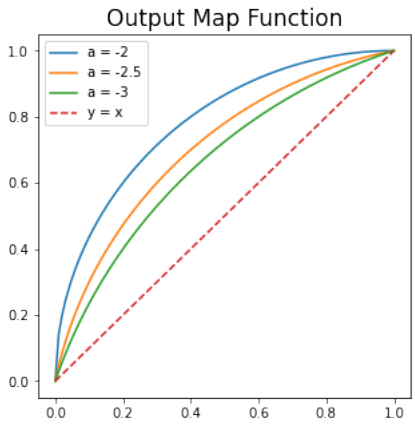


Figure 8: The plot of the function $f_a(x)$ applied in the inference part.

where y is the actual vector of interests of a user, while $\hat{y}$ is the output of the model. It corresponds to the vector of the predicted probabilities obtained from the partial vector of interests (with one hidden category, as explained in Section 6.1). N, instead, corresponds to the total number of categories.

As optimizer we used Adam and the Stochastic Gradient Descent (SGD): the results we obtained are comparable, so in what follows we report only the results found with Adam.

As far as the number of training epochs is concerned, looking at the value of the loss function on the validation data we noticed that a number of epochs equal to 2 was enough in order for the network to converge.

7.3 Results

In this Section we report the experiments we have done for the tuning of the architecture and of the features: for the type of architecture, we focused on the simple autoencoder and on the *PseudoAE* described in Section 7.2.1. Regarding the choice of the features, we compared the results obtained by taking into account the scores of interests, the titles of the URLs and the representation obtained from the entities.

Moreover, we compared the results obtained from two different datasets with the purpose to see how the recommender system performs when the distribution of the interests changes. All the results of this Section refer to the test set.

First period data

For these experiments we worked on the data related to the first period of time, containing mainly information about video-games and technology (see Section 5). All the experiments with this data are reported in Table 7. The values of Precision and Recall refer to a threshold of 0.6.

AutoEncoder									
Param	Loss	RNN	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
52K	0.0069	-	-	20	0.50	0.52	0.52	0.82	2.16
73K	0.0065	-	-	30	0.51	0.53	0.52	0.82	2.13
83K	0.0062	-	-	40	0.51	0.53	0.52	0.83	2.14
105K	0.0061	-	-	50	0.50	0.52	0.52	0.82	2.16
PAE with Titles									
Param	Loss	RNN	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
63K	0.0063	SimpleRNN	20	30	0.54	0.52	0.53	0.84	2.07
73K	0.0068	LSTM	20	10	0.58	0.55	0.58	0.85	1.85
95K	0.0059	LSTM	20	30	**0.59**	0.58	0.58	**0.86**	1.83
107K	0.0057	LSTM	20	40	**0.59**	**0.59**	**0.59**	**0.86**	**1.80**
PAE with Entities									
Param	Loss	RNN	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
51K	0.0065	LSTM	10	30	0.56	0.53	0.55	0.84	2.01
54K	0.0063	GRU	20	25	0.53	0.55	0.54	0.84	2.02
74K	0.0061	LSTM	20	40	0.55	0.55	0.55	0.84	1.99
110K	0.0057	LSTM	20	70	0.56	0.55	0.55	0.85	1.98

Table 7: The results obtained with the first period data.

We start analyzing the results obtained with the simple autoencoder: from Table 7 we see that the values of Precision and Recall are about 0.52 for both the metrics. The value of the HIT@1 is 0.52 as well and the values of the HIT@4 lie around 0.82. The taxonomy metric, instead, measures on average 2.15.

Looking at the results obtained also from the information of the titles of the URLs, we noticed that all the metrics improve. The best model found in this setting has values of Recall, Precision and HIT@1 of 0.59, consisting of an improvement of 7% compared to the simple autoencoder. Furthermore, looking at the value of the taxonomy metric we see that it is 1.80: this means, first, that the model is able to better predict the right recommendation. Secondly, when the recommendation differs from the actual interest of the user it gives recommendations that are closer to the actual one, in terms of the taxonomy. Given these values, we state that the textual information of the titles positively affects the model, leading to better recommendations.

In addition to this, we have done experiments considering also other information from the websites: in this case we created the URLs embeddings starting from the entities present in the website. In particular we computed the vector representation of a URLs by considering the mean between all the vectors of the entities contained in the website. We noticed that even this attempt improves the quality of the recommendations: from the Table 7 we see that the values of the metrics slightly improves, even if they seem to be

worse than the results found by using the titles.

We remind here that the vocabulary used consists of 2000 words, obtained using all the entities appearing in at least 50 different websites: we think that the recommender system could further improve by collecting information about more websites thus enriching the vocabulary of the Word2Vec.

We point out that the taxonomy metrics we adopted represents well the overall performance of the models: looking at the Table 7 we noticed that the models whose value of the taxonomy metric lie below 2.00 have evident better values of Recall, Precision an HIT. Furthermore, we highlight that all the models that use an additional textual information have a lower taxonomy metric value than the values returned by the simple autoencoder.

As last remark about these experiments, we recall that the value of HIT@1 for the popularity model trained on this data was 52%: this means that the only models that are able to exceed this quantity are those that leverage the entities or the titles. In any case, also the simple autoencoder provides an improvement over the popularity model since it returns values of confidence for the interests and not only a binary value (suggest or not suggest).

Second period data

Running the same experiments on the data about the second period we noticed that our deep learning RS clearly outperforms the popularity model: in this case, indeed, it provides a value of HIT@1 of 7%, while our recommender system gave values of HIT@1 of over 30%. This denotes a clear advantage of our model over the recommender base on popularity. Moreover, this shows that in our case the popularity model performs well only with monothematic data, and this will not be the situation our recommender will operate in.

The results obtained with this data are reported in Table 8: from there we see that even in this case the models that exploit additional information perform better than the simple autoencoder, and, again, we noticed that the PAE with the titles with a LSTM recurrent neural network is the best model, having a value of the taxonomy metric of 2.86.

We underline that also in this case the taxonomy metric represents well the overall performance of the model.

7.4 Refinement of the model

As already pointed out, the data related to the first period regards mainly websites concerning gaming and technology, leading to highly monothematic data. In this way the model catches only the peculiarities and correlations between a restricted number of categories, while there is a large number of categories that is not represented by the model. The category *News and Politics*, for instance, is not well represented in the first data (it appears in only 8 different browsing histories) and the model is not able to provide meaningful recommendations, as shown in Figure 9. Here are reported the highest output probabilities for a user that is only interested in *News and*

AutoEncoder									
Param	Loss	RNN	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
20K	0.0155	-	-	10	0.28	0.33	0.31	0.53	3.17
40K	0.0117	-	-	20	0.24	0.41	0.31	0.55	3.13
62K	0.0105	-	-	30	0.23	0.43	0.32	0.55	3.13
83K	0.0098	-	-	40	0.22	**0.46**	0.32	0.55	3.10
PAE with Titles									
Param	Loss	RNN	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
68K	0.0105	LSTM	10	30	0.30	0.39	0.36	0.60	2.92
75K	0.0100	SimpleRNN	20	40	0.21	0.42	0.31	0.54	3.15
95K	0.0100	LSTM	20	30	**0.31**	0.41	**0.37**	**0.61**	2.89
107K	0.0096	LSTM	20	40	0.29	**0.46**	**0.37**	**0.61**	**2.86**
PAE with Entities									
Param	Loss	RNN	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
51K	0.0109	LSTM	10	30	0.27	0.40	0.33	0.57	3.05
62K	0.0104	LSTM	20	30	0.26	0.41	0.33	0.57	3.05
63K	0.0103	LSTM	10	40	0.26	0.42	0.34	0.57	3.01
74K	0.0099	LSTM	20	40	0.26	**0.46**	0.34	0.58	3.03

Table 8: The experiments done with the data of the second period.

Politics: we noticed that they correspond to the categories most represented in the first period data.

In order to solve this problem we try to integrate the new categories into the recommender system. With this purpose, for each architecture we consider the best model found in the previous Section and we retrain it on the data of the new period, starting from the weights obtained after the first training. The best models were chosen by selecting a balance between model size and performance. While retraining the recommenders we noticed that the metrics decrease, as expected, since now the train set contains a larger number of relevant categories.

In Table 10 we reported the results obtained in this case: here, again, we denote an advantage of the models that leverage the titles and the entities. Even in this case, the taxonomy metric chosen turned out to be representative for the overall performance of the model.

Looking now at the recommendations for a person interested in *News and Politics* we see (Figure 11) that the model was able to learn the correlation between the new categories. From here we can also observe that the input categories have high scores in the output: this is essential for the creation of the segments with the look-alikes modeling because we want the segment to contain both users that are likely to be interested into a category and users that we already know are interested into it.

On the other hand, comparing the results obtained in Table 10 with those

News and Politics	
Video Gaming / PC Games	0.99
Video Gaming / Video Game Genres	0.81
Technology & Computing / Computing / Search	0.65
Hobbies & Interests / Games and Puzzles	0.53
Technology & Computing / Computing	0.42
Technology & Computing / Computing / Cloud Computing	0.29
Technology & Computing / Computing / Internet	0.08
Video Gaming / Video Game Genres / Simulation Video Games	0.08
Hobbies & Interests / Games and Puzzles / Card Games	0.07
Video Gaming / Video Game Genres / Role-Playing Video Games	0.06

Table 9: The recommendation provided to a user interested in *News and Politics* before the new training of the model.

Type	Param	Loss	Units	Enc Size	Rec	Prec	HIT@1	HIT@4	Tax
AE	73K	0.0120	-	30	0.22	**0.43**	0.32	0.55	3.12
PAE-T	73K	0.0150	20	10	**0.32**	0.32	**0.34**	**0.57**	**3.03**
PAE-E	74K	0.0126	20	40	0.26	0.41	0.33	**0.57**	3.04

Table 10: The model obtained by re-training a pre-trained model on a new set of data.

obtained using only the second part of data (Table 8) we noticed that they are quite similar: this is because with the new training the network tends to forget the knowledge about the other categories. Looking at the suggestions given to a *PC Games* person for example, in Figure 12, we see that now they are completely meaningless.

In order to solve this problem we simply propose to merge the new train set with a part of the old train set, in such a way that information about the old data is kept. We used a proportion 1:4 between the old and the new distribution, in such a way that the greatest contribution for the learning is given by the new data, while the old data is only necessary for preserving the old knowledge.

Looking at the recommendations provided after the new re-training of the model we noticed that the model is now able to provide proper suggestions for both the users, and this denotes that the new training correctly integrated the knowledge of the new categories into the model. The recommendations given to a user interested in *News and Politics* are comparable to those obtained in the previous case (Table 11) and those suggested to a *PC Games* person are now reasonable (see Table 13).

Other examples of recommendation are reported in the Appendix A: these suggestions are obtained with the model that exploits the titles, but we considered no titles here for sake of simplicity.

News and Politics	
News and Politics	0.99
Business and Finance / Industries / Power and Energy Industry	0.33
Style & Fashion / Street Style	0.30
Business and Finance / Industries	0.28
News and Politics / Politics	0.27
News and Politics / Politics / War and Conflicts	0.25
Business and Finance / Business	0.24
News and Politics / Crime	0.23
Business and Finance / Business / Business Banking & Finance	0.22
Style & Fashion / Designer Clothing	0.21

Table 11: The recommendation provided to a user interested in *News and Politics* after the new training of the model.

Video Gaming / PC Games	
Business and Finance	0.33
News and Politics	0.31
Business and Finance / Business	0.15
Business and Finance / Business / Business Banking & Finance	0.11
Family and Relationships	0.10
Healthy Living	0.08
Food & Drink / Desserts and Baking	0.08
Style & Fashion / Street Style	0.07
Real Estate	0.05
Real Estate / Retail Property	0.05

Table 12: The recommendation provided to a user interested in *PC Games* after the new training of the model.

Video Gaming / PC Games	
Video Gaming / PC Games	0.83
Video Gaming / Video Game Genres	0.69
Hobbies & Interests / Games and Puzzles	0.55
Technology & Computing / Computing / Search	0.51
Technology & Computing / Computing	0.49
Technology & Computing / Computing / Cloud Computing	0.31
Pets	0.23
Music and Audio	0.21
Hobbies & Interests / Collecting	0.20

Table 13: The recommendation provided to a user interested in *Video Gaming* after the refinement of the model using a merged train set.

Even if this training procedure is the best we found for the refinement of the model, it brings some drawbacks. First of all, every training session requires to store the old data and this could make the training quite expensive, after several updates. Secondly, the more data is used in the refinement process, the less is the ability of the model to generalize, since the proportion of new data compared to the entire data will be lower and lower as the time goes by. A solution to both these issues could be to restrict the training only to a fixed temporal window: indeed, it becomes unnecessary to keep the information about how categories related in the distant past, since the correlation between the categories can possibly change over time and we are interested only at how they related in the recent past.

In a federated environment, however, we do not have the possibility to access and store the data. Therefore, we will need to find a way of integrating new categories without their direct collection: this led to our proposal for the continual learning of the model with federated learning, in Section 8.6.

7.5 Model Pipeline

The entire process for the look-alikes modeling, from the query of the data to the final creation of the look-alikes, has been implemented using Apache Beam. It is an open source unified programming model that allows to define both batch and streaming data-parallel processing pipelines. It allows to execute pipelines on distributed processing backends, including Apache Flink, Apache Spark and Google Cloud Dataflow.

We used precisely Google Cloud Dataflow for this purpose, because it is a fully managed service that works within the Google Cloud Platform (GCP) ecosystem, and this is helpful since the data we work with is stored in GCP buckets. The data is then accessed through BigQuery, a data warehouse that allows to easily manage data in GCP with SQL queries.

The overall pipeline is composed by two processes, namely the pipeline involved with the processing of the data (scores and titles) and the pipeline that

is responsible for creating and saving the look-alikes. The entire pipeline is based on the model of Section 7.3 that use the titles as additional information, but an analogue process is done also for the entities.

An Apache Beam pipeline is designed as a Directed Acyclic Graph (DAG) of components and each of them has a specific task within the pipeline. Its execution is conditioned with the successful finish of its predecessor components, as shown in Figures 9 and 10, that represent the DAGs of the two pipelines.

7.5.1 Process Pipeline

This pipeline runs daily and it aims at processing the raw data, creating the scores and title embeddings described in Section 5.

When the pipeline is triggered three parallel processes start, corresponding to the initial components *Clickstream*, *Old scores* and *New Titles* in Figure 9.

The first branch aims at computing the interests of the users that have browsed at least one page in the last day: indeed, it is not necessary to compute every time the scores of all the users, since we assume that if they do not browse any new website their interests do not change.

In the meantime, the pipeline loads the interests of the remaining users through a BigQuery query and these two branches are merged together with the aim to create a unique table with the updated scores. The table is then saved in BigQuery for the computation of the look-alikes (second pipeline).

The third branch of this process computes the embeddings of the titles that have never been computed before and saves them in a separate table: this part of the pipeline is quite time consuming since it uses a large external NLP model.

The entire process has been optimized and the distributed computation has been possible through the use of the Beam ParDo functions, that allow the components to run on multiple machines.

The main bottleneck of the process was the computation of the embeddings: storing them gradually day by day and running the pipeline daily allowed us to minimize the waiting time of this component.

The process of computing the embeddings of all the titles from scratch (300K websites, collected in over one year) took around 3 hours; this time corresponds to a distributed time of 20 minutes on 10 different machines.

On the other hand, in Figure 9 we show that by computing only the new embeddings (about the last day) the process becomes much faster; the *New Titles* component took only 21 seconds.

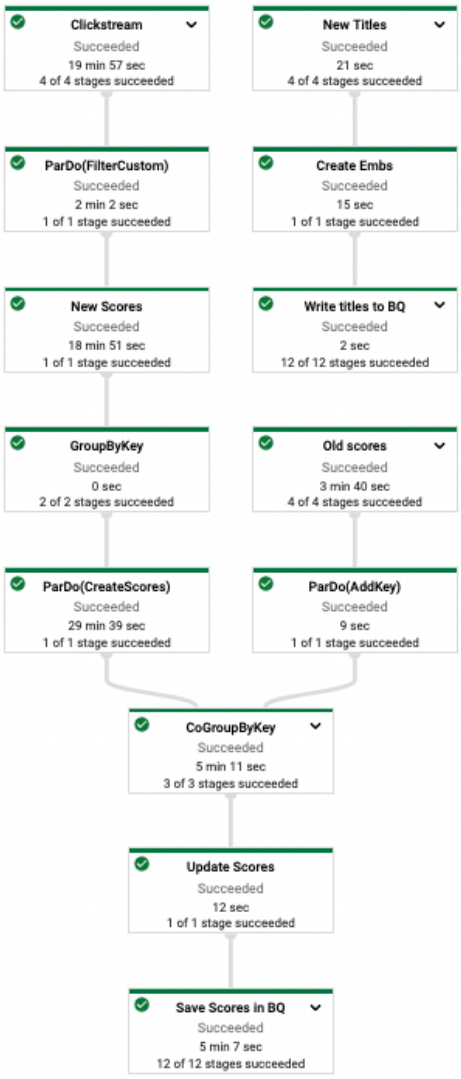


Figure 9: The process pipeline.

7.5.2 Look-alikes Pipeline

The pipeline for the creation of the look-alikes runs as soon as the first pipeline ends. Once it is triggered, two different processes start: they correspond to the components *Load data* and *Old Look Alikes*, in Figure 10.

The *Load data* branch takes as input the data processed in the previous pipeline: first, the proper matrices are created, then, they are given as input to the recommender system. By doing this we create the look-alikes of only those users that have browsed websites in the last day. The obtained output is then merged with the old look-alikes creating the updated look-alikes table.

This pipeline is more expensive than the previous one, and the main time consuming activity is the loading of the data: the query done in this passage

merges data from different tables, leading to a huge query. Anyway, we point out that the execution times of Figure 10 are relative to the computation of the look-alikes of all the users, while it is enough to run this pipeline only on those users that browsed a new site in the last day (this allows to lighten the computation).

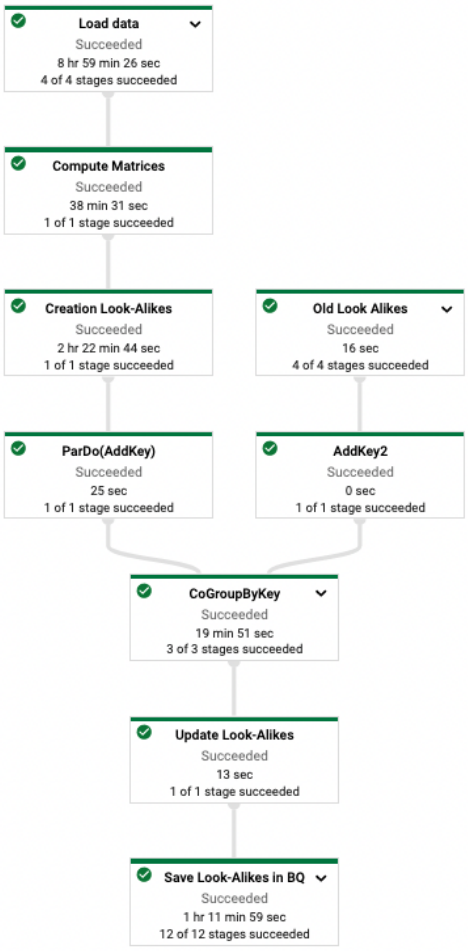


Figure 10: The look-alikes creation pipeline.

8 Federated Learning

The direct collection of data in a central database and the training of data-driven models in a standard centralized way is nowadays getting more and more complicated due to several reasons.

Firstly, it suffers from storage problems because the data collected needs to be stored outside the user's device in a central system. This becomes a problem if the number of users is very large. For preventing this issue we would like the data to not leave the user's device and to train the models without collecting it.

The second reason, and one of the main aspect tackled in this work, is that the direct collection of data could lead to serious privacy risks. The unsolicited transmission of personal data for profits from the company that directly collects it to third parties is one of these issues. By accepting the conditions showed once a page is browsed, the users have to trust that the company will not share their data with other companies. This problem can be overcome by "zero-trust" companies, namely companies that provide a personalized product to the user without the need to know their data: by doing this the users do not need to trust that the company will sell their data, simply because the company does not have it.

Even if the GDPR regulates sharing and selling of data to third parties, a lot of companies violate it. Indeed, it is hard to verify that the requirements of the GDPR are met, and usually the amount of the fine they have to pay for this violation is still less than the profit they gain from the sell.

These reasons have brought the birth of data privacy preserving techniques that allow to train a ML model without the need to access the data of the user: the resulting systems are known as privacy-preserving machine learning systems (PPML). Federated Learning (FL) is one of these solutions.

Federated learning is an emerging technology for training machine learning models that aims to solve the privacy problems caused by transmission of personal data. FL provides an alternative way of learning from data, different from the standard centralized way. In a federated environment the learning takes place on multiple local datasets stored at isolated data sources, such as smartphones, tablets and PCs, that correspond to the clients' devices.

A FL setting is composed by different roles, that are mainly:

- the *clients*, namely all the possible users that could take part the learning
- the *server*, namely the central system that coordinates the learning of the overall model: it repeatedly receives the new models trained on local datasets and creates a new overall model.

A FL algorithm is usually composed by different *rounds*: at each round the central model (stored in the central system) is broadcast to a set of selected clients, chosen randomly. Then, the training occurs in every client's

device and, after that, each client shares with the central system the new parameters of the model. All the models are then combined to obtain a new model. This process is repeated a multiple number of times, until the overall model converges. The overall process is shown is Figure 11.

The main advantage of this type of algorithm is that the model is trained without the need for collecting and processing the personal data, because the only information that is shared with the central system is the one regarding the new weights of the model trained on the user's device.

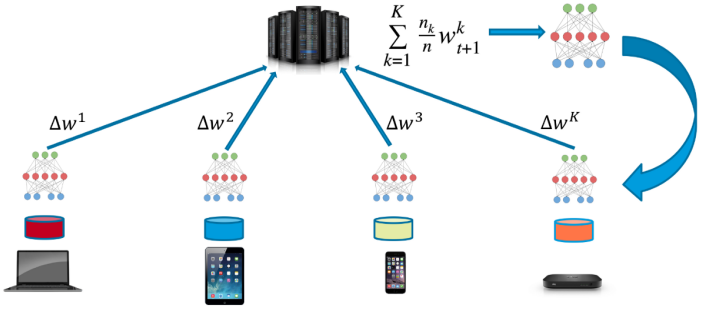


Figure 11: The federated process: the server model is broadcast to each selected client. Then, after the training on their local devices, they share with the central system only the updated model.

8.1 Application for Recommender Systems

While there are some ML algorithms that need to be trained only once to perform a certain task (such as general classification tasks where the distribution of the data does not change over time) there are data-driven algorithms that need to be trained repeatedly. This is because the distribution of the data can possibly change: they need to adapt to it and perform well even on data totally different from those available at the moment of the first training.

This is the case of NLP algorithms such as "Next Word Prediction" algorithms, where the task is to predict the next word a user will type in a keyboard and to suggest to them the most probable. This type of models need to be personalized since the set of common words can radically change even within the same country, depending on factors such as age, habits and personality. Besides, their behaviors - as well as their language - can slightly change over time due to new interests or other changes caused by the growth of the subject.

Even Recommender Systems deal with data that is not static, it changes over time, and this happens very frequently: in the case of product recommedation, for instance, the necessity of purchase of a particular item usually rises suddenly from external reasons that cannot be predicted by the RS. Moreover, a RS needs to suggest also new items that have just been released

and it has to adapt its recommendation based on the new trend: this could not be done based exclusively on the first data used for training the model.

In the best possible scenario we would like to frequently collect new data of the users and repeatedly use it for training the model with a standard centralized learning. Unfortunately, this is not possible due to storage and privacy reasons pointed out at the beginning of this Section.

Here comes into play the FL: starting from the data collected in a central dataset we want to run simulations in a federated setting with the aim to train the model on local datasets. We want to compare some federated algorithms checking which returns the best model and, furthermore, we want to see if the model obtained with FL gives a performance comparable to that obtained with a centralized training.

Moreover, as far as the application for ID Ward is concerned, we aim to find a way to train periodically the model with FL keeping the knowledge acquired in the previous training sessions: with this purpose we propose a solution for the Continual Learning of the model with federated learning (Section 8.6).

8.2 Problem Formulation

In terms of the mathematical formulation, the optimization problem we want to solve with Federated Learning is a minimization problem over a loss function $f(w)$, where $w \in \mathbb{R}^d$ are the weights of the model that we want to learn. Typically $f(w)$ is defined as

$$f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w), \tag{3}$$

where n is the number of training samples and $f_i(w)$ is the value of the loss function for the i-th sample. It is usually defined as $f_i(w) = l(x_i, y_i; w)$, that is the loss on the example (x_i, y_i) made with model parameters w, where x_i is the input, y_i is the target of the i-th sample and l a loss function (in our case the Binary Cross Entropy). We suppose that there are C different clients over which the data is partitioned, and for each client there are $n_c = |P_c|$ different samples, where P_c is the set of the indices of the samples belonging to the c-th client.

By treating the problem in this way we can rewrite the loss function as

$$f(w) = \sum_{c=1}^{C} \frac{n_c}{n} F_c(w), \text{ where } F_c(x) = \frac{1}{n_c} \sum_{i \in P_c} f_i(w). \tag{4}$$

In this setting we assume that the loss function $f(w)$ cannot be directly computed because we have access only to a small portion of clients. This happens because of complex circumstances beyond the control of the orchestrating server that limit the possibility of including some clients into this

process. For example, there could be some limitations due to internet connections problems or low-battery issues. For these reasons we assume that only a limited number of clients can be reached. In our experiments we simulated this problem by selecting only a random subset of clients.

As a result, we want to optimize the loss function with methods that aim to approximate $f(w)$ or $\nabla f(w)$ given only a partial participation of the clients. Usually the optimization problem (3) is solved through gradient descent (GD) techniques, namely optimization iterations of the form $w^{(t+1)} = w^{(t)} - \eta \cdot \nabla f(w)$, where η is a proper *learning rate* and t is the number of the iteration (or *round*, in this case).

This idea is usually implemented in a FL setting through algorithms that aim at approximating the optimization step of GD in a decentralized way given only a limited number of reachable clients. In particular in our work we focused on FedSGD, FedAdam and FedAVG.

8.3 Federated Optimization

In this section we present the federated optimization methods used in this work, namely FedSGD, FedAdam and FedAVG.

8.3.1 FedSGD

The Federated Stochastic Gradient Descent (FedSGD) is a FL optimization algorithm based on SGD. The optimization works in the following way.

At first we initialize the server model at random and we fix the learning rate η, the total number of rounds T and the percentage of clients that take part at each round. At each round we sample a set of clients (uniformly without replacement) among all the possible clients and we broadcast the server model to all the sampled clients. Then, for each client, we compute their gradient $\nabla f_c(w)$ given their training samples and we send back to the server only the information concerning the gradient. After that, we approximate the total gradient by averaging all the gradients weighted by the number of sampled n_c for client c. Then we compute the new weights with the optimization step of GD using the approximation of the gradient just computed, in the following way,

$$w^{(t+1)} = w^{(t)} - \eta \cdot \sum_{i=1}^{C} \frac{n_c}{n} \nabla f_i(w). \tag{5}$$

In our particular case we have a single sample for each client and so the approximation of the gradient in equation (5) becomes the simple mean.

8.3.2 FedAdam

We also tried a different version of FedSGD by changing the optimizer in the server side, maintaining the same overall structure. In particular we chose to

use Adam as optimizer, that is an optimization method more complex than SGD: if SGD keeps the same learning rate for each parameter, Adam, instead, uses per-parameters learning rates that change during time, because they are adapted to the first and second moments of the gradients, as described in [11].

8.3.3 FedAVG

FedAVG is a Federated optimization algorithm that was first introduced by *McMahan et al.* ([13]), who developed this method with the aim to reduce the communication costs between the server and the clients.

The FedSGD, in fact, suffers from an highly communication cost problem: FedAVG seeks to solve this problem by performing multiple epochs of training on the user's device before sending back to the server only the last updated weights. This should allow for a faster convergence with respect to the exchanges between the server and the clients.

The updated step in the server side is then computed as the average of the new weights of the clients, as follows:

$$w_{t+1} = \frac{1}{|C|}\sum_{i\in C} w_t^i = w_t - \frac{1}{|C|}\sum_{i\in C}(w_t - w_t^i), \tag{6}$$

where w_t are the weights of the server network at round t, w_t^i are the weights of the i-th client at round t and C is the set of clients that take part at the round.

Unlike what happens for the FedSGD case, with FedSGD we were not able to use the Adam optimizer since the training is done on several different networks, thus not allowing us to use the overall information about the first and second moment of the gradient. In the previous case this was possible since there was only a single trained network, namely the server network.

Moreover, we prove here the equivalence between the FedSGD with $\eta = 1$ and the FedAVG in the particular case of a single epoch training for each client and the optimization in the client side with a gradient descent with learning rate $\eta_c = 1$:

Proof. : Defining $\Delta_t \coloneqq \frac{1}{|C|}\sum_{i\in C}\Delta_t^i$, where $\Delta_t^i \coloneqq w_t - w_t^i$ we obtain that

$$\Delta_t^i = w_t - w_t^i = w_t - (w_t - \eta_c \cdot \nabla f_i(w_t^i)) = \eta_c \cdot \nabla f_i(w_t^i) = \nabla f_i(w_t^i), \tag{7}$$

where the second equivalence is due to the gradient descent update rule, because the learning in the client side consists of only one epoch, and the last equivalence is due to the fact that $\eta_c = 1$.

From equations (6) and (7) it follows that

$$w_{t+1} = w_t - \Delta_t = w_t - \frac{1}{|C|}\sum_{i\in C}\Delta_t^i = w_t - \frac{1}{|C|}\sum_{i\in C}\nabla f_i(w_t^i), \tag{8}$$

that corresponds to the server update rule of FedSGD when $\eta = 1$ and each client contribute with a single observation. □

8.4 Implementation

For the simulations with Federated Learning we implemented the algorithms using TensorFlow as deep learning framework. In order to prevent memory problems due to the large number of clients, we did not use a model for each client, but we took advantage of only 3 networks: a *server* network, a *client* network and a *temporary* network.

The *server network* is the central model that is updated after every round and that is broadcast to every client at the beginning of each new round. The second network, called *client network*, is the model that at each round is initialized with the weights of the server network, for each client: the gradient of this network (in the case of FedSGD and FedAdam) or its new weights (in the case of FedAVG) are gradually added together and assigned to a third network, the *temporary network*. The weights of this network are initially set to zero and after every round its weights are divided by the number of sampled clients. The resulting parameters are then assigned to the server network. Finally, the new weights of the temporary network are set to zero.

To further improve the computation, we calculated the mean of the gradients (or of the weights for the FedAVG) in an incremental way in order to avoid the problem of exploding of the gradient that could happen by constantly adding new terms. Indeed, after initializing g to 0, for each round we computed the overall approximation of the gradient as follows:

$$g \leftarrow g + \frac{\nabla f_{i_c}(w) - g}{c+1}, \quad \text{for } c \in \{1, ..., C\}, \tag{9}$$

where C is the number of clients sampled at each round and i_c is the index of the c-th client sample at each round. The formula in equation (9) corresponds to the formula of the incremental mean.

8.5 Experiments

In this Section we report the results obtained with our experiments in a federated environment. The algorithms exposed in Section 8.3 were used to train the models from scratch with the aim to compare the resulting models with the recommender obtained with a centralized training. All the results of this Section refers to the architecture of the best model found in Section 7.3, that consists in a PAE with the titles of the URLs.

In Table 14 are reported the numerical results obtained by changing the learning rates and the number of clients per round. All the models were trained for a number of rounds such that on average every client has been seen by the network in one round.

Firstly, we point out that the FedAVG was the most difficult algorithm to tune: we were not able to find a proper configuration of the parameters that

Method	η	Clients	Rounds	Rec@0.3	Prec@0.3	Hit@1	Hit@4	Tax
Standard	-	-	-	0.31	0.41	0.37	0.61	2.89
FedAVG	0.01	1	150K	0.25	0.06	0.02	0.10	4.45
FedAVG	0.1	5	30K	0.29	0.05	0.02	0.11	4.42
FedSGD	1	5	30K	0.12	0.19	0.14	0.34	3.94
FedSGD	1	1	150K	0.22	0.23	0.22	0.42	3.55
FedSGD	0.1	50	3K	0.7	0.17	0.12	0.33	4.07
FedAdam	0.1	5	30K	0.28	**0.34**	**0.33**	**0.56**	**3.06**
FedAdam	1	1	150K	**0.30**	0.27	0.30	0.33	3.21
FedAdam	0.1	50	3K	0.29	0.30	0.32	0.53	3.09

Table 14: Comparison of the federated algorithms. The number of round for each algorithm is such that every client has been seen by the model on average one time.

gave good results, except for the configuration shown in the *Proof* in Section 8.3.3. The results in the Table refer to the choice of two local training epochs for each client.

Furthermore, we noticed a considerable supremacy of FedAdam in terms of tuning of the parameters: it converged for all our trials, while for FedSGD it was harder to find a good learning rate and a number of clients that lead to convergence. In particular, as already said, we noticed that FedAVG with more than a single local training epoch is much more sensitive than the other methods. As observed also in [18], this could be due to the presence of heterogeneous data, that could lead to the *client drift* issue, i.e. situations where local client models move away from globally optimal models. From our experiments we noticed that this problem can be solved by incorporating in the optimization also knowledge about previous iterations (second moment of the gradient).

In addition to this, from Table 14 it is clear that for this particular task FedAdam is the best method, because it was the only algorithm able to reach a model whose performance is similar to that obtained with a standard learning.

Moreover, the training of the model took an amount of time comparable to the centralized model: we remark that this is a simulation time, and not the actual time that it would take with multiple devices, that would be definitely higher due to transmission time. On the other hand, however, we underline that in our simulations we were not able to compute the gradients of the clients in parallel. In the real-world application, instead, this can be possible because the computation of the gradients takes place in different devices.

In what follows we used FedAdam with learning rate equal to 0.1 and 5 clients per round, since it turned out to be the best algorithm.

8.6 Continual Learning for Federated Learning

In a centralized setting it is easily possible to solve the problem of knowledge forgetting (at least in our case) by merging data from the old and the new distribution, as shown in Section 7.4. In a federated environment, instead, we are not able to recover the old data, because we cannot collect it: the simulations done previously, indeed, have been possible because the old data is available to us.

In this Section we propose a solution for the Continual Learning of the model with Federated Learning, with the purpose to integrate the information about the new categories into the recommender system without losing the knowledge about the previous training. With the term *Continual Learning* (also known as Incremental Learning or Life-long Learning) we mean the ability to learn a model for a large number of tasks sequentially without forgetting knowledge obtained from the preceding tasks.

Since the data already used to train the model is not available, the algorithm we propose is based on the idea of generating a replica of it. In particular, we want that the data generated is as similar as possible to the dataset that was used for the previous learning of the model and that this contains the old information about the correlation between the categories.

Data Generation

We noticed that for those users that do not have any interest the model is inclined to suggest the most frequent categories, namely the categories that the model saw the most during the learning (see Table 9). Exploiting this behavior, we basically aim to infer the categories that were seen by the model during the training, together with their correlation.

Firstly, we compute the probability of generating a certain category by passing the zero vector to the model and normalizing the output in such a way that it sums to 1. Once the probabilities are computed, for each new user we sample a category according to these. Then, assuming that the model gives proper recommendations to the most frequent categories, we compute the output of the model considering as input the sampled category and we keep only the top 5 categories with highest correlation with the sampled one. At this point we normalize the output and we sample another (distinct) category according to these probabilities.

By doing this, we create users with only two interests each, but we suppose that this is enough for maintaining the knowledge about the previously learned categories (we remind that the original dataset was composed by mainly users with a couple of interests).

After the generation of the new dataset, we train the model with Federated Learning in a similar way to the idea we proposed in Section 7.4: at each round we sample a portion of clients from the actual clients (from the new data) and a smaller portion from the data we generated, in such a way that the model

learns the latest behavior of the clients while keeping what learned until that moment.

We then compared the original dataset with the generated one, with the aim to check if the data generated truly reflects the behavior of the actual input. We do this by checking how many times the interests of each generated user appear together in the interests' list of the users in the true dataset. In Figure 12 we reported the relative histogram obtained by generating 10k users: from there we noticed that the majority of the couples appeared together in the original dataset (composed by 400k users) more than 20k times. On the other hand we see that there are also couples that were never seen together in the actual dataset: we underline that all these couples are distinct, acting only as a little amount of noise in the re-training of the model. Furthermore, we remark that the couples created by our algorithm appear in the original dataset in mean 75k times.

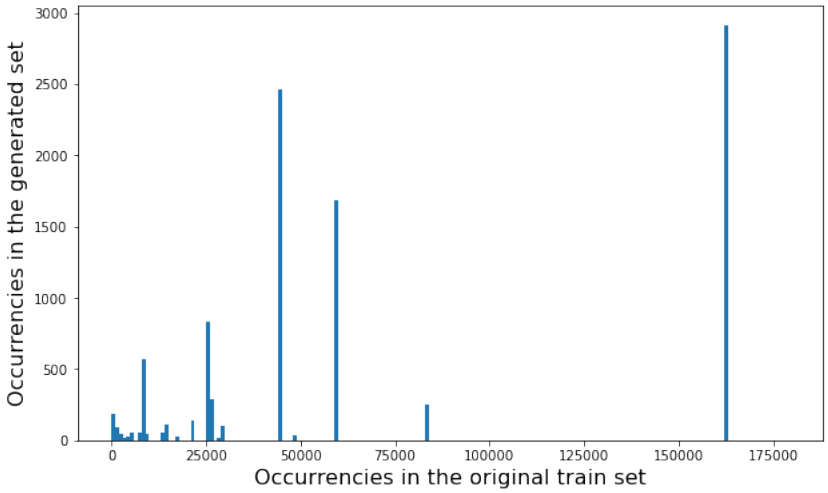


Figure 12: The histogram obtained by generating 10k users: it represents how many times the sampled couples appear together in the original dataset (composed by 400k users). From here we see that the majority of the couples created appears in more than 20k true users.

Experiments

In this section we report the results obtained with this algorithm, with the purpose to check if this approach is a good solution for the continual learning of the model with federated learning. We compare the model obtained through our solution with the model trained on all the data available. In particular, we want to compare their performance and check if the model trained with continual learning has actually kept the information about the first training.

Starting from a model pre-trained on the first portion of data with FL, we

generate a new dataset and we merge it with the data of the second period. The proportion of the two distributions is 1:4, in such a way that the main learning of the model is due to the new data, while the portion of old data is used just for keeping the old knowledge. Once obtained this new train set we use it to train the model with FL (using FedAdam with 5 clients per round and learning rate of 0.1). We then evaluate it using the true test data with the same proportion between the old and the new data.

By doing this (as shown in Table 15) we obtained a model with value of the taxonomy metric of 2.97, while the model trained on a centralized setting using the true data has a value of 2.84 for the same metric. As far the other metrics are concerned, the values of the centralized model are not so distant from the values obtained with federated learning, even if the values obtained with the FL are a little bit lower, as expected.

In addition to this we computed the performance of the model trained only on the second period of data with the standard learning and we evaluated it on the same validation set (with proportion 1:4 between the two distributions): we did this with the purpose to check if the model trained with continual learning was actually able to keep the information about the old data. By doing this we noticed that the performance is worse than the previous cases (last row of Table 15): this means that our solution really allowed the model to maintain the knowledge of the old data.

Type	Rec	Prec	Hit@1	Hit@4	Tax
Centralized (proportion 1:4)	0.33	0.40	0.37	0.62	2.84
FL with Continual Learning	0.29	0.37	0.35	0.57	2.97
Centralized on 2nd period of data	0.26	0.29	0.26	0.45	3.35

Table 15: Comparison of the result obtained with our Continual Learning solution and with the centralized training: the evaluation is done on data from both the first and the second period (with proportion 1:4).

9 Conclusions and Future Work

In this work we paved the way for a zero-trust process for the segmentation of the users by means of look-alikes values, based on Federated Learning. This process is totally privacy-preserving and compliant with the GDPR. Moreover, the final result is almost as good as the one we can reach with a centralized approach, in terms of quality of the segmentation.

This process starts with the data collection and processing, presented in Section 5, where we proposed a specific way to identify the interests of each user. In addition to this we studied techniques for a proper vectorization of textual information.

In Section 7 we proposed and designed a customized architecture for the recommender system: we studied the impact of several features finding an advantage on the use of the titles of the URLs. The entire process, from data processing to the creation of the look-alikes, was optimized and implemented in Apache Beam for a distributed computation.

The models were compared in terms of different metrics, introduced in Section 6. Among these, we proposed a tailored metric based on the taxonomy, that turned out to well represent the overall performance of the model: we think that this metric can be extended to other recommender systems whose items are taxonomy-based.

Furthermore, we proposed a solution for a proper refinement of the model on new items, in such a way that the model updates without losing the knowledge about the previously learned categories: in Section 8 we extended this idea to a federated environment proposing a process for continual learning for federated learning.

In addition to this, we compared several federated learning algorithms, finding a clear advantage of FedAdam over the other methods: among the algorithms we studied it is the only one that allows to train a model whose performance is similar to that we obtained in a centralized setting.

For future developments of this work we aim to improve the performance of the recommender system by training it on users whose click history is composed of a larger number of interests. In this project we dealt with users whose number of interests was limited (on average about two categories per person): this made our task quite difficult to tackle, and we think that a better characterization of the subject could significantly improve the quality of the suggestions.

Further research can be carried out by taking into account a new loss function that is "taxonomy aware", namely that considers how far the suggestions are from the ones we would like to suggest. In particular we want to properly weight the terms of the binary cross-entropy according to the taxonomy distance we formulated in this work (Section 6.2.3). By doing this we want to punish more the categories that are far from the masked category in the taxonomy.

Moreover, we want to proceed our experiments on our continual learning proposal for federated learning, verifying if this approach can also work with multiple training sessions and, eventually, if the quality of the data generated by the recommender gets worse over the time.

A Appendix

The refinement of the model on new data, in Section 7.4, allows us to obtain a recommender system whose suggestions are meaningful for a large number of categories. Here we reported some examples of suggestions, obtained using the scores and the zero matrix as titles. Further examples can be found in Section 7.4.

Automotive		*Home and Garden*	
Automotive	0.91	**Home & Garden**	0.77
Sports	0.438	Style & Fashion / Designer Clothing	0.46
News and Politics / Politics / War and Conflicts	0.29	Family and Relationships	0.27
Home & Garden	0.26	Real Estate	0.26
Home & Garden / Remodeling & Construction	0.25	Automotive	0.24
News and Politics	0.23	Style & Fashion	0.20
Business and Finance	0.20	Pop Culture	0.18
Family and Relationships	0.20	News and Politics	0.17
Healthy Living	0.19	Real Estate / Industrial Property	0.17

Table 16: Suggestions for the interests of *Automotive* and *Home and Garden*.

News and Politics and *Sports / Soccer*		*Pets / Dogs*	
News and Politics	0.99	Pets	0.99
Sports / Soccer	0.96	**Pets / Dogs**	0.99
Academic Interests / Life Sciences	0.74	Pets / Pet Adoptions	0.41
Sports / College Sports	0.67	Pets / Large Animals	0.40
Hobbies & Interests / Model Toys	0.66	Pets / Cats	0.19
Style & Fashion / Street Style	0.59	Real Estate	0.10
Automotive / Scooters	0.51	Music and Audio	0.07
Hobbies & Interests	0.39	Real Estate / Office Property	0.07
News and Politics / Weather	0.38	Family and Relationships	0.07

Table 17: Suggestions to a person interested in *News and Politics* and *Soccer* and interested in *Dogs*.

Music and Audio		*Sports / Soccer*	
Music and Audio	0.99	**Sports / Soccer**	0.85
Music and Audio / Talk Radio	0.98	Sports / College Sports	0.33
Television / Special Interest TV	0.42	Hobbies & Interests / Games and Puzzles	0.32
Academic Interests	0.36	Style & Fashion / Street Style	0.29
Sports	0.27	Real Estate / Land and Farms	0.27
Television	0.18	News and Politics	0.24
News and Politics	0.16	Food & Drink / Desserts and Baking	0.24
Academic Interests / Life Sciences	0.14	Pop Culture	0.22
Pop Culture	0.14	Sports / Auto Racing	0.21
Music and Audio / Adult Contemporary Music	0.12	Technology & Computing / Computing	0.21

Table 18: Suggestions for the interests of *Music and Audio* and *Soccer*.

References

[1] Waqar Ali et al. "A Federated Learning Approach for Privacy Protection in Context-Aware Recommender Systems". In: *The Computer Journal* 64 (Apr. 2021). DOI: 10.1093/comjnl/bxab025.

[2] Muhammad Ammad-ud-din et al. *Federated Collaborative Filtering for Privacy-Preserving Personalized Recommendation System*. 2019. DOI: 10.48550/ARXIV.1901.09888. URL: https://arxiv.org/abs/1901.09888.

[3] bluepi. *Classifying Different Types of Recommender Systems*. 2015. URL: https://www.bluepiit.com/blog/classifying-recommender-systems/#:~:text=There%5C%20are%5C%20majorly%5C%20six%5C%20types,system%5C%20and%5C%20Hybrid%5C%20recommender%5C%20system..

[4] Intersoft Consulting. *General Data Protection Regulation (GDPR)*. 2016. URL: https://gdpr-info.eu/.

[5] Jose Corbacho. *Federated Learning, Bringing Machine Learning to the edge with Kotlin and Android*. 2018. URL: https://proandroiddev.com/federated-learning-e79e054c33ef.

[6] Paul Covington, Jay Adams, and Emre Sargin. "Deep Neural Networks for YouTube Recommendations". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 191–198. ISBN: 9781450340359. DOI: 10.1145/2959100.2959190. URL: https://doi.org/10.1145/2959100.2959190.

[7] Leonardo Di Perna Giulio Ravasio. *GilBERTo: An Italian pretrained language model based on RoBERTa*. 2020. URL: https://github.com/idb-ita/GilBERTo.

[8] TensorFlow Hub. *universal-sentence-encoder-multilingual*. 2019. URL: https://tfhub.dev/google/universal-sentence-encoder-multilingual/3.

[9] IAB. *Content Taxonomy*. URL: https://www.iab.com/guidelines/content-taxonomy/.

[10] Giannis Karamanolakis et al. "Item Recommendation with Variational Autoencoders and Heterogeneous Priors". In: *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*. DLRS 2018. Vancouver, BC, Canada: Association for Computing Machinery, 2018, pp. 10–14. ISBN: 9781450366175. DOI: 10.1145/3270323.3270329. URL: https://doi.org/10.1145/3270323.3270329.

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[12] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. DOI: 10.1561/2200000056. URL: https://doi.org/10.1561%2F2200000056.

[13] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: (2016). DOI: 10.48550/ARXIV.1602.05629. URL: https://arxiv.org/abs/1602.05629.

[14] Paulina Zawiślak Michael Sweeney. *What is Programmatic Advertising? The Definitive Guide for 2022*. 2022. URL: https://clearcode.cc/blog/programmatic-advertising/.

[15] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[16] Vasileios Perifanis and Pavlos S. Efraimidis. "Federated Neural Collaborative Filtering". In: *Knowledge-Based Systems* 242 (Apr. 2022), p. 108441. DOI: 10.1016/j.knosys.2022.108441. URL: https://doi.org/10.1016%2Fj.knosys.2022.108441.

[17] Ola Rask. *What is Programmatic Advertising? The Ultimate 2022 Guide*. 2022. URL: https://www.match2one.com/blog/what-is-programmatic-advertising/.

[18] Sashank Reddi et al. *Adaptive Federated Optimization*. 2020. DOI: 10.48550/ARXIV.2003.00295. URL: https://arxiv.org/abs/2003.00295.

[19] Aaron Segal et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *CCS*. 2017. URL: https://eprint.iacr.org/2017/281.pdf.

[20] Chris Shuptrine. *GDPR and Ad Tech: The Definitive Guide for 2022*. 2022. URL: https://www.kevel.com/blog/gdpr-ad-tech/.

[21] Karan Singhal et al. *Federated Reconstruction: Partially Local Federated Learning*. 2021. DOI: 10.48550/ARXIV.2102.03448. URL: https://arxiv.org/abs/2102.03448.

[22] TensorFlow. *Federated Reconstruction for Matrix Factorization*. 2022. URL: https://tensorflow.google.cn/federated/tutorials/federated_reconstruction_for_matrix_factorization?hl=nn%5C&skip_cache=true.

[23] Chi-Feng Wang. *The Vanishing Gradient Problem*. 2019. URL: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[24] Yining Wang et al. *A Theoretical Analysis of NDCG Type Ranking Measures*. 2013. DOI: 10.48550/ARXIV.1304.6480. URL: https://arxiv.org/abs/1304.6480.

[25] Kang Wei et al. “Federated Learning With Differential Privacy: Algorithms and Performance Analysis”. In: *IEEE Transactions on Information Forensics and Security* 15 (2020), pp. 3454–3469. DOI: 10.1109/TIFS.2020.2988575.

[26] Qiang Yang et al. *Federated Learning*. 2019.

[27] Maciej Zawadziński. *What is Look-alike Modeling and How Does it Work?* 2020. URL: https://clearcode.cc/blog/look-alike-modeling/.

[28] Jinjin Zhao. “Collaborative Deep Denoising Autoencoder Framework for Recommendations”. In: 2019.