Cross-device tracking with machine learning

Elena Volkova

Thesis submitted for the degree of
Master in Informatics: Language and Communication
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017

Cross-device tracking with machine learning

Elena Volkova

Cross-device tracking with machine learning

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Personalizing the user experience on the web is important for news or product recommendations, online advertising and other domains. Users increasingly expect personalized experiences, and personalization technologies allow publishers to create more appealing digital products that they can charge for or otherwise monetize. On the web users are generally anonymous and the majority of traffic comes from users that haven't logged in or otherwise authenticated themselves. For this reason, one of the big challenges for personalization is the fact that the same user will often access websites from several different devices (e.g., PCs, mobile phones, or tablets) and the websites have problems discerning if the requests come from the same user or not. The problem of identifying users across multiple devices is known as cross-device tracking, and has not been extensively researched yet. In this project we carried out a number of experiments with cross-device tracking techniques based on applying machine learning to real-world traffic data. We extracted labelled datasets from traffic logs, applied both supervised classifiers and unsupervised clustering techniques to the data, focusing on minimizing the number of false connections, and evaluated them using the standard binary recall and precision metrics. Some of the resulting models performed well enough for practical applications, although important issues remain unsolved, such as how to make sure that a model performs well after applying it to the data from a new website.

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to say how grateful I am to my supervisor Aleksander Øhrn for all the help and guidance I have got from him in these past semesters.

Next, I would like to thank Simon Lia-Jonassen, Arne Sund, and everyone else that I worked with at Cxense, for providing me with all the necessary data and tools for this project, as well as for helping me learn how to use them.

I also thank my friends Sasha and Mikkel for their insightful comments and for reading through the text of this thesis with me.

Finally, I am grateful to my family for their understanding, moral support, and for keeping it down while I was working.

Chapter 1

Introduction

The way people surf the Internet today is considerably different from what it was just a few years ago. The number of devices with Internet access per user is constantly increasing: apart from PCs people now have tablets, mobile phones, and e-books that they can use to go online. According to a recent UiO policy [36], students are not even allowed to wear wrist watches during exams for the same reason: smart watches can have Internet access. Another significant point is that, according to the survey in [53], 90% of the users sometimes use several different devices to complete the same task. For instance, browsing interesting events in town, finding tickets and buying them can be performed from three different devices. Of the entire digital display advertising budget over 63% is predicted to be spent on mobile display in the next three years [54], which can also be indirect evidence of how actively people use their smart phones. As a consequence, user's online activities become spread across multiple devices, making it harder for websites to recognize them as the same person.

The reason why websites want to recognize the same user behind several devices is that it is important for personalization: tailoring recommendations, online ads, or search results to fit the specific user, based on their preference, search history, etc. Typical examples of personalization include recommended videos on YouTube, different search results on Google when you are logged into your account and when you are not, and being shown an advertisement from an online store after having searched for its products. Since one consumer can use several devices, recognizing them across device would allow the website to have a more complete profile of its visitors, thus improving the quality of personalization.

Personalization can be beneficial for both the consumer and the content provider. The consumers get the results that better fit their needs: personalization is one of the ways to avoid getting lost in the enormous amounts of data available on the Web nowadays. The content providers, in their turn, can make their websites more appealing, increase the number of page views and improve click-through rates on ads by making sure users see ads about something they might be interested in. The last reason is quite possibly the most important one, since digital advertising market is huge and rapidly growing: a total of 86 billion dollars is predicted to be spent on digital advertising in the US in 2020, for the first time replacing TV as the biggest advertising category [54]. Applying personalization techniques to newspaper websites has been known to considerably improve different performance measures such as engagement rate[1], CTR[2] on recommendations, or conversion rate[3] [38].

The challenge of connecting several devices that have accessed a website to the same user is known as cross-device tracking. The simple solution is to make the user log in to the website on every device: it is nothing new and has existed for a long time, because some services require user authentication regardless of whether their providers pursue the goal of cross-device tracking or not. Such services include e-mail, social networks, and similar cases. This solution is called "deterministic", as opposed to "probabilistic", which is based on machine learning, doesn't require a login and is therefore preferable to deterministic. See Section 2.2 for more details on both strategies. Both terms—"deterministic" and "probabilistic"—refer to the ways of controlling and exploiting web traffic between the user and the website. There is at least one more exotic and less ethical type of tracking, also referred to as "cross-device tracking": it was first developed by a company named SilverEdge, and it makes use of the physical proximity of the devices to each other [34]. One device emits a noise on an inaudible frequency, which is picked up by an app on another device, which allows it to make a connection between the two devices. The technology came to be known as "audio beacons". This type of tracking, though it attracted the attention of the media and raised awareness of the problem itself, is not discussed in this thesis.

Even though traffic-based cross-device tracking is a relatively new field, there already is a range of companies that offer their services in probabilistic cross-device tracking: Drawbridge, Tapad, Adbrain, etc. There are, however, very few scientific works published on the topic.

This thesis is based on a research project about probabilistic cross-device tracking. It is organized as follows: Chapter 2 ("Cross-device tracking: background") provides an overview of what cross-device tracking is, the main methods and strategies, discussion of previous research on the topic, and mentions some technical and ethical challenges raised by the problem. Chapter 3 ("Cross-device tracking: the project") describes in detail attempts to implement a cross-device tracking algorithm using web traffic data: constructing the dataset, feature engineering, training, evaluation, etc. It deals with both supervised and unsupervised machine learning methods. Finally, Chapter 4 ("Discussion and conclusions") summarizes the results of the project and proposes directions for further work.

[1]Engagement rate: average amount of time spent by a user, actively interacting with the webpage.

[2]CTR (click-through rate): percentage of users who click on a specific link among all the users who see it on a page

[3]Conversion rate: percentage of users who become paying subscribers.

Chapter 2

Background

This chapter gives an overview of cross-device tracking: the motivation for it, common approaches to solving it, previous research, as well as technical challenges and ethical concerns that it raises.

2.1 Motivation

As was mentioned in the introduction, cross-device tracking is crucial for personalization purposes, i.e. it is important to be able to utilize the available information about the user across all devices they own. However, it is not the only application. A survey in [10] lists what marketers can do with the help of cross-device tracking, which include:

Impression capping. Not only can advertisement be personalized better, but it can also be ensured that one user is not shown the same ad more than a certain number of times across all of their devices.

Conversion uplift. Marketers can better understand which media channels are worth investing more into, so that more users become permanent subscribers.

Better understanding of the audience. When the users' identities are not split into pieces over multiple devices, it is possible to get more accurate statistics of the audience, research audience segments and so on.

From a more scientific perspective, research into cross-device tracking can contribute to answering more theoretical, but nevertheless important questions about the relationship between an individual and the Internet, such as:

- Is anonymity on the Internet possible, or are we defined by our online activity well enough to be recognized regardless of the physical device we are using?
- If it is the latter, how much information is needed to successfully match two online footprints, and what kind of information is most useful for that?

- How should cross-device tracking be performed to respect the privacy of the users?

2.2 Strategies

There are two main strategies for cross-device tracking: deterministic and probabilistic. Deterministic cross-device tracking is done with a unique identifier, or login, that a user has to get to access the website. The website then requires users to login on every device, browser or app that they use. One of the most popular and accessible tools for this is Google Analytics, which can link devices either making use of the IDs the website assigns to its visitors, or of their Google accounts, provided they have it and are logged in. Of course, this method is not faultless: both accounts and devices can sometimes be shared by several people; but despite the occasional exceptions, it is a simple and effective way to identify users across devices. However, there are obvious disadvantages: not all websites are able to urge their visitors to get and use a login. Logins are common for online stores, forums, social networks and other types of web pages whose require exact identification of the user to function, but are not as common, for example, for news sites or product landing pages. Even though most newspaper websites provide an option of getting an account, making it mandatory is not desirable because this can lead to fewer readers. There are, of course, exceptions, such as, for instance, the *Wall Street Journal*, where unregistered users get very limited access to content, but as it is a business-focused newspaper, it is directed at a specific audience. When loggin in is not mandatory, it can be hard to motivate users to register and, most importantly, identify themselves every time they access the website. Some incentives that publishers use for that purpose include making a part of the content only accessible after logging in (*Rebulic* [45]), allowing comments only from registered users (*Bergens Tidende*) or providing the option to share the content via social networks (almost any online publisher).

The alternative to the deterministic approach is probabilistic cross-device tracking. It involves building a model that would try to predict whether two devices that have accessed the website belong to the same user, based on session information such as time, location, viewed content, etc. The model is usually built with the help of machine learning methods (see Section 2.3. The obvious disadvantage, compared to the deterministic approach, is the uncertainty: these methods estimate the probability of two devices belonging to the same user based on the model they have from previous data, but they can't guarantee a 100% correct answer. Another disadvantage is the complexity of this solution. While it requires no effort on the part of the user, the website operator has to create a prediction model using the data they have. As the existing solutions are normally based on supervised machine learning (see Section 2.5 for related works), they need a big enough sample of training data first. Such data can be expensive and hard to obtain in sufficient amounts. Besides, the resulting

	Deterministic	Probabilistic
Advantages	Exact predictions Simple and fast	Login not required
Disadvantages	Login required	Not 100% accurate Complex and takes time

Table 2.1: Comparison of deterministic and probabilistic cross-device tracking.

predictive model can be quite complex itself, so an additional challenge for probabilistic cross-device tracking is to make sure the model can be used efficiently. All challenges considered, it is quite difficult for the website owner to create the necessary software themselves, so there are a number of companies that offer their services in cross-device tracking, combining both approaches.

Table 2.1 summarizes the differences between the two mentioned strategies.

As shown in this Section, deterministic cross-device tracking is quite straightforward to implement, so this thesis is going to focus on the probabilistic approach instead. For convenience, further on the term ″cross-device tracking″ refers to ″probabilistic cross-device tracking″.

2.3 Methodology

Most good known solutions are based on supervised machine learning, as described below in Section 2.5. Indeed, cross-device tracking is a task that has to do with recognizing patterns in big samples of data with unknown distributions and properties, which makes it a suitable target for machine learning algorithms.

Arthur Lee Samuel, a pioneer of machine learning, defined it as ″programming of a digital computer to behave in a way which, if done by human beings or animals, would be described as involving the process of learning. [...] Programming computers to learn from experience should eventually eliminate the need for much of the detailed programming effort″ [48]. In other words, machine learning is a set of techniques for programming computers to analyze data and infer conclusions from examples rather than explicitly programming a list of instructions. It is widely used in bioinformatics, information retrieval, natural language processing and many other domains.

The common preprocessing step for any type of machine learning is the way data is represented. Every instance of the input data has to be made into a vector of values, where each position in the vector corresponds to a variable, or feature. Features are some properties of the instances that might be useful for solving the task. With modern hardware the number of such properties can be quite high: in image processing it is not unusual to have tens of thousands of features. It is not obvious in advance what information about the instances is useful and what is not, so selecting and constructing features is a separate subfield of machine learning called feature engineering. After that a machine learning algorithm, taking feature vectors as input, analyzes them and somehow models the distribution of the data.

There are three main strategies for machine learning:

1. Supervised. A supervised learning algorithm takes as input a set of labelled instances (training set), i.e. a set of feature vectors, each with a corresponding label. The label is the value of a dependent variable that we want to predict. It then tries to fit the data into a previously chosen mathematical model by optimizing its parameters. The model with the correct parameters is then used to predict the label for new, unlabelled instances. The set of unlabelled instances used to test the model and evaluate its performance is, respectively, called the test set. This method was used by all the authors of the articles described in Section 2.5. The term ”classification” is often used synonymously with ”supervised learning”, although technically it stands for both regression and classification. The difference is that during classification the predicted variable is discrete, its values being two or more labels corresponding to the classes, while a regression algorithm predicts a continuous variable. A model that was created using a classification algorithm, and even the algorithm itself, is often referred to as a ”classifier”.

   There exists a great number of machine learning libraries for different programming languages and environments, and each one provides the most common classifiers, such as Naïve Bayes, support vector machines, decision trees, etc. Classifiers differ in how they model data, what kind of datasets they are best suited for, what limitations they have, as well as in time and memory efficiency. Choosing the right algorithm is the next important decision after feature engineering. The basic principles for several classifiers, used in this project, are described in Section 3.3.4.2.

   The biggest disadvantage of supervised learning, for our problem as well as many others, is the need for a labelled training set. Acquiring such a set can be expensive and time-consuming, or sometimes even impossible, because the data is too scarce. The process of extracting the training set for the present project is described in detail in 3.3.1.1.

2. Unsupervised. Unlike supervised learning, it does not require labels. A typical task for unsupervised learning is arranging data into clusters based on their feature values. Common algorithms, also provided by most machine learning libraries, include *k*-means and agglomerative clustering. The main idea of any unsupervised learning method is to introduce a metric on instances in order to be able to compute how close two instances are to each other. Since instances already are numerical vectors, it is natural to look at them as points in a *k*-dimensional vector space, where *k* is the number of

	Classified as a match	Classified as non-match
Actual match	True positives	False negatives
Actual non-match	False positives	True negatives

Table 2.2: Confusion matrix for cross-device tracking.

features, and then apply any common geometrical metric like, for example, Euclidean or cosine distance.

There was not much work involving unsupervised learning in the course of this project. Some experiments with this approach are described in Chapter 3.4.

3. Semi-supervised. When there are no or few labelled instances available, certain techniques can be used to augment the training set. The authors of [16] added a semi-supervised component to their solutions. However, neither we, nor any of the authors of related works report data shortage. On the contrary, most of them mention how they had to downsize the number of data points and introduce limitations to make the problem more tractable. We briefly discuss possible semi-supervised approaches in Section 3.4.2.

2.4 Evaluation

2.4.1 Confusion matrix

Whatever approach is chosen for performing cross-device tracking, the output of the system would normally be in the form of a set of pairs of user identifiers that, according to the system, belong to the same person (or a "match"). These pairs make up the first class of instances. All other possible pairs, not included in the result set, constitute the second class. Most evaluation metrics are based on the confusion matrix, which is a table where the columns represent classes, predicted by the system, and the rows represent actual classes of the instances in the test set. The value of the cell (m, n) is the number of instances from class m which got attributed by the system to class n. For binary classification problems such as cross-device tracking the size of the confusion matrix is 2×2. For convenience we will call the matching class "positive" and the non-matching – "negative". Table 2.2 shows what the confusion matrix for cross-device tracking looks like.

True positives are the instances that were correctly attributed to the positive class, i.e. appeared in the result set; true negatives are the instances that were not a match and were not in the results; false positives are the pairs in the results set that are not actually a match; false negatives are the pairs that should have been in the result set, but were not discovered by the system.

2.4.2 Precision, recall, F-score and accuracy

The most common classification evaluation metrics are accuracy, precision, recall and F$_1$-score. The formulas for the metrics are as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1\text{-}score = 2 \cdot \frac{Precision + Recall}{Precision \cdot Recall}$$

Although some systems report very high accuracy [17, 46], it is not the most suitable metric for cross-device tracking. The problem with accuracy is that the device graph is not too dense, and the size of the negative class is much larger than the positive class, unless the negative class is artificially downsized (no such downsizing was mentioned in the reports). For instance, if there are N mobile phones and N desktop computers, and each phone is connected to exactly one computer, the positive class contains only N pairs, while the negative class contains $N(N-1)$ pairs. In this case the system that blindly assigns every pair to the negative class achieves $accuracy = N(N-1)/[N(N-1)+N] = N-1/N$, which is $> 90\%$ for every $N > 10$.

The more appropriate metrics for cross-device tracking are recall and precision, since they are directed at the positive class. According to [10], some companies create two versions of their systems, one with high recall and another with high precision. High recall systems are more useful for reaching and targeting individual users, while high precision helps analyze the audience more accurately. F$_1$-score represents the trade-off between recall and precision.

The F$_1$-score is the basic version of the F-score, which weighs precision and recall equally. That is why the subscript 1 is often omitted, and the metric is referred to as just the F-score. For cases when we wish to give more weight to either precision or recall, there exists a more general version of the F-score, with an adjustable coefficient β:

$$F_\beta\text{-}score = (1+\beta^2) \cdot \frac{precsion \cdot recall}{(\beta^2 \cdot precision) \cdot recall}$$

2.4.3 ROC and AUC

All the metrics above are defined on the assumption that the output of the classifier is binary: one of the two class labels. But underlying this output is normally a score, or a probability, based on which the label is decided. By default the threshold for the score to assign an instance to one of two classes lies at 0.5 (provided the score is normalized to the interval from 0 to 1, which is usually the case). By moving the threshold we can change the sensitivity of the classifier, i.e. make the trade-off between recall and precision favor one of those metrics more (see Section 3.3.4.3.7). For example, if the threshold is increased, then the model is required to be more certain to assign an instance to the positive class, which leads to better precision, but lower recall. It should be noted that this pattern is only true if the model is more or less calibrated (see Section 3.3.4.3.5).

By moving the threshold and changing the sensitivity of the classifier we can draw a curve of how the true positive rate depends on the false positive rate. True positive rate is the same as recall, and false positive rate is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

This curve is called Receiver Operating Characteristic curve. Figure 2.1 is an example of such a curve.

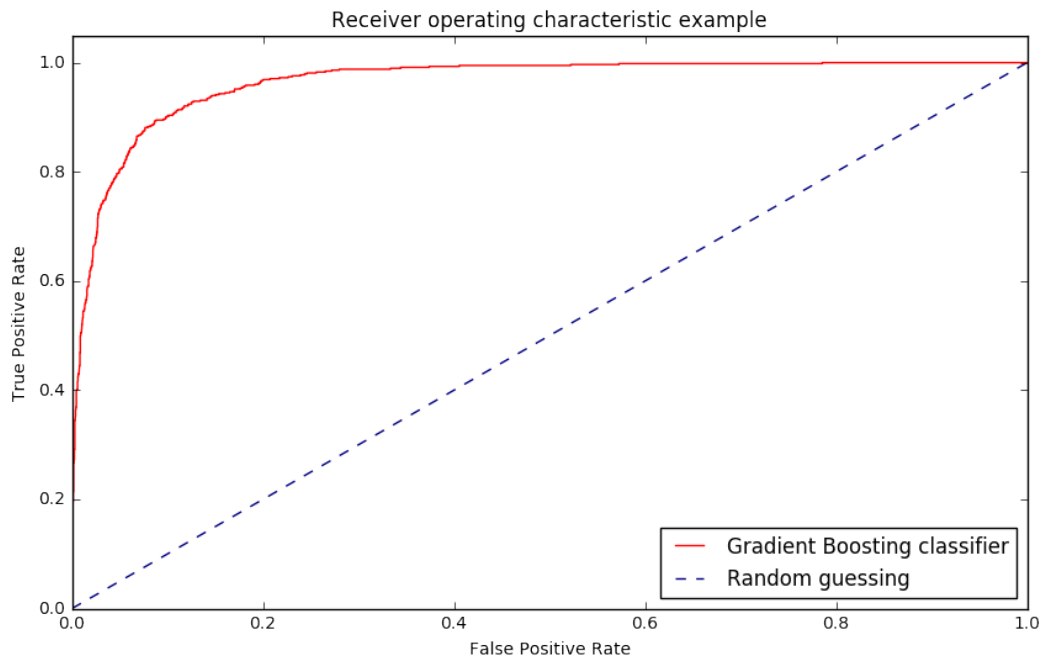


Figure 2.1: An example of ROC curve for a Gradient Boosted Tree-based classifier, trained and tested on a randomly drawn and split subsample of instances collected over one-month period from WSJ.

Naturally, the curve includes points with coordinates $(0, 0)$ and $(1, 1)$: $(0, 0)$ corresponds to the threshold 1.0, when everything is assigned to the negative class, and $(1, 1)$ corresponds to the threshold 0.0, when everything is assigned to the positive class. If the curve is a straight line between those two points, then at any given threshold the classifier assigns positive and negative instances to the two classes in equal proportions, i.e. the distribution of actual positive and negative instances is the same in both classes. That kind of behavior is as useful as random guessing, so a good ROC curve should lie above the line $TPR = FPR$, like in Figure 2.1. The curve can help choose the optimal decision threshold: sometimes it is is more important to minimize FPR, other times high TPR is the main goal. When the dimensions are equally important, there is more than one way to choose a good threshold: for instance, taking the one corresponding to the point on the curve that is closest to $(0, 1)$, or the one that maximizes $2 + TPR - FPR$ (the Youden index [58]).

The curve itself gives useful insights into the relation between different types of errors of the classifier, but it is a sequence of numbers, unlike the previously described metrics. If a single numeric score based on the ROC curve is needed, then the area under the curve (AUC) is used, i.e. the integral of the ROC curve over the interval $(0, 1)$. The bigger the AUC value, the better; for random guessing its value is 0.5.

2.5 Related work

As was mentioned in the introduction, there is not a lot of scientific literature on cross-device tracking. It is partially due to the fact that the problem itself is relatively new, since it only appeared when people started actively using several devices with Internet access.

There are some areas of research that are loosely related to cross-device tracking and could be considered its predecessors, since they pursue similar goals and share much of the same methodology. For instance, a great number of articles was published on producing media content recommendations and personalized web user experience. Exploiting the trail the user leaves by showing their interest in online content was used in recommendation systems as old as Tapestry, an article about which came out in 1992 [21], or Siteseer [47], which incorporated both content-based filtering (based on similarity between the content of the item and of the user profile) and collaborative filtering (recommending items that similar users liked). Later most works in recommendation started using machine learning techniques, both supervised and unsupervised, like Naïve Bayes classification for constructing user profiles [40], Probabilistic Latent Semantic Analysis for discovering audience segments [57], k-means clustering for expanding the previously established audience segments [44], and so on. Among the most recent studies machine learning is the prevalent methodology, and the focus is often on its particular aspects like efficiency or feature engineering.

The fact that there are very few articles on cross-device tracking itself could also be due to the commercial benefits of selling cross-device tracking software. Tapad, one of the companies that specialize in it, has a patent application for a cross-device tracking method combining probabilistic and deterministic approaches [33]. Another such company, Drawbridge, was granted a patent on their cross-device technology in 2016 [52]. A whitepaper has been released by Adbrain, who also provides cross-device solutions, describing various aspects of cross-device tracking from the marketer's point of view [10]. Drawbridge has also organized a competition in recognizing cross-device connections [25].

Drawbridge competition

The competition in cross-device tracking, that more than 300 teams took part in, was hosted by Kaggle [25]. Some of the participants have published articles on the methods they used to address the task, which are likely the first computer science articles on the topic: two of the authors mentioned that they had not been able to find any research papers on cross-device tracking [16] [15]. The data provided to the participants consisted of several tables with information about mobile phone identifiers (referred to as "devices" in the competition description), computer identifiers (referred to as "cookies"), and IP addresses. That information included individual properties of each device and cookie, such as model or browser, the IP addresses where they appeared, websites that they visited, etc. All personal identifiers in the dataset were fabricated so that they would not cause any privacy violations. The dataset also included an identifier called the "Drawbridge handle", which was the same for devices and cookies if and only if they were indeed from the same user. The test set was similar, but it didn't provide information about the Drawbridge handle for devices. The task of the competition was therefore to create a model for predicting which cookies come from the same user as the given device, based on the dataset provided for training. Later the model would be evaluated on the test set, which included a list of 61156 devices to find matching cookies for. The main evaluation metric for the competition was mean $F_{0.5}$-score, which combines precision and recall (more about evaluation in 2.4).

Several articles have been published by the teams who were in the top 10 on the competition leaderboard. All of them used supervised machine learning methods, and most teams treated the task as a classification problem, where the items for classification were pairs in the form $(device, cookie)$, and each of them belonged to one of the two classes: "from the same user" or "not from the same user" (see Figure 2.2) [16, 27, 28, 30].

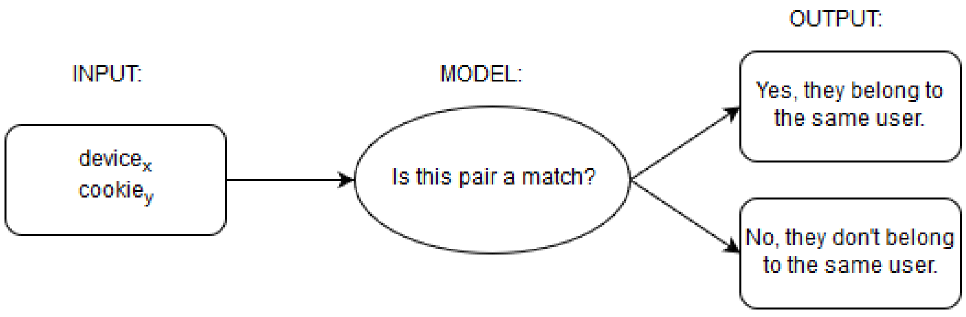

INPUT:
device$_x$
cookie$_y$
MODEL:
Is this pair a match?
OUTPUT:
Yes, they belong to the same user.
No, they don't belong to the same user.

Figure 2.2: Cross-device tracking as a classification problem.

The winning team used triples $(device, cookie, IP\ address)$ instead of pairs, but they did not consider this to be the decisive factor for their performance [56]. Instead of a "strict" classification all the teams used probabilistic classification, assigning a probability to the fact that an item belongs to a class, rather than strictly attributing it to one class. One team formulated the problem as regression, which is essentially the same as probabilistic classification [50]. Two teams chose a slightly different approach and used ranking models as their final solution. Ranking models differ subtly from classification, although they require similar training data. Let S_d be the set of cookies that have been via some data preprocessing chosen as possible matches for the device d. A ranking model finds the correct ordering on S_d, such that the first cookie is the most probable candidate. For [55] the ranking model was the reason they won the competition, according to the author of the article.

Building the prediction model was the core stage of solving the task. [16], [30], [28] and [55] used very similar algorithms based on gradient boosting (see below) and even utilized the same implementation, provided by XGBoost library. The differences in the performance are therefore influenced not only by the choice of the algorithm, but by how the details are handled when the algorithm is applied to the data.

The data provided by Drawbridge was not specifically tailored for classification: it contained tables with information on individual devices, cookies and IP addresses, which meant that the first step for the participants would be creating the training set for the prediction model. As the training data contained over 140K devices and 2M cookies, the set of all possible pairs would have been too large to train a model, so it had to be downsized first. [16], [30], [50] and [28] report reducing the number of device-cookie pairs in the training set to approximately 3.66M, 1.11M, 900K and 700K correspondingly. Such reductions, or pruning, of the data can also be used later during testing to determine which cookies are good candidates for a given device, because computing the probability of a match with every one of the 2M cookies would take too much time and be inapplicable in real-life situations. Common methods for pruning are based on the idea that matching devices and cookies have to share IP addresses (for short, IPs), and, according to [15], 99.90% of them do. Thus disregarding all other cookies as candidates for the given device leads to a very slight drop of recall, but a big advantage in processing time. Another reduction method is getting rid of cellular IPs [15, 27, 28, 55], which are very unreliable for user tracking [11], and IPs that have too many devices or cookies that use them (likely from publicly used networks) [15, 16].

Another important subtask is feature engineering. The participating teams proposed a wide variety of features for representing device-cookie pairs. Apart from the basic device and cookie properties available from the provided tables,—like device type, OS, or browser,—most solutions also used IP-based features: how many IPs are shared by the cookie and the device, the average values of the properties of those IPs, etc. Some of the authors also propose to aggregate features based on the relations between items in the training set, which are generally more difficult for a classifier to pick up. For instance, [28] add to the list of features the cosine similarity between the device and the cookie vector, containing the properties of their common IPs. They also proposed a tf–idf weighting scheme for finding the feature values that are most characteristic of a class. [30] created a new feature for every IP-property available in the dataset: they first computed the sum of values of this property over all IPs linked to the cookie, then over all IPs linked both to the cookie and the device, and then used the ratio between the two sums as the feature value. According to the article, this type of features raised the F-score by about 10%. The IP-based features were found more useful than those based on the website history [30]. One of the teams even dropped the features not related to IP address and performed the tracking based solely on the IP footprint of the users [15]. The number of features also varies considerably, from 38 in [27] to in 699 [55].

The choice of the loss function also influenced the results. The optimization in gradient boosting is performed through minimizing a loss function, which penalizes the model for training instances that do not fit, given model parameters. The main difference between binary classification and ranking is the loss function. The function for ranking is more lenient in that it does not penalize discrepancies in scores between different devices, only between cookies from the same device candidate set. In other words, even if the system scores a non-matching pair $(device_1, cookie_1)$ higher than a matching pair $(device_2, cookie_2)$, it does not influence the performance, as long as the ranking inside the candidate sets for both devices is correct.

Finally, after defining the set of candidate cookies for a device and computing the probability that any of them are a match, the system has to decide on the threshold for that probability to attribute a cookie to the same user as the device. For that the participants used empirical thresholds [27], lists of rules [28] and even ensembles of trained predictors [15].

Beside the common subtasks listed above some of the participants employed additional techniques commonly used to improve the results in machine learning: bagging, or training several models and averaging them in order to avoid overfitting and make the model more robust [16] [30], and semi-supervised learning, i.e. augmenting the training set by the items from the development set with an unmistakably high probability of a match [16]].

2.6 Challenges

There are several common challenges that have to be met to successfully create a cross-device tracking system. First of all, the solutions described in Section 2.5 all use supervised machine learning, but that requires a labelled training set. One way to create the training set is to use the login data for verifying that two devices belong to the same user. However, this data can be scarce or not representative in case the system is being built on a smaller scale than Drawbridge. Another way is to apply semi-supervised and unsupervised learning. The former was used to increase the training set in [16], while on the latter no research has been published so far. The construction of the training set for our project is described in detail in Section 3.3.1.

The second important problem is that even if there is enough login data from some websites to construct a training set, it is not every website that has enough traffic from logged-in users to make a separate training set. It means that the same model that has been trained on whatever source or sources had enough data, has to be used for other sources as well, that had not been included into the training set. It is a well-known problem in machine learning that training data and production or test data might have some differences, known as covariate shift or dataset shift, if they are from different sources. See Section 3.3.4.5 for the results of transferring the model to a new source.

Next, unlike some problems where machine learning is involved only because labelling instances manually is expensive and time-consuming, in cross-device tracking it is strictly speaking impossible. The training set that we obtain in this project is built through a heuristic that employs registered users, but how data is distributed and how the model performs on the traffic from unregistered users is impossible to know.

Finally, there is the matter of scalability: for real-life applications it is no less important than the performance of the system. Even if the algorithm for determining whether a pair is a match is efficient, the number of pairs it is applied to has to be reduced to make the whole system efficient as well. Several ways of pruning are mentioned in Section 2.5, but the highest-scoring team in the Drawbridge competition still reported that processing the test set took $>$ 8 hours even on powerful hardware [55].

2.7 Ethical concerns

Both personalization in general and cross-device tracking in particular have given rise to ethical and legal concerns in the past several years. At a TED conference in 2011 Eli Pariser, political and internet activist, talked about how the personalization of search results and news feeds creates informational bubbles around individual users so that they get different results for the same queries and ultimately get a different picture of the world around them [39]. In 2012 an article came out about how the online preferences of the users, namely ”likes” on Facebook, could help predict personal details like sexual orientation or political views with very good accuracy [29]. One of the authors of the article, Michal Kosinski, even expressed his concerns in an interview that a project based on his research might have played a role in the 2016 US elections, allowing to target voters by their predicted psychometric profiles instead of just demographics [22]. In 2015 a Federal Trade Commission workshop was held in the USA on the subject of cross-device tracking, the main point of which was that such techniques are not at all controllable by the user. This primarily concerns probabilistic tracking and audio beacons, since in those cases it is totally invisible to consumers. The FTC issued warnings to some developers who used user-tracking software without an explicit opt-out option [20]. Yet the privacy issues around cross-device tracking are still not widely discussed or regulated, and it is not clear who in this case is responsible for the privacy and safety of the users’ personal data [14].

Chapter 3

The project

This chapter describes a series of experiments on cross-device tracking that have been carried out for this MSc project. They apply cross-device tracking theory to real-life web traffic data, making use of both supervised and unsupervised machine learning methods.

3.1 Goals

The main goal of the project is to create a probabilistic cross-device tracking model that would identify some of the devices that belong to the same user in the stream of traffic data from a website. Ideally the model has to perform well on different websites, including the ones that did not contribute to the training set. It is not necessary to achieve complete coverage, i.e. identify every matching pair, but the identification should be reliable, i.e. false positives are more undesirable than false negatives. In other words, the project focuses more on precision than on recall: if a system manages to identify every tenth pair of devices belonging to the same person, but does it with high precision, it would mean that the goal was achieved. The intermediate goals include finding ways to adapt common traffic data to cross-device tracking, comparing how suitable different media websites are for building a cross-device tracking model, and other possible insights into the problem.

3.2 Data sources

The data for the project was provided by Cxense. It comes from a particular type of websites: online newspapers and magazines. It is raw traffic data, which consists of information about "events", i.e. clicks on the web pages, from a number of media websites for the time period of one or two months. To get a good idea of how the problem varies from source to source we took three newspapers with quite different geographical profiles. For convenience we'll use abbreviations for them:

1. **WSJ.** An international newspaper written mostly in English with more than 100 million page views and millions of online readers per month

across the world;

2. **ELP.** A Spanish-language newspaper, that has fewer clients than WSJ, but is still read internationally, mostly in Spain and South America, with close to 100 millions page views per month;

3. **WFP.** A smaller regional English-language newspaper with about 8 million page views and 900 000 readers per month.

It should be noted, of course, that we do not know the exact number of readers, mainly because we cannot yet track them across devices: we can only count the cookies. The number of page views is therefore a more reliable metric.

Raw event logs record visits to the website in chronological order. Each such event has a number of properties that are stored in its fields, such as: the exact time the event happened, the address of the webpage that was visited, the browser that was used, and many others. One of the most important fields for the experiment was the unique device identifier, or device_id, which is in fact the first-party cookie. This identifier is a relatively safe way to find all the events made from the same physical device-browser combination. There are some exceptions to that, such as a user who deletes their cookies, but they were not addressed in the experiments for several reasons: first, for a popular media website there is no shortage of data even when the identification of devices is not complete; second, deleting the cookies signals about the unwillingness to get tracked, and should be respected at least to some extent.

3.3 Supervised learning

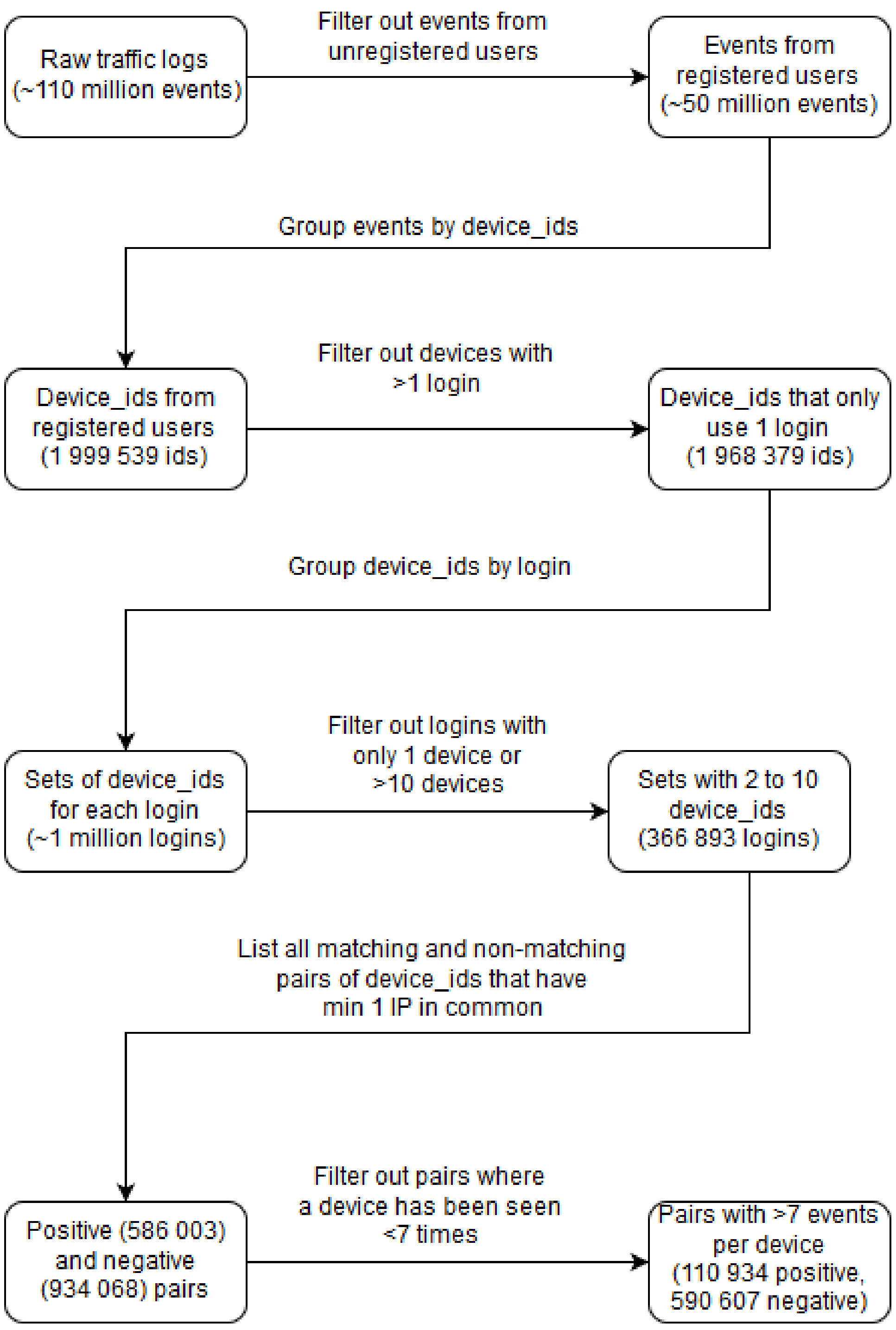
This section describes experiments on solving the cross-device tracking problem with supervised machine learning, namely classification. This is the only method used in the previous works on the topic, and it constitutes the bigger part of this project as well.

3.3.1 Preprocessing

3.3.1.1 Creating the training set

Since the data was not specifically tailored for making a cross-device tracker, the first step of the experiment was data preprocessing. Preprocessing is essentially a sequence of filters to get rid of unusable and inconvenient data. Figure 3.1 shows the outline of this step, as well as how much was eliminated by every filter.

First, we can only use the events from users who have a login: no matter what machine learning algorithms are used, supervised or unsupervised, they need to be evaluated, which means that for every prediction made by the algorithm on our data we have to be able to tell if the prediction is true or not. That is done through an identifier that corresponds to the login and is recorded in every event that happened from a registered user.



Raw traffic logs (~110 million events)
Filter out events from unregistered users
Events from registered users (~50 million events)
Group events by device_ids
Device_ids from registered users (1 999 539 ids)
Filter out devices with >1 login
Device_ids that only use 1 login (1 968 379 ids)
Group device_ids by login
Sets of device_ids for each login (~1 million logins)
Filter out logins with only 1 device or >10 devices
Sets with 2 to 10 device_ids (366 893 logins)
List all matching and non-matching pairs of device_ids that have min 1 IP in common
Positive (586 003) and negative (934 068) pairs
Filter out pairs where a device has been seen <7 times
Pairs with >7 events per device (110 934 positive, 590 607 negative)

Figure 3.1: The steps of preprocessing from raw logs to a set of pairs ready for training. The data is taken from WSJ for one month.

From some devices several logins were used: it could be a public computer, or a device that is used by several members of the household, yet each of them has their own account. Whatever the reason is, there are relatively few such cases: Figure 3.1 shows that for WSJ out of almost two million device_ids only 31 160 devices are shared. Since the main concern of cross-device tracking models in this project is precision, and not complete coverage (see Section 3.1), all events from those devices were removed from the dataset.

What interests us most is the opposite situation: the cases when one login has been seen with several different device_ids. That is when cross-device tracking can be applied. Two devices are considered to belong to the same user if the same login has been used on both of them. However, there are some exceptions to that, and not any such case is usable. Some logins have been seen on over a thousand different devices: they might belong to a user who clears cookies, or uses incognito mode all the time, or it could be a corporate account, or the combination of those factors. Fortunately, such cases are also very rare: if a one-month period is considered, most logins have only been used from one device. Figure 3.2 shows how quickly the number of logins drops as we increase the number of device_ids, connected to them. The higher the number of device_ids, the less realistic it is that the login is used privately by one person, so for further work it was decided to cut the number of devices at 10. Even with a relatively small number of devices that used the login there is still a possibility that the account is used by more than one person, especially if the subscription is not free, but there is no apparent way to correct this, so we have to accept some noise in the dataset.

In order to get from event-based to device-based representation one has to group the events by device_ids. Here we introduce the first substantial restriction: device_ids with less than k events are not included in the training data, where k is a small positive number normally between 5 and 10. This has made the results considerably better, since a lot of features are based on correlations between event sequences for the two devices (see Section 3.3.2)[1].

For a supervised approach we need to get from individual devices to labelled pairs. We have already made sure that each device_id is connected to exactly one login, so the data can be grouped by logins. It would then be represented by a number of sets of device_ids, where the device_ids in the same set share the same login. Supervised learning, as described in Section 2.2, requires a training set with examples of all classes. In our case there are two such classes: positive, made up of pairs of devices that share the same login, and negative, made up of pairs that don't. Section 2.5 mentions two slightly different approaches to what an instance of the training set should look like: a pair $(device_1, device_2)$ or a triple $(device_1, device_2, IP\ address)$. Triples increase both training time and the number of possible instances, so the former approach will be used.

[1]For technical reasons this step was performed later in the program, after the pairing step, as it is shown in Figure 3.1

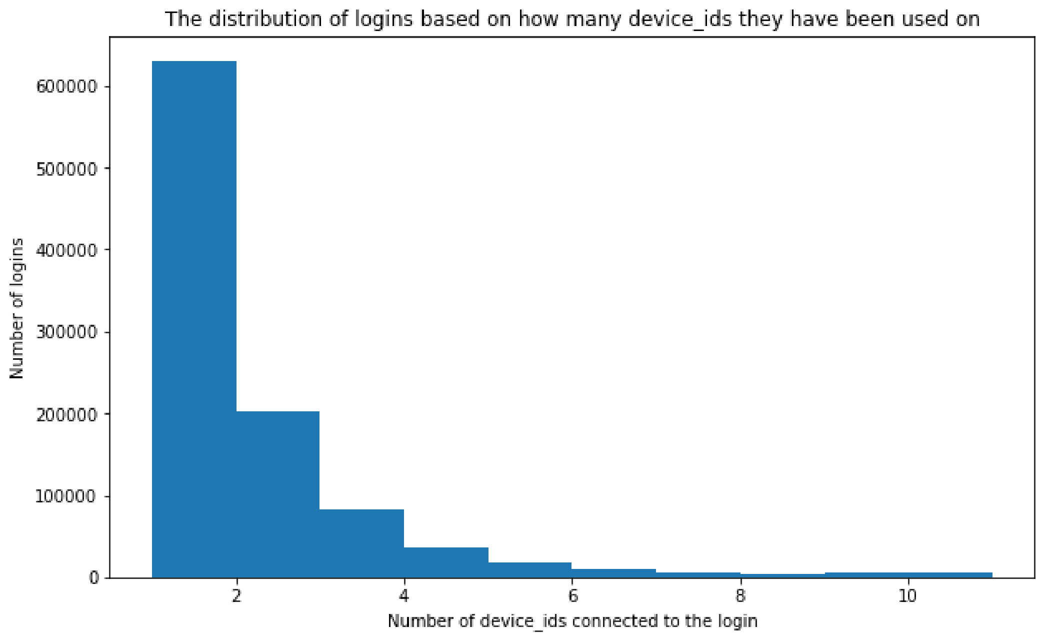


Figure 3.2: The distribution of logins based on how many devices they have been used on. The data is for one month of traffic on WSJ.

Obtaining the positive pairs is fairly straightforward: from every set of device_ids that share a login one simply takes all distinct pairs of elements. When constructing the set of negative pairs, the most straightforward way would be to take all possible pairs of device_ids where the first and the second element came from two different sets. However, such a negative set would be much larger than the positive set, and the classification problems are often more difficult with heavily unbalanced classes [31]. We therefore introduce another restriction on all pairs in the training set: they have to share at least one IP address, i.e. the two devices have to have been seen from the same IP address at least once. The same restriction was used during data preprocessing by several Drawbridge participants. After applying it, the ratio of classes in our data ranged from around 1:5 (five times as many negative instances as positive) to, surprisingly, 4:1 (four times more positive instances than negative). These ratios are not ideal, but much less heavily skewed than before the restriction. Table 3.1 shows how many instances are left to work with from one month after all the preprocessing steps.

It should be noted that these class ratios do not give us much information on how the classes are distributed in the data in general. All that can be concluded is that the ratios are like this among registered users, since they were the only ones considered in the training set. In practice, however, new instances would be taken from the set of all users, registered or unregistered, and the real distribution of classes might differ from the one in the training set.

Lastly, since we have all possible combinations of device types in the

	WSJ	WFP	ELP
Positive class size	110 934	13 858	3 181
Negative class size	590 607	31 500	757
Class ratio (positive:negaitve)	0.18	0.44	4.2

Table 3.1: Number of positive and negative pairs for all three sources for a one-month period.

training set, the two device_ids inside each pair are sorted by their device type. That allows to avoid a needless distinction between pairs where the first device is a phone and the second device is a computer, and pairs where the first device is a computer, and the second device is a phone.

3.3.1.2 IP-tables

Efficient checking whether two device_ids have been seen from a common IP address requires storing indices for what devices have been using which IP addresses. They had to be built prior to extracting device pairs, and even though they take up a lot of memory, it pays off, since those indices are later useful at the feature extraction step. The Figures 3.4 and 3.3 give some idea about the distribution of values in both indices. They have the same shapes as Figure 3.2: the most common value is 1, and then the frequency drops rapidly.

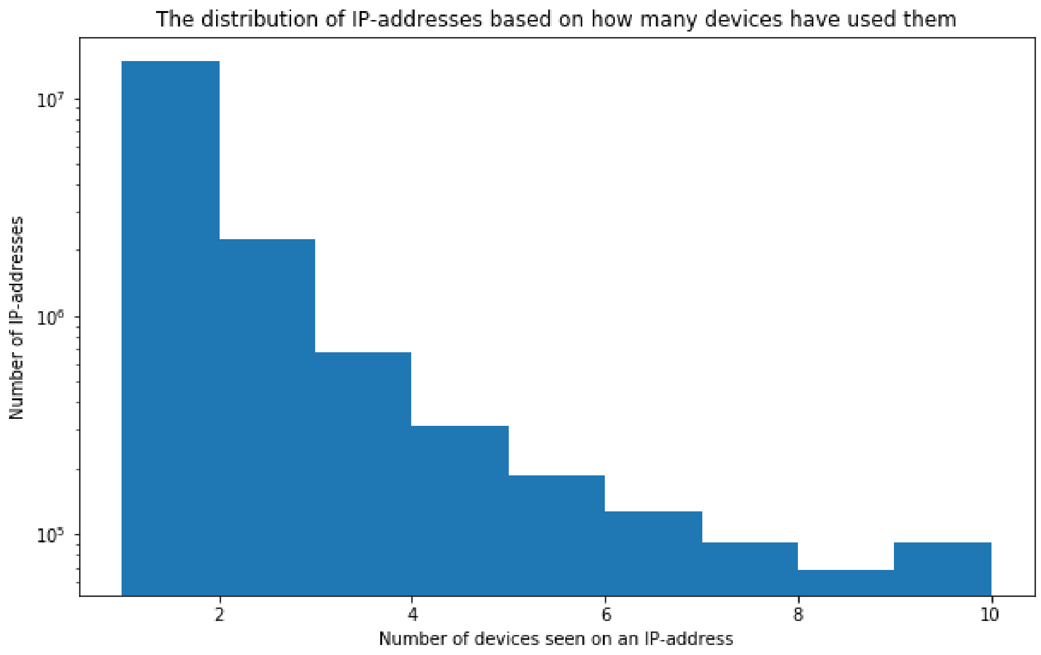


Figure 3.3: The distribution of devices based on how many IPs they have been seen on. The data is for one month of traffic on WSJ.

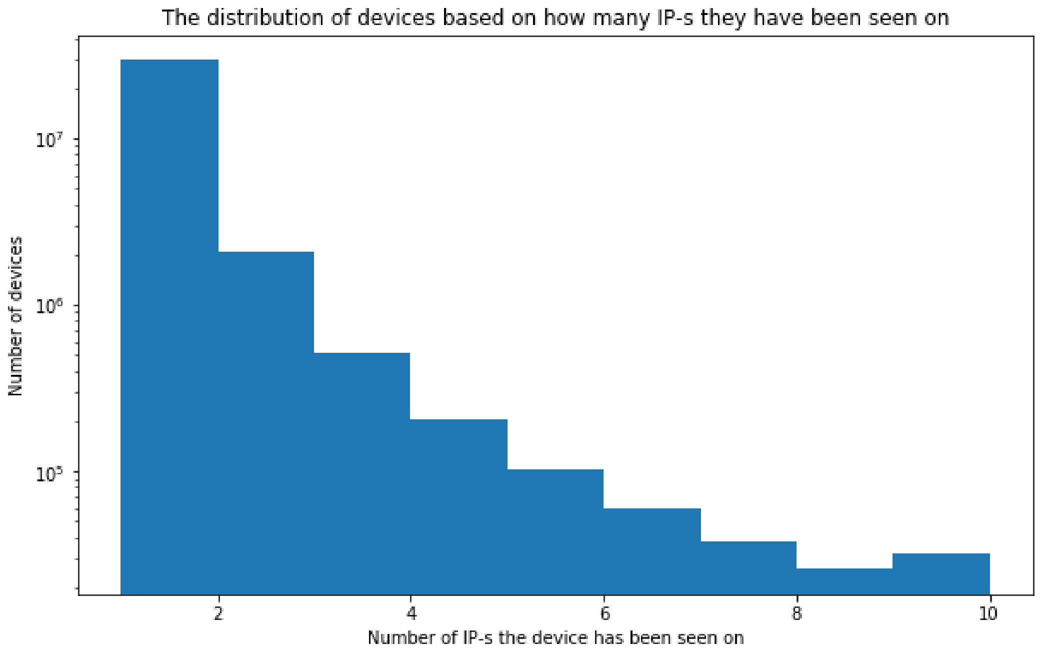


Figure 3.4: The distribution of IP addresses based on how many devices have used them. The data is for one month of traffic on WSJ.

Another preprocessing step involving IP addresses is getting rid of IPs used for big public networks, such as in the subway, or in a university. Such addresses do not provide reliable evidence that two devices using them are connected, and eliminating them helps narrow down the dataset. Strictly speaking, we do not know which IP addresses fall into that category, so we use a rough heuristic: all IP addresses with more than N devices seen on them are considered to be too uninformative, where N is typically an integer between 10 and 50. The numbers may seem high, but it is necessary to remember that device_ids are not device identifiers, but rather individual browser identifiers. If a household has two people, each of whom has two devices with two browsers, and a friend, who visited them during the processed time period, and they all accessed the website at least once, then the total number of device_ids recorded by the website from the household IP address is 9. If there are more people in the house, or if they sometimes clear cookies, or if it is not a household, but an office, that number would get higher. Later in this paper such IP addresses are sometimes referred to as "public", even though technically any address accessible over the Internet is public. In our case "public" stands for "used by the public". The primary goal of filtering out widely used IPs is to reduce the number of possible matches. One of the restrictions for including a pair into the training set was that the two devices share a common IP address. Therefore, if we remove very "popular" IPs from the data, we get fewer potential matches to check. For that purpose it is not essential to have a very exact mechanism of removing "public" addresses, and our heuristic is enough. Very few IPs are "public"—about 0.003% for WSJ, for instance,— but since they are frequent, they are responsible for a larger portion of traffic.

3.3.1.3 Environment

The amount of data from logged in users can vary, but for a popular website with heavy traffic it can be very large. In this project a month of raw log data from WSJ in json format took approximately 70 Gb of space. The necessary structures often don't fit into memory, and one of the ways of dealing with that is using parallel computing. All data processing and feature extracting was done on an Apache Spark cluster set up by Cxense.

Apache Spark operates on the so-called RDDs, or Resilient Distributed Datasets. RDDs are collections of items split into partitions that are processed in parallel on different machines. The specifics of how RDDs work lead to some challenges for efficient sorting and transforming the raw logs into a dataset, but they are not described in detail here.

3.3.2 Features

Choosing the right features when using machine learning is a a key success factor. One of the most popular recent trends is "deep learning", which normally stands for multi-layered neural networks. Those algorithms work best in combination with a very large number of features that occur naturally in the data, but are very low-level, i.e. the values of one separate feature do not correlate well with the class, rank or cluster of the instance. A typical example is image processing: with digital images as instances, the most natural way to represent them as numerical vectors is by using the color characteristics of the pixels. The dimensionality will be extremely high, and to the naked eye each individual feature will not give any useful information about the contents or the class of the picture. However, when such feature values are used as input to a multi-layer neural network, it can extract higher-level features from them after several layers. This process is called "representational learning", or "feature learning", and it is much more effective than manual feature engineering in tasks like image or speech processing.

Though representational learning has been proven useful in many areas and even has become a buzzword in recent years, the nature of our problem makes it difficult to use this type of learning. An instance in our case is represented by two sequences of events, often not very long ones, where each event has a relatively small (compared to image-processing magnitudes) number of different fields with non-homogenous values (string, numeric, and Boolean). In addition, there is evidence (see Section 2.5) that the class or the score of the instance correlates with some easily definable properties like similarity between sets of IP addresses.

That is why our approach to choosing features is almost a direct opposite of what is normally paired with deep learning. We have a relatively small number of manually crafted features that involved a time-consuming feature-extraction step. There were 25 features used in total in the experiments:

1. **Individual features.** The first thing that comes to mind when trying to answer the question whether the two devices are from the same user is their geographical location. Clearly, a pair where one device is from Australia, and the other one is from France is most probably not a match. Same logic is applied to the language of the browser. Another piece of information that might be important, though it is not obvious how exactly it correlates with the class labels, is the type of the device: mobile phone, computer, or tablet. These two kinds of features make up the first group. All these features are Boolean: six one-hot-encoded[2] features, for each device in the pair, for each of the three types of devices, plus four features that indicate whether the browser language, the region, the city and the device type are the same for both devices in the pair.

   A hypothetical example of a training instance consisting of two devices, *A* and *B*, is shown in Table 3.2. Table 3.3 provides additional information on all the IP addresses, which will be needed later for some types of features.

	Device *A*	Device *B*
Type	mobile	pc
Region	Manitoba	Manitoba
City	Winnipeg	Winnipeg
Language	English	English
IPs of events made by the device	1.1.1.1, 2.2.2.2, 1.1.1.1, 1.1.1.1, 1.1.1.1, 2.2.2.2, 3.3.3.3, 4.4.4.4	1.1.1.1, 1.1.1.1, 2.2.2.2, 3.3.3.3, 3.3.3.3, 5.5.5.5, 6.6.6.6, 7.7.7.7

Table 3.2: A hypothetical training instance. One device is a phone, the other one is a computer. They are both located in Winnipeg, Manitoba. Device *A* has 7 events from 4 different IP addresses, device *B* has 8 events from 6 different IP addresses.

   Concerning the location, it should be noted that it can vary from event to event, at least for mobile devices and tablets, so we record the most frequent location, trying to determine the ”home base” of the user. The nuances of the user's moving around are meant to be captured in the next group of features, which are based on IP addresses.

[2]One-hot encoding is a simple way of representing categorical features (i.e. non-numeric ones) as numbers. Consider a feature for the operating system of a phone. Let's assume it can take three values: ”iOS”, ”Android” or ”Windows Mobile”. It would not be correct to encode those three values as numbers 1, 2, 3, since that implies some sort of quantifiable relationship between the three values. Instead we can transform this feature into three numerical features, by the number of possible values of the original categorical feature. Each of the three new features answers the question ”does the instance have value *X*?”. The first feature takes value 1 for the devices on iOS, and 0 – for all others, the second and the third features will behave similarly, but with Android and Windows Mobile correspondingly.

	1.1.1.1	2.2.2.2	3.3.3.3	4.4.4.4	5.5.5.5	6.6.6.6	7.7.7.7
Number of devices seen from this IP	3	10 ("public")	2	1	1	4	3

Table 3.3: Number of devices seen from the IP addresses from the example in table 3.2. IP address 2.2.2.2 will be considered "public" in this example.

Feature name	Feature value
sameLanguage	1
sameRegion	1
sameCity	1
sameType	0
mobile1	1
mobile2	0
pc1	1
pc2	2
tablet1	0
tablet2	0

Table 3.4: Values of individual features for the example in table 3.2.

Table 3.4 shows what the values of the individual features will be for the example 3.2. For instance, "tablet1" has value 0, since the second device is a computer, not a tablet, and "sameLang" is 1, since both devices have the same browser language.

2. **IP-based features.** Each of the two devices in a pair has a set of IP addresses that it has accessed the website from. This next group of features includes information about how IP addresses were shared by the two devices, namely, several similarity measures between the two sets of IPs. One would expect that the more similar IP "footprints" are, the better the chances that the devices are a match.

   Let A and B be the sets of IP addresses for the first and the second device in the pair correspondingly, and let P be the set of "public" IPs (see Section 3.3.1.2 for what had been defined as a "public" IP for the experiments). Table 3.5 lists all the features with descriptions and values for the example 3.2.

3. **Event- and IP-based features.** Understandably, the range of numbers of common IP addresses is not very big: in one or two months period a user might not have moved around a lot, so a great majority of pairs, both negative and positive, have only 1 shared IP address, which was the minimum required to get into the training set. Figure 3.5 shows

Name	Description	Definition	Value for the example 3.2
number-of-IPs-1	Number of IPs in the first set.	$\lvert A\rvert$	4
number-of-IPs-2	Number of IPs in the second set.	$\lvert B\rvert$	6
common-Ips	Number of common IPs in the two sets.	$\lvert A \cap B\rvert$	3
common-IPs-non-public	Number of common IPs in the two sets, excluding the "public" IPs.	$\lvert A \cap B \setminus P\rvert$	2
common-IPs-weighed	Same as common-Ips, but weiged by how many other devices have been seen on this address, i.e. by the degree of publicity.	$\sum_{x \in A \cap B} \frac{1}{f(x)}$, where $f(x)$ is the number of device_ids seen from the IP address x	0.93
common-IPs-non-public-weighed	Same as common-Ips-Non-Public, but weighed by the degree of publicity.	$\sum_{x \in A \cap B \setminus P} \frac{1}{f(x)}$, where $f(x)$ is the number of device_ids seen from the IP address x	0.83
dice	Dice coefficient	$\frac{2\lvert A\cap B\rvert}{\lvert A\rvert+\lvert B\rvert}$	0.6
jaccard	Jaccard coefficient	$\frac{\lvert A\cap B\rvert}{\lvert A\cup B\rvert}$	3/7
overlap	Overlap coefficient	$\frac{\lvert A\cap B\rvert}{min(\lvert A\rvert,\lvert B\rvert)}$	0.75
common-IP-ratio1	Ratio of common IPs to total number of IPs for the first set.	$\frac{\lvert A\cap B\rvert}{\lvert A\rvert}$	0.75
common-IP-ratio2	Ratio of common IPs to total number of IPs for the second set.	$\frac{\lvert A\cap B\rvert}{\lvert B\rvert}$	0.5

Table 3.5: IP-based features.

how many IP addresses are normally shared between the two devices in the pair: in the positive class there are relatively more pairs with several common IPs, but most of the times it is only one.

However, there is a big difference between a pair where two devices have shared an IP once in the past month or regularly: intuitively the second case is more likely to be a match. This group of features is similar to the previous one, but instead of counting every IP once, they are counted proportionally to how many times the device was seen from them.

Let $E(X, d)$ be a function that takes a set of IP addresses X and a device_id d as arguments and returns a set of events made by that device_id from any IP address in X. Let d_1 and d_2 be the two devices in a pair, let A and B be the sets of IP addresses for d_1 and d_2 correspondingly, and let P be the set of "public" IPs, like in the previous example. Table 3.6 lists all features in this group.

Name	Description	Definition	Value for example in table 3.2
common-IP-events-1	Ratio of events from common IPs to total number of events from both devices.	$\frac{\lvert E(A\cap B,d_1)+E(A\cap B,d_2)\rvert}{\lvert E(A,d_1)\rvert+\lvert E(B,d_2)\rvert}$	$\frac{6+5}{7+8} = 0.73$
common-IP-events-non-public-1	Same as the previous feature, but without "public" IP addresses.	$\frac{\lvert E(A\cap B\setminus P,d_1)+E(A\cap B\setminus P,d_2)\rvert}{\lvert E(A\setminus P,d_1)\rvert+\lvert E(B\setminus P,d_2)\rvert}$	$\frac{4+4}{5+7} = 0.67$
common-IP-events-2	Sum of ratios of events from common IPs to total number of events over the two devices.	$\frac{\lvert E(A\cap B,d_1)\rvert}{\lvert E(A,d_1)\rvert} + \frac{\lvert E(A\cap B,d_2)\rvert}{\lvert E(B,d_2)\rvert}$	$\frac{6}{7} + \frac{5}{8} = 1.48$
common-IP-events-non-public-2	Same as the previous feature, but without "public" IP addresses.	$\frac{\lvert E(A\cap B\setminus P,d_1)\rvert}{\lvert E(A\setminus P,d_1)\rvert} + \frac{\lvert E(A\cap B\setminus P,d_2)\rvert}{\lvert E(B\setminus P,d_2)\rvert}$	$\frac{4}{5} + 47 = 1.37$

Table 3.6: Event- and IP-based features.

3.3.3 Differences in problem setup from the previous work

Like in all the articles from the Drawbridge competition, our data was not specifically tailored for cross-device tracking and required preprocessing

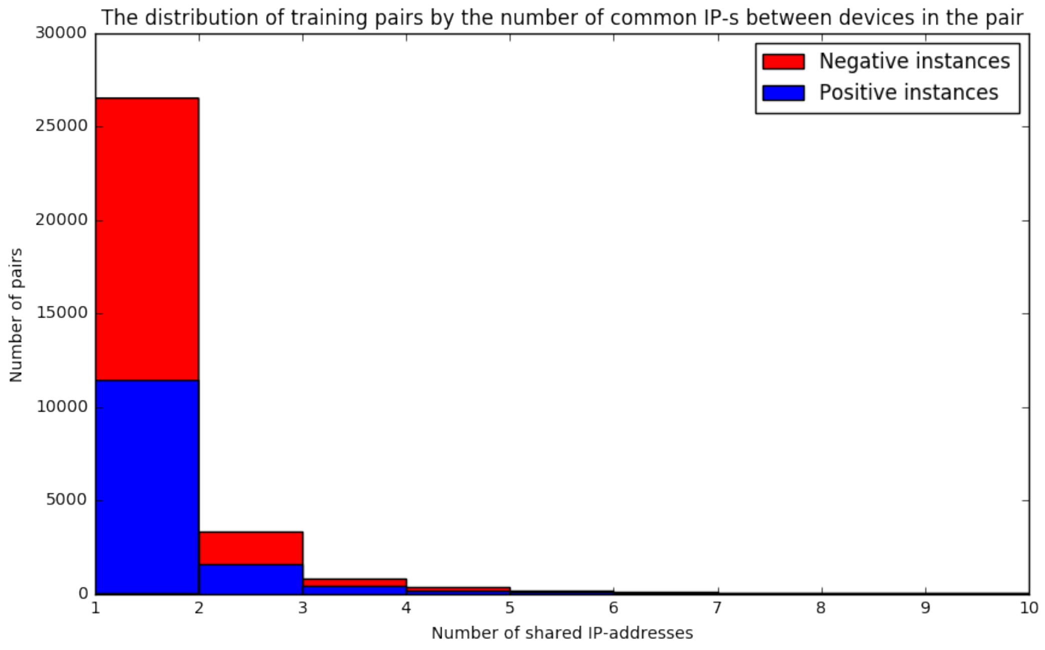


Figure 3.5: The distribution of training pairs by the number of common IPs between the devices in the pair. The data is for one month of traffic on WFP.

and feature engineering. The main differences from the previous works were as follows:

1. In the Drawbridge competition the two devices in the pair had to be a mobile device and a computer, while we did not put any restrictions on the type of the devices. In practice the coverage is better without this kind of limitations. We did try, however, to train several different classifiers for different types of pairs: phones + computers, phones + tablets, and tablets + computers (see Section 3.3.4.3.4), but did not retain this separation for most of the experiments.

2. Most properties in the Drawbridge tables were renamed (f. ex. "Anonymous_c2"), and only a few names were preserved, such as device type, OS, browser, etc. It was done to protect the anonymity of the users whose device information was included into the dataset, and also, presumably, to protect the knowledge about what features can be used for cross-device tracking. As a result, it is unclear what other properties of devices and IPs proved to be informative in the competition, apart from the basic ones, like country, OS, and browser.

3. Presumably, the Drawbridge dataset contained information across a number of different websites, which not only means more data, but also gives an opportunity to track a user more consistently. In our case, unfortunately, all identifiers were limited to one source, i.e. even if the same device accessed two or three of the sources that were used in the experiment, it would not be possible to track it.

4. The Drawbridge dataset contained tables with cooccurrences of devices, IPs, and their properties, while our data is in the form of traffic logs. It opens a possibility for using time stamps in feature engineering: for instance, the hours of the day when both devices are active, or some measure of similarity between two sequences of timestamps. Such features were not in the scope of this project.

3.3.4 Training a classifier

After the training set was complete, we used several algorithms to make sense of the data, like it was shown in Figure 2.2. The overall goal, as stated in Section 3.1, was to find matches that are certain enough to document them as devices belonging to the same person. The following sections describe fitting a classifier on the training set.

3.3.4.1 Environment

Most of the experiments were written with the help of the Python library scikit-learn [49]. Even though the datasets are initially very large and require parallel computing, they can be transformed into numpy arrays after feature extraction step is complete, and all instances are represented with numerical vectors. Numpy arrays are memory-efficient, and even the biggest dataset in the experiments could be trained on in memory. Apache Spark also has libraries for machine learning, but for several reasons scikit-learn is more practical: it is on average more flexible than Spark libraries, has a wider range of available algorithms, and works faster.

3.3.4.2 Baseline experiments

During supervised learning a dataset is divided into a training and a test set, which do not intersect. Then the training is performed on the training set, and the evaluation—on the test set. However, there is one more distinct set required when building a classifier: a development set. The system is usually not made in one go, and after the first attempt the researchers introduce various new ways to improve the performance. Some of such potentially improving methods are discussed later in this chapter. Each of them has to be evaluated to see if it really makes the system better. This can be done on the test set, but then the combination of the techniques chosen as the most improving ones is "tailored" to this test set. To get more reliable results and to understand whether these techniques really help, or whether they just happened to be optimal for that particular test set, we split the testing phase into two: first, the development phase, and then—the test phase itself. The development phase is evaluating all the versions of the classifier in search for the best one. The set that the evaluation is performed on is called the development set. Then, after we are finished with building the classifier, we can test it on completely fresh data that was not used previously and did not affect any choices of modules or parameters of the system. That is the actual testing phrase, and the results of the evaluation on the test set are more reliable that way. This is performed in Section 3.3.4.4.

Out of the dataset produced by the preprocessing step 20% of instances from each source are withheld for final testing. The remaining instances are split into a training set and a development set (see Table 3.7).

Training set size (%)	Development set size (%)	Test set (%)
60	20	20

Table 3.7: Relative sizes of the train set, development set, and test set.

At first we followed a simple pipeline for classification to obtain a baseline. We trained a classifier using off-the-shelf algorithms without any scaling, feature selection, class sampling, etc., then evaluated them on the development set from the same source it that they have been trained on. We do not try to transfer a model to a different source for evaluation until Section 3.3.4.5. Table 3.8 lists the properties of the first experiments.

Algorithm	Main parameters
Naive Bayes	Smoothing: Laplace
Logistic regression	Max. Iterations, if not converged: 100 Optimizer: liblinear
Random forest	Draw subsamples with replacement: True Tree split quality measure: Gini impurity Min. leaf size: 1 instance Min. number of instances to split a node: 2 Number of trees: 10
Support vector machine	Kernel coefficient: 1/(number of features) Kernel: radial basis function
Gradient tree boosting	Learning rate: 0.1 Loss function: deviance Max. tree depth: 3 Min. leaf size: 1 instance Min. number of instances to split a node: 2 Number of trees: 50

Table 3.8: Properties of the baseline experiments with different algorithms from scikit-learn.

Gradient tree boosting (GTB) [3] was used in most of the works in the Drawbridge competition, and turned out to often produce the best results in our experiments as well. Its concept is described in 2.5. Other algorithms were tested, such as logistic regression, Naive Bayes, random forest, support vector machines. We will not describe each algorithm in detail, only the core ideas that they are based on.

Naive Bayes method [7] is a classification algorithm that calculates conditional probabilities of each feature value depending on the class labels. It is simple, efficient, and widely used, but has a number of limitations: it operates under the assumptions that a) the classes are linearly separable b) the features are independent of each other and c) continuous feature values are distributed normally (in scikit-learn implementation). Logistic regression [1] is another probabilistic classifier, but, unlike Naïve Bayes it does not have the limitations b) and c). It models the probabilities by a logistic function to which it learns the parameters during training. Support vector machine [6] is a kernel-based classifier, i.e. it separates the classes by finding a decision boundary plane that is as far as possible from the instances closest to it. Finally, random forest [2] is an ensemble of predictors called decision trees. A decision tree attempts to split the dataset by some feature value in every node and have a label in every leaf, so that if an instance starts to "walk" down the tree from the root to a leaf, according to its feature values, it finishes in a leaf with the right label. In a random forest classifier each decision tree is trained on a random subsample of the dataset, and then the trees "vote" on the label of each new instance.

In Table 3.8 only the basic parameters are listed, and their values are the default values suggested for the given classifier with our type of dataset. These and other parameters, some of them implementation specific, were optimized later, during the tuning stage (see Section 3.3.4.3.6).

Tables 3.9 - 3.11 show the results of the baseline experiments for all three sources.

	Recall	Precision	F-score	Accuracy
Naive Bayes	0.83	0.47	0.60	0.79
Logistic Regression	0.65	0.78	0.71	0.90
Support vector machine	0.62	0.81	0.70	0.90
Random Forest	0.77	0.78	0.78	0.91
Gradient Tree Boosting	0.80	0.76	0.78	0.91

Table 3.9: The results of the baseline experiments on WSJ.

The results differ considerably for the three sources. Overall, it is clear that WFP is more problematic for all classifiers, while ELP is the easiest. A possible explanation is that WFP is a small regional newspaper, and the differences between positive and negative pairs are not that big. As for ELP, it is the only source with the reverse class size ratio, i.e. there are a lot more positive pairs than negative, so the fraction of false positives is lower because the negative class is small, and not necessarily because the classifier is very well fit. Recall is also good, possibly because classifiers often tend to favor the majority class (see more details in Section 3.3.4.3.1).

	Recall	Precision	F-score	Accuracy
Naive Bayes	0.84	0.51	0.64	0.71
Logistic Regression	0.61	0.62	0.61	0.77
Support vector machine	0.49	0.69	0.57	0.78
Random Forest	0.60	0.67	0.63	0.79
Gradient Tree Boosting	0.58	0.67	0.65	0.80

Table 3.10: The results of the baseline experiments on WFP.

	Recall	Precision	F-score	Accuracy
Naive Bayes	0.83	0.86	0.84	0.75
Logistic Regression	0.97	0.88	0.92	0.87
Support vector machine	0.98	0.87	0.92	0.87
Random Forest	0.96	0.91	0.93	0.89
Gradient Tree Boosting	0.96	0.91	0.93	0.89

Table 3.11: The results of the baseline experiments on ELP.

Gradient tree boosting produces the best results for all the sources, closely followed or matched by random forest, then come support vector machines and logistic regression. Naive Bayes performs worse, especially when it comes to precision, which is the most important metric for us, so it is dropped from further experiments.

3.3.4.3 Improving over the baseline

Training on the data from WSJ achieved results that are considerably better than random guessing, but are not nearly good enough to create links between devices based solely on this classifier. There are several common ways to improve classification results, as well as some less common ones which are suitable for the problem at hand:

- dealing with class imbalance;
- scaling;
- feature selection;
- separating instances by device types;
- calibration;
- tuning;
- moving the decision threshold;
- combining classifiers.

Throughout this section we provide tables with results to show how these methods affect the system performance. Each table is accompanied by a summary of whether the method helped, and if so, for which of the algorithms. In the cases when it wasn't clear whether the improvement was statistically significant, we performed the two–sample t–test to compare the values of some evaluation metric. If we are interested in precision, then the two samples are the two sets of positive predictions from the two versions of the model. Both samples contain values 1 (for true positives) and 0 (for false positives). Their sample means thus correspond to the two precisions. The null hypothesis is that both samples come from the same distribution, i.e. that the models are equally precise. Since we have large datasets, in most cases even a 1% increase is statistically significant.

The methods that helped are kept and applied in all the following experiments, i.e. every new section builds upon the previous methods, and the results shown in the tables are the best results so far. We assume that all the methods are fairly independent of each other, and that it does not matter what order they are applied in: for instance, if scaling and features selection separately prove to be useful, then applying them both at the same time is not going to decrease the performance of the classifier compared to when we include only one of them.

3.3.4.3.1 Dealing with class imbalance

As shown in Table 3.1, positive and negative classes in our problem are not of the same size. Even though what is usually referred to as "class imbalance" has lead to a great amount of research in the recent years, it is difficult to find a clear definition of what is the exact or, at least, an approximate ratio of classes, starting from which the dataset can be considered "imbalanced". This is a quote from an extensive review on the class imbalance problem [23, p. 1264]:

> "Technically speaking, any data set that exhibits an unequal distribution between its classes can be considered imbalanced. However, the common understanding in the community is that imbalanced data correspond to data sets exhibiting significant, and in some cases extreme imbalances. Specifically, this form of imbalance is referred to as a between-class imbalance; not uncommon are between class imbalances on the order of 100:1, 1000:1, and 10000:1, where in each case, one class severely outrepresents another".

Our case is hardly extreme, since in none of the experiments has the ratio of the classes been less than 1:10. However, even if applying the methods for dealing with class imbalance is not absolutely essential for us, it is still worth exploring what kind of effect they have on performance.

To understand why imbalanced classes might be problematic for a classifier, we have to remember that the overall goal for most classification algorithms is optimizing accuracy [42]. That means that for a highly skewed distribution of classes the classifier is inclined to be biased in favor of the majority class. Research suggests, however, that not all algorithms are equally prone to suffering from imbalanced data: support vector machines and other kernel-based methods are relatively unaffected by it [26]. The reason for that advantage is that Support Vector Machines rely only on the instances closest to the class boundary (the support vectors) and largely ignore the general distribution of instances. If a different algorithm is used, there are two main strategies to deal with class imbalance: sampling and cost-sensitive learning.

Sampling is artificially making the classes equal in size by either duplicating some instances in the minority class (over-sampling) or downsizing the majority class (under-sampling). Both have their disadvantages: oversampling can lead to overfitting based on some attributes that now happen in the dataset more often, and undersampling can remove valuable information about the majority class [31]. In [18] the authors state that undersampling produces superior improvements in classifier performance, plus we have a relatively big number of data points for both classes (at least several thousand, except for the negative class in ELP), so we chose to focus on undersampling instead of oversampling. Cost-sensitive learning is trying to take into consideration different costs of mistakes of different types while training, and it has not been applied here.

Tables 3.12 - 3.14 show what happens when a classifier trained on a dataset with 1:1 class ratio is evaluated on the development set with the natural class ratio.

	Recall	Precision	F-score	Accuracy
Logistic regression	0.89	0.55	0.68	0.84
Support vector machine	0.91	0.54	0.68	0.83
Random forest	0.91	0.68	0.78	0.90
Gradient tree boosting	0.92	0.67	0.77	0.90

Table 3.12: The results of testing on the normal class proportions after training on an undersampled training set with equal size classes (WSJ).

First, the results are very different from the baseline, even for SVM, which was supposed to be relatively unaffected by disbalanced classes.

	Recall	Precision	F-score	Accuracy
Logistic regression	0.81	0.55	0.66	0.75
Support vector machine	0.86	0.54	0.66	0.73
Random forest	0.82	0.57	0.67	0.76
Gradient tree boosting	0.87	0.56	0.68	0.75

Table 3.13: The results of testing on the normal class proportions after training on an undersampled training set with equal size classes (WFP).

	Recall	Precision	F-score	Accuracy
Logistic regression	0.86	0.89	0.88	0.80
Support vector machine	0.86	0.89	0.87	0.80
Random forest	0.85	0.93	0.88	0.82
Gradient tree boosting	0.87	0.93	0.90	0.84

Table 3.14: The results of testing on the normal class proportions after training on an undersampled training set with equal size classes (ELP).

Second, the classifiers predictably produce better recall, but precision drops significantly. It happens because a sampling strategy changes the class ratio in the training set, but the class ratio in the development set remains the same. A lot of algorithms take into account the class distribution, either explicitly, like logistic regression, or implicitly, like tree-based classifiers. If before the sampling a classifier might be more inclined to produce too many false negatives, after the sampling there appears the opposite problem: the classifier predicts too many instances as positive, because it overestimates how probable a positive instance is. With ELP it is the other way round: the positive class has been downsized during training, so now the classifiers underestimate the probability of the positive class, which leads to a drop in recall.

Good recall values would have been a positive thing if we had a radical class imbalance and the positive class had been ignored before undersampling. However, that was not our case: recall in the baseline experiments was not far behind precision. Moreover, high precision is more valuable for us than high recall. Therefore, the undesirable rise in false positives after undersampling clearly outweighs the gain in recall. This step is dropped and is not used in any of the following sections.

3.3.4.3.2 Scaling

While it is not always essential, some algorithms (for example, certain types of SVM) might not work correctly if the features have a widely different mean and standard deviation [24]. It is therefore common practice to scale the data beforehand, so that every feature has a zero mean and a unit standard deviation. To achieve that, first the standard deviation and mean are estimated from all the values of a given feature, and then every value is decreased by the mean and divided by the standard deviation:

$$x_{scaled} = \frac{x - \overline{x}}{\sigma_x}$$

where x is the random variable that takes the feature values, $\overline{x}$ is the sample mean, and σ_x is the sample standard deviation, both estimated from the training set.

There are other scaling techniques, like mapping all the values of a feature to an interval $[0, 1]$. Most of our features are Boolean or some sort of similarity measure between zero and one, which means this type of scaling would not make much difference.

The hypothesis is that support vector machines get better with scaling [9], while tree-based methods are unaffected due to how they are built: each decision in a tree answers a question ”is feature_value(x) > threshold?”, so the scale those values are located on is not important. Tables 3.15 - 3.17 show the performance for algorithms with standard scaling. Our results match the expectations: SVM does better with scaling, while gradient tree boosting and random forest are not different from the baseline. For logistic regression the improvement is not significant, but with stochastic average gradient descent, that is being used for logistic regression in scikit-learn, scaling is important for fast convergence [9].

	Recall	Precision	F-score	Accuracy
Logistic regression	0.65	0.78	0.71	0.90
Support vector machine	0.73	0.79	0.76	0.91
Random forest	0.77	0.78	0.78	0.91
Gradient tree boosting	0.81	0.78	0.79	0.92

Table 3.15: The results after applying standard scaling (WSJ).

	Recall	Precision	F-score	Accuracy
Logistic regression	0.62	0.62	0.62	0.77
Support vector machine	0.55	0.68	0.61	0.79
Random forest	0.58	0.66	0.62	0.79
Gradient tree boosting	0.62	0.67	0.65	0.80

Table 3.16: The results after applying standard scaling (WFP).

	Recall	Precision	F-score	Accuracy
Logistic regression	0.98	0.88	0.93	0.87
Support vector machine	0.98	0.88	0.93	0.88
Random forest	0.95	0.91	0.93	0.89
Gradient tree boosting	0.96	0.91	0.93	0.89

Table 3.17: The results after applying standard scaling (ELP).

Scaling is kept for support vector machines and logistic regression in the following experiments and dropped for the other two algorithms.

3.3.4.3.3 Feature selection

Feature selection is reducing the dimensionality of the input space by eliminating some of the features. It is often used for problems with very high dimensionality: for example, in text processing, where features are words, there can be hundreds of thousands of features, depending on the dataset, and that might cause problems for the classifier. First of all, it is more time- and memory-consuming to work with very long input vectors, and second—not all words are important for classification, so reducing the number of features to several thousand eliminates a lot of noise and usually improves the results. There is also the danger of overfitting: if the number of instances is not very big, compared to the number of features, then the classifier might learn a very complex decision boundary that separates the classes perfectly in this particular dataset, but doesn't work too well for new instances from the same general sample (see Section 3.3.4.5.4 for more details). Another way to deal with numerous raw features is to use neural networks (see Section 3.3.2).

Scikit-learn provides several ways to select the best features [4]. The most simple method is variance thresholding, i.e. removing all features whose variance is too low. The threshold in this case is a fixed value specified manually. In case we want to eliminate features by comparing them to other features instead of a fixed threshold, we can use a selector that only leaves the best N or N% of the features by performing a statistical test of independence: for instance, the chi-square test. The features that are most likely to be independent of the class are then removed.

Cross-device tracking is a new field, and all the features used in this project were based more on an educated guess than on a solid body of related research, so there is no way of estimating in advance how useful the features are. However, there are only 25 of them, and they involve a higher level of feature engineering than the "raw" features in text or image processing. Overfitting is therefore not a problem, and trying to remove any of the features is highly unlikely to improve performance.

It is also important to remember that not all features in our set are independant: those that are in the same group often correlate (same-region and same-city, IP-dice and IP-Jaccard). Figure 3.6 demonstrates the correlation between two highly informative features and, quite possibly, the most correlated ones, based on chi-square test. Both of them are based on the IP history of the devices, with only a slight difference in averaging the values from the two devices in the pair. Strong correlations mean, among other things, that features in the same group are likely to have similar degree of usefulness, so if an N-best selector picks one feature from a group, it will most probably pick at least several more from the same group. But since they are heavily dependant, it does not necessarily mean picking them together will contribute much more to the performance than picking just one of them. In fact, none of the reductions of the feature set proved to be beneficial, so feature selection is not used further.

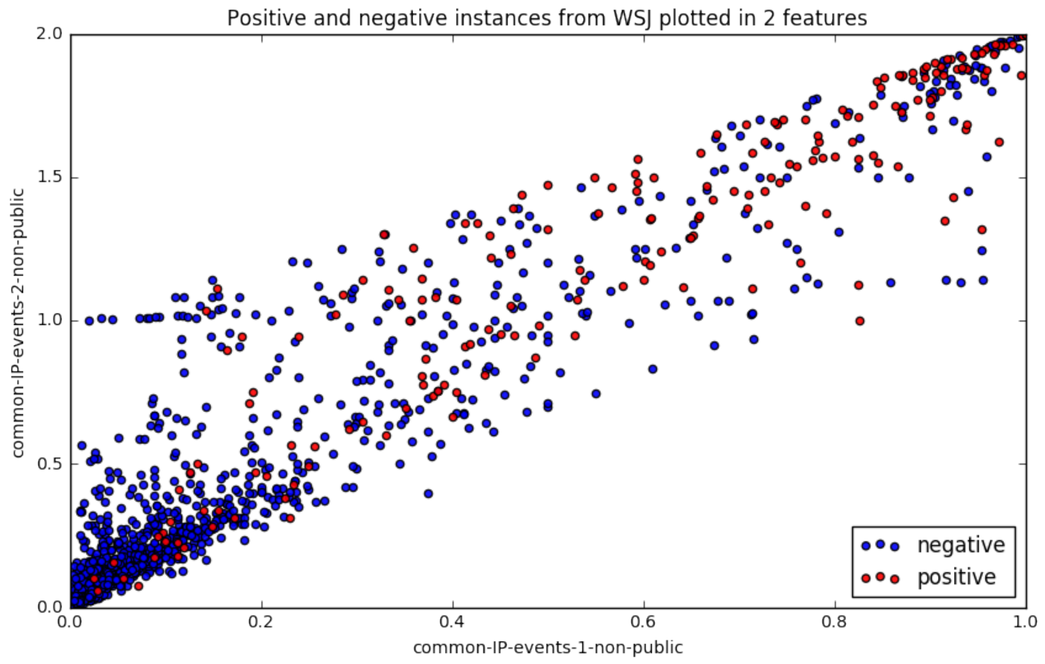


Figure 3.6: A random subset of WSJ dataset plotted in two features estimated as the most useful ones by the chi-square criterion. Pearson correlation coefficient = 0.97

3.3.4.3.4 Separating instances by device types

One of the limitations in the Drawbridge competition that has been lifted in this work was the types of the devices in the dataset. We have three types of devices: tablet, pc and mobile; all combinations of types have been included into training set. It would not be surprising if different pair types formed distinct and somewhat dissimilar clusters inside the positive class, which would make it less homogenous and, possibly, harder to learn. To see if reintroducing this limitation is useful, we tried training and evaluating separately on different types of pairs.

There are six possible types of pairs: tablet+pc, pc+mobile, mobile+tablet, mobile+mobile, tablet+tablet and pc+pc. Table 3.18 shows how many pairs of each type there are for two of the sources, as well as the class ratios for each subset. ELP was not included, because it had the smallest dataset, so after splitting it into six parts it was problematic to get a training and a development set big enough to produce good results and evaluate them reliably.

The biggest class contains pairs of computers, which does not seem to fit the intuition of how people use the same login from their own private devices. It could stem from a lot of users clearing cookies or using different browsers. It might also be a sign that even after filtering out logins used by too many different devices during preprocessing, we still did not completely eliminate corporate accounts.

Class ratios are never extreme: the biggest disbalance is for the pairs

	Number of instances		Class ratio (positive : negative)	
	WSJ	WFP	WSJ	WFP
tablet+tablet	23221	4269	1.67	1.06
tablet+mobile	48407	4108	0.15	0.23
tablet+pc	71561	7513	0.24	0.25
mobile+mobile	143855	4974	0.09	0.35
mobile+pc	157263	9121	0.15	0.30
pc+pc	397234	15373	0.30	0.63

Table 3.18: Training set divided by the types of devices in the pairs.

of phones in the WSJ dataset, where there are approximately ten times as many negative instances as positive. Other than that the distributions are no more skewed than before separating by device types.

Tables 3.19 - 3.20 show the results of training and evaluating on the pairs of the same type.

	Tablet + phone				Phone + PC				PC + tablet			
	R	P	F	A	R	P	F	A	R	P	F	A
LR	0.69	0.83	0.76	0.94	0.56	0.75	0.64	0.91	0.73	0.81	0.77	0.92
SVM	0.75	0.84	0.79	0.95	0.63	0.79	0.70	0.93	0.82	0.84	0.83	0.94
RF	0.78	0.85	0.81	0.95	0.72	0.79	0.75	0.93	0.82	0.85	0.84	0.94
GBT	0.79	0.83	0.81	0.95	0.75	0.78	0.76	0.94	0.85	0.84	0.84	0.94
	Tablet+tablet				Phone + phone				PC + PC			
	R	P	F	A	R	P	F	A	R	P	F	A
LR	0.94	0.93	0.94	0.92	0.70	0.86	0.77	0.96	0.57	0.77	0.66	0.86
SVM	0.96	0.93	0.95	0.93	0.70	0.88	0.78	0.97	0.66	0.76	0.71	0.87
RF	0.96	0.94	0.95	0.94	0.77	0.89	0.83	0.97	0.79	0.73	0.76	0.88
GBT	0.97	0.94	0.95	0.94	0.78	0.88	0.83	0.97	0.80	0.73	0.76	0.88

Table 3.19: Results for training and testing on data from WSJ, separately for each device type. R is recall, P is precision, F is F-score, A is accuracy. LR is logistic regression, SVM is support vector machine, RF is random forest, GBT is gradient tree boosting.

It appears that some types are better separable than others. Some of the differences can be explained by the fact that class ratios are different: pairs of tablets in both classes are more likely to be a match, so the scores are higher for the positive class. However, the peculiarly bad results for the upper half of Table 3.20 are caused by something other than class imbalance, because the class ratio is normal for all three cases:

	Tablet + phone				Phone + PC				PC + tablet			
	R	P	F	A	R	P	F	A	R	P	F	A
LR	0.03	0.32	0.05	0.79	0.21	0.51	0.30	0.76	0.1	0.45	0.2	0.79
SVM	0.05	0.58	0.08	0.80	0.15	0.55	0.24	0.76	0	0	0	0.79
RF	0.30	0.54	0.38	0.81	0.31	0.56	0.40	0.77	0.56	0.32	0.41	0.80
GBT	0.31	0.55	0.39	0.81	0.34	0.54	0.42	0.77	0.35	0.58	0.44	0.81
	Tablet+tablet				Phone + phone				PC + PC			
	R	P	F	A	R	P	F	A	R	P	F	A
LR	0.82	0.73	0.77	0.75	0.55	0.69	0.61	0.81	0.77	0.68	0.72	0.77
SVM	0.84	0.74	0.79	0.76	0.63	0.70	0.67	0.83	0.71	0.70	0.70	0.77
RF	0.80	0.78	0.79	0.78	0.56	0.73	0.64	0.83	0.75	0.71	0.73	0.79
GBT	0.82	0.78	0.80	0.78	0.62	0.74	0.67	0.84	0.75	0.71	0.73	0.79

Table 3.20: Results for training and testing on data from WFP, separately for each device type. R is recall, P is precision, F is F-score, A is accuracy. LR is logistic regression, SVM is support vector machine, RF is random forest, GBT is gradient tree boosting.

approximately 1:3 to 1:4. The most remarkable thing is that classifiers performed well on pairs of mobile phones, despite the class ratio for WSJ for this type being lower than usual.

Overall this separation is not particularly useful: it almost does not affect WSJ and lowers performance on WFP, which is the most problematic dataset. Even though some types seem better separable, there is not enough data to state with certainty that some types of pairs are universally easier to classify, regardless of the source. This method might be useful at some point in future research, on more extensive and better filtered data, but it is not applied in the rest of this project.

3.3.4.3.5 Calibration

Class labels are normally not the only output of a classifier: they are usually decided based on a numerical score, that roughly stands for how certain the label is. Some classifiers are probability-based (Logistic Regression), others operate in terms of weights or scores (support vector machines). It would not be correct to interpret such scores directly as probabilities or levels of confidence, even if they are scaled to lie in the interval $[0, 1]$, because SVM, gradient boosting and other maximum-margin methods (i.e. methods that score instances by the distance from the class boundary) are not naturally well-calibrated [5]. That means that even though the class boundary produced by the classifier can be good (in other words, the classifier discriminates well), the exact scores it assigns to

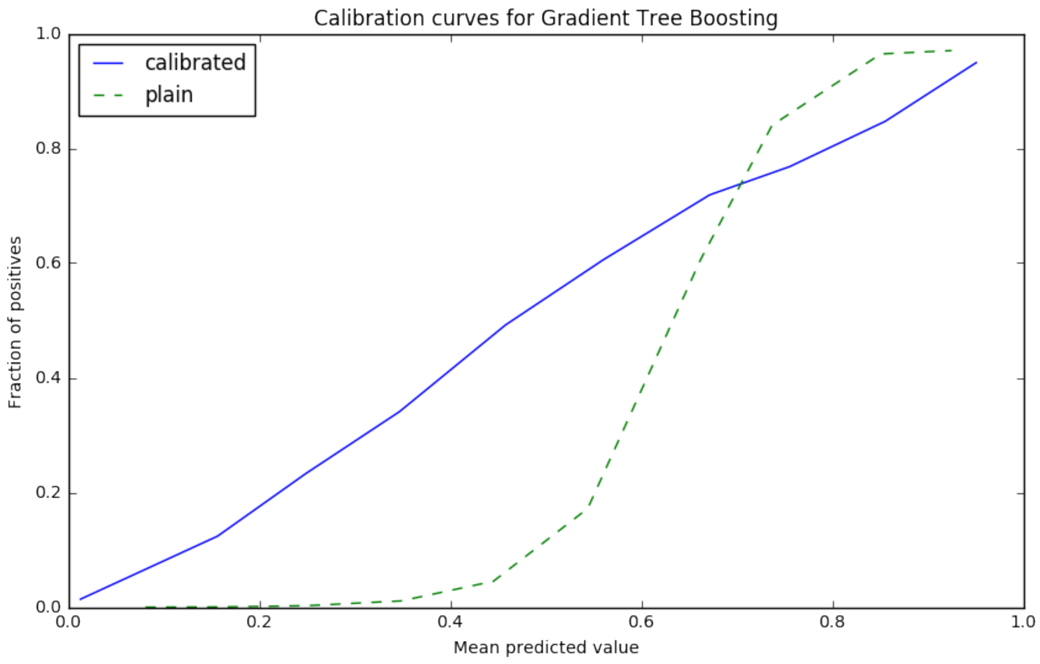


Figure 3.7: Calibration curve for gradient tree boosting before and after calibration.

instances do not correspond to the probabilities of their labels. The way to deal with it, in case it is necessary to operate with immediate probabilities, is to calibrate the classifier on new data, unseen during training.

Visually calibration can be represented by a calibration curve: it plots the fraction of true positives against probabilities, i.e. for every probability interval it shows how many positive predictions with probabilities within that interval were actually correct. In Figure 3.7 the more straight line shows how a well-calibrated classifier works: among the predictions with probability around $N\%$, the fraction of true positives is about $N\%$. A gradient tree boosting model without calibration produces a sigmoid-shaped curve, which is typical for uncalibrated models [37].

Scikit-learn provides an option for most classifiers to transform their scores into probability estimates. In order for threshold–moving to work correctly (Section 3.3.4.3.7), we have to check that all the models are properly calibrated. If they are not, scikit-learn allows explicitly adding calibration by one of two methods: Platt scaling and isotonic regression. Platt scaling was developed for turning the output scores of the SVMs into real probabilities [41]. Isotonic regression is a more general method first applied to Naïve bayes and decision trees [59]. Both methods have been thoroughly tested in combination with various machine learning algorithms in [37].

The classifiers that have scored highest so far in the experiments are tree-based: random forest classifier and gradient tree boosting. For random forest calibration problems usually appear close to 0 and 1. According to [37], the difficulties near 0 and 1 happen because in order for an instance to get 0 score, all the trees that are being averaged have to produce the 0 score. That does not happen often because of the variance of the trees. The authors also conclude that random forest is still relatively well-calibrated, unlike maximum-margin methods, where the mass of predictions is even stronger shifted from 0 and 1 towards the middle of the interval. We leave the default scikit-learn calibration in place for both tree-based algorithms, because their probability estimates look fine on the calibration curves (see Figures 3.7 and 3.8). We will, however, explicitly apply isotonic regression later, in Section 3.3.4.5.4, when due to a dataset shift the default calibration method is not enough.

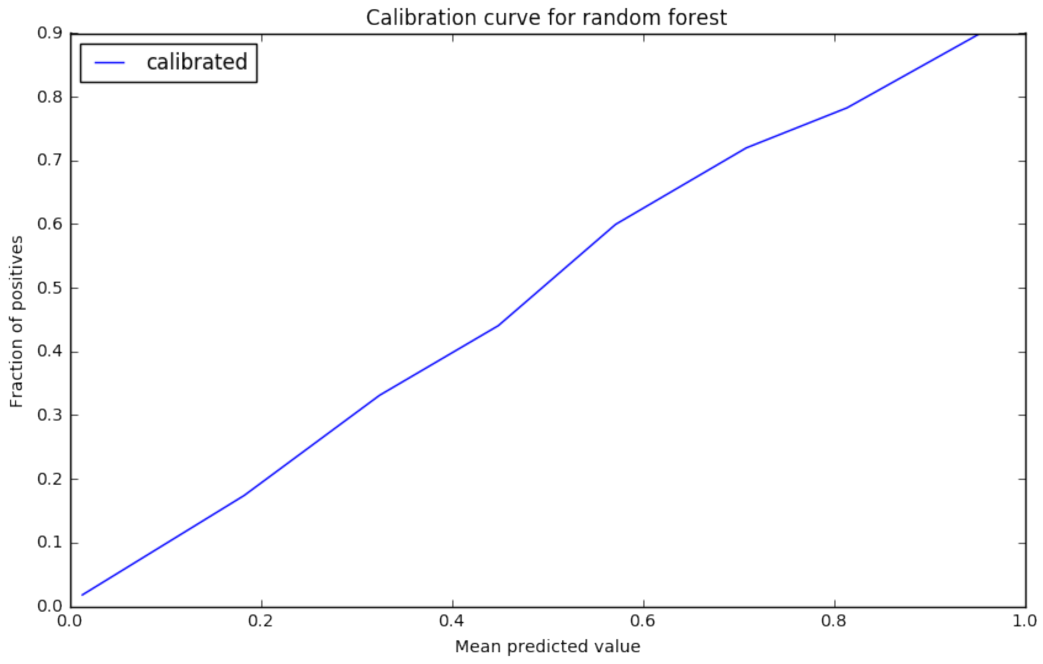


Figure 3.8: Calibration curve for the calibrated random forest classifier.

As for support vector machines, Platt scaling, used in scikit-learn by default, is clearly inferior to isotonic regression in our case (see Figure 3.9). For all the following experiments we switch to isotonic regression for calibrating support vector machines.

3.3.4.3.6 Tuning

Every algorithm has a number of hyper-parameters like, for instance, the function to optimize, the number of iterations (for iterative algorithms), the depth of trees (for tree-based classifiers), and so on. The default values, provided for the hyper-parameters by the library they are taken from, are not necessarily the best for every given dataset, so the hyper-parameters should be optimized, or tuned. The most straight-forward way to do that is to check how good the results are for every combination of parameter values. To do it faster and avoid overfitting, each combination of parameter

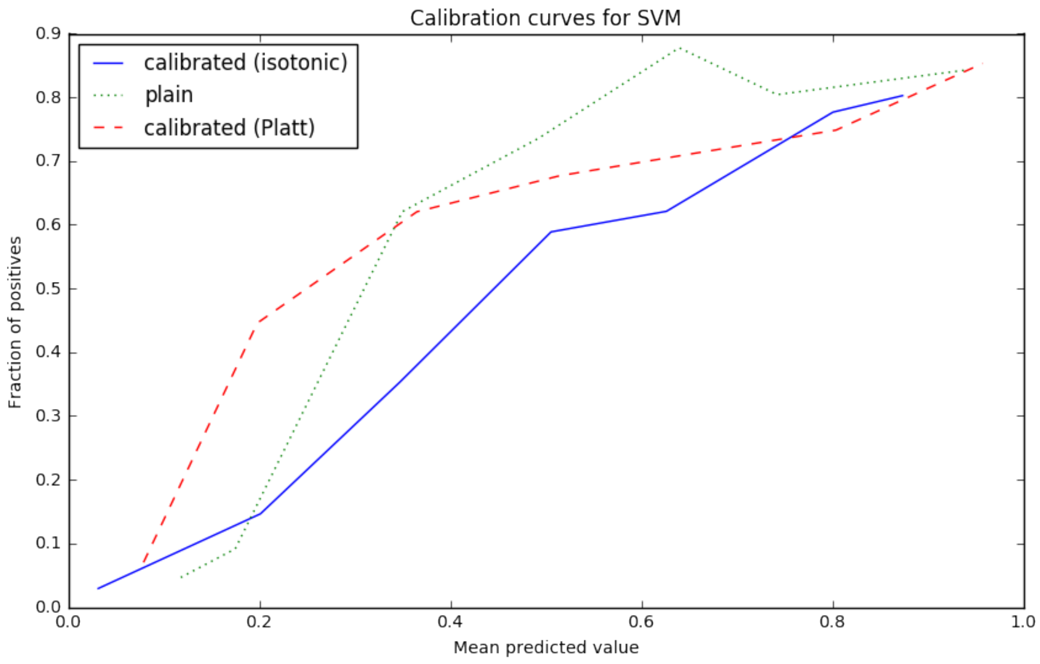


Figure 3.9: Calibration curves for SVM with different calibration methods.

values is tried out on a relatively small subsample of the training set, like it is done in cross-validation. In case a hyper-parameter has continuous values instead of categorical or integer, the interval in which the values lie is divided into several equal parts, and the points at the beginnings of all parts are taken as possible values. For instance, to tune a hyper-parameter that lies between one and zero we can take values 0, 0.1, 0.2, 0.3, . . . 0.9, 1.0. In scikit-learn this type of tuning is implemented in the class GridSearchCV [8]. For the evaluation metric to optimize we chose precision, and not accuracy, since it fits our goal better. Table 3.21 shows how the main parameters of the classifiers changed, compared to their values in the baseline experiment for WSJ.

The optimized parameters were estimated through 3-fold cross-

	Optimized parameters
Logistic regression	-
Support vector machine	Kernel coefficient: 1
Random forest	Tree split quality measure: entropy Number of trees: 10
Gradient tree boosting	Loss function: exponential Max. tree depth: 10 Number of iterations: 200

Table 3.21: Basic classifier parameters after tuning for WSJ.

validation on the training set. In some cases moving the value of a parameter further continues to be beneficial, but we have to stop because of time concerns. That is the situation with maximum tree depth for gradient boosting and some other parameters. In the following Sections all classifiers are tuned, if not written otherwise.

It should be noted that tuning hyper-parameters is not only to be used with the classifier itself: it is applicable to other modules of the system as well, such as feature selection, where we need an optimal threshold or the number of features to keep. Those tuning procedures are done in every case when it is necessary, but for conciseness and better readability the intermediate results before tuning are not explicitly mentioned here for any other modules than the classifier itself.

3.3.4.3.7 Moving the decision threshold

To improve the results of the classifier we can make use of the scores of the instances. As was mentioned in Section 3.3.4.3.5, most algorithms output scores, which either roughly correspond to probabilities of assigned labels, or can be calibrated to do so.

Normally the threshold for assigning an instance to a class is at 0.5 (like it was in all our previous experiments), but after the classifying is done, that threshold can be adjusted manually to be, for example, 0.9, to extract only the instances that were classified as positive with >0.9 certainty. It influences the trade-off between precision and recall, which is illustrated by Figure 3.10: every threshold corresponds to some recall and precision values, and as precision increases, recall drops. Our goal is therefore to find a point on the horizontal axis where precision is adequately high, but recall is not yet so low that the system would almost never make positive predictions.

Tables 3.22 - 3.24 provide the results after adjusting the probability threshold for all the sources.

Overall this method is very helpful: in most cases at least one of the classifiers—typically, gradient boosting—is well fit and calibrated enough so that the higher the threshold, the fewer false positives, and we can reach a very good precision, while keeping acceptable recall: for instance, using a gradient boosting classifier on WSJ with decision threshold 0.95 we successfully identified approximately every sixths match in the development set, and only 4% of those pairs were false positives (lowest row in Table 3.22. WFP is somewhat more difficult: the 0.95-threshold corresponds to a slightly lower precision (0.95) and very low recall (0.05). However, it is a significant improvement over the previous results on this dataset.

Numbers for ELP are technically high, but the constant value of precision at 0.91, while the threshold is being moved up 15%, shows that even though there are few false positives, they get a very high score, and such misclassifications are hard to eliminate.

Moving the decision threshold is kept in all the following experiments

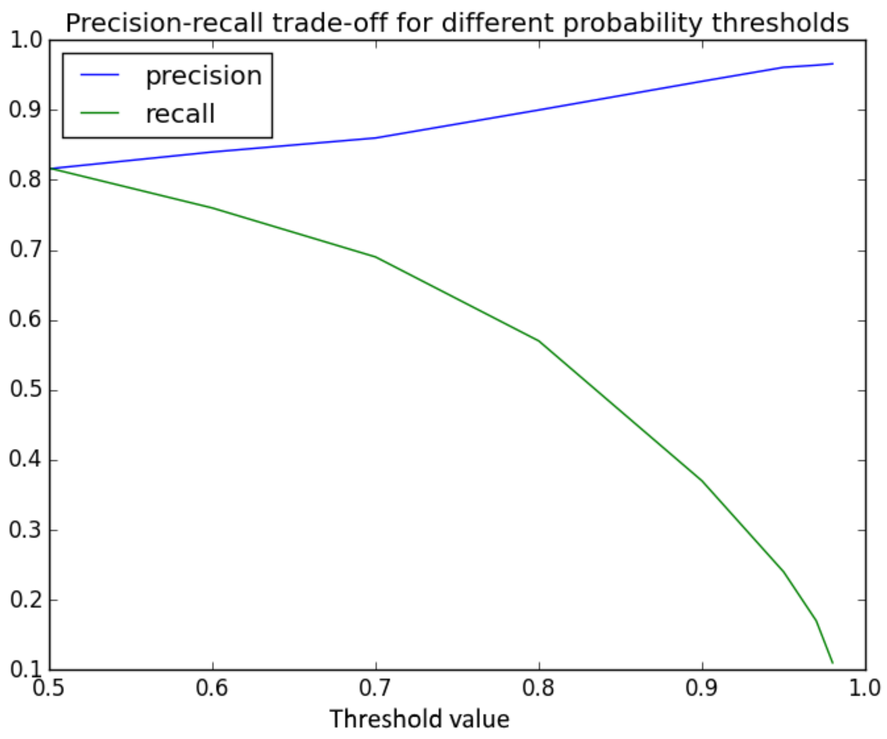


Figure 3.10: An example of precision and recall curves, depending on different decision thresholds.

	Threshold	Recall	Precision	F-score	Accuracy
Logistic regression	0.8	0.32	0.87	0.47	0.86
	0.9	0.17	0.90	0.29	0.84
	0.95	0.07	0.91	0.13	0.82
SVM	0.8	0.33	0.87	0.48	0.86
	0.9	0.15	0.90	0.26	0.84
	0.95	0.01	0.95	0.02	0.81
Random forest	0.8	0.45	0.87	0.59	0.88
	0.9	0.23	0.92	0.37	0.85
	0.95	0.17	0.93	0.29	0.84
Gradient tree boosting	0.8	0.44	0.88	0.79	0.92
	0.9	0.23	0.94	0.37	0.85
	0.95	0.15	0.96	0.25	0.84

Table 3.22: Evaluating with different decision thresholds (WSJ).

	Threshold	Recall	Precision	F-score	Accuracy
Logistic regression	0.8	0.01	0.71	0.02	0.70
	0.9	-	-	-	-
	0.95	-	-	-	-
SVM	0.8	0.01	0.75	0.02	0.70
	0.9	0.001	1.0	0.002	0.70
	0.95	0.0003	1.0	0.0007	0.70
Random forest	0.8	0.22	0.85	0.36	0.76
	0.9	0.10	0.91	0.18	0.73
	0.95	0.06	0.91	0.11	0.72
Gradient tree boosting	0.8	0.25	0.84	0.39	0.76
	0.9	0.09	0.91	0.16	0.72
	0.95	0.05	0.94	0.09	0.71

Table 3.23: Evaluating with different decision thresholds (WFP).

	Threshold	Recall	Precision	F-score	Accuracy
Logistic regression	0.8	0.84	0.90	0.87	0.79
	0.9	0.50	0.94	0.65	0.57
	0.95	0.17	0.95	0.29	0.32
SVM	0.8	0.93	0.89	0.90	0.85
	0.9	0.36	0.94	0.53	0.47
	0.95	0.05	1.0	0.09	0.23
Random forest	0.8	0.88	0.93	0.90	0.84
	0.9	0.70	0.93	0.80	0.72
	0.95	0.51	0.93	0.66	0.57
Gradient tree boosting	0.8	0.95	0.91	0.93	0.89
	0.9	0.93	0.91	0.92	0.87
	0.95	0.83	0.91	0.87	0.80

Table 3.24: Evaluating with different decision thresholds (ELP).

	Recall	Precision	F-score	Accuracy
Random forest + Gradient tree boosting + SVM (average score)	0.80	0.79	0.79	0.92
Random forest + Gradient tree boosting + SVM (AND)	0.72	0.87	0.79	0.93
Random forest + Gradient tree boosting + SVM (voting)	0.79	0.79	0.79	0.92

Table 3.25: Evaluating three strategies for combining classifiers (WSJ).

in all the cases when it is possible, i.e. when doing so provides a good precision-recall trade-off point.

3.3.4.3.8 Combining classifiers

One can train several classifiers and somehow combine their predictions to get better results. Before classifying instances in two classes every algorithms assigns some score to the instance. This score can be a distance to something the decision boundary (as in SVM), or an actual probability (Naïve Bayes), or some other type of measuring class membership. The scores can be averaged for several independently trained classifiers, or the classifiers can "vote" on the class label to assign to each instance. A Boolean function can also determine the output label: e.g., an instance can be assigned to the positive class if and only if all the classifiers in the ensemble predicted it to be positive (AND-function).

There also exist more advanced ways of combining classifiers, for example, through ROC convex hull (ROCCH-hybrid classifier). An example of an ROC curve is shown in Section 2.4. In our experiments we try to move the decision threshold after the model is already trained. The idea of ROCCH-hybrid is to change models depending on what threshold we want to use. That is, we can keep a classifier for each threshold, according to which point of the ROC convex hull corresponds to the optimal ROC angle (see [43] for more details). However, later one of the authors of the method reported that it can be outperformed by a simple AND/OR combination of classifiers [19], so we did not try out the ROCCH approach in this project.

Combining several classifiers seems to only be useful, when all of them produce similarly good results; otherwise a poor predictor ruins the performance of the better ones, unless it is a majority voting scheme. We first provide the results with the decision threshold at its normal value of 0.5. Tables 3.25 - 3.27 show three different ways of combining predictors: averaging their scores, taking AND-function of their predictions and majority voting.

Overall, a combination of classifiers tends to work a little better than each one of them individually. The AND-combination of classifiers

	Recall	Precision	F-score	Accuracy
Random forest + Gradient tree boosting + SVM (average score)	0.59	0.69	0.64	0.80
Random forest + Gradient tree boosting + SVM (AND)	0.49	0.74	0.59	0.79
Random forest + Gradient tree boosting + Support vector machine (voting)	0.60	0.68	0.64	0.80

Table 3.26: Evaluating three strategies for combining classifiers (WFP).

	Recall	Precision	F-score	Accuracy
Random forest + Gradient tree boosting + SVM (average score)	0.97	0.89	0.93	0.88
Random forest + Gradient tree boosting + SVM (AND)	0.96	0.91	0.93	0.90
Random forest + Gradient tree boosting + Support vector machine (voting)	0.97	0.91	0.94	0.90

Table 3.27: Evaluating three strategies for combining classifiers (ELP).

		Recall	Precision	F-score	Accuracy
Average	threshold = 0.8	0.38	0.90	0.54	0.87
	threshold = 0.9	0.18	0.95	0.30	0.84
	threshold = 0.95	0.07	0.99	0.14	0.82
AND	threshold = 0.8	0.25	0.91	0.39	0.85
	threshold = 0.9	0.08	0.97	0.15	0.82
	threshold = 0.95	0.01	1.0	0.02	0.80
Voting	threshold = 0.8	0.40	0.90	0.55	0.87
	threshold = 0.9	0.22	0.94	0.36	0.85
	threshold = 0.95	0.15	0.95	0.25	0.83

Table 3.28: The results for three classifier ensembles with different decision thresholds (WSJ).

		Recall	Precision	F-score	Accuracy
Average	threshold = 0.8	0.10	0.92	0.19	0.73
	threshold = 0.9	0.003	0.91	0.007	0.70
	threshold = 0.95	0.001	1.0	0.002	0.70
AND	threshold = 0.8	0.01	0.82	0.01	0.70
	threshold = 0.9	-	-	-	0.70
	threshold = 0.95	-	-	-	0.70
Voting	threshold = 0.8	0.20	0.88	0.32	0.75
	threshold = 0.9	0.07	0.95	0.13	0.72
	threshold = 0.95	0.04	0.96	0.07	0.71

Table 3.29: The results for three classifier ensembles with different decision thresholds (WFP).

produces a good increase in precision for two of the three sources (WFP and WSJ), even though for WFP it is accompanied by a decrease in recall.

Now these classifier ensembles can be combined with the previous step—moving the decision threshold. For averaging it is simply moving the threshold for the average score; for conjunction and voting the thresholds are moved for each of the classifiers individually before combining their predictions. Tables 3.28-3.30 show the results for different thresholds, using the ensembles on the three sources.

For all the sources combining classifiers and moving the threshold produces better results that just moving the threshold for one strong classifier. WFP is still the most problematic dataset, but the system reaches almost the same performance on it as on WSJ with the voting scheme. The voting scheme also appears to be the best ensemble for ELP. As for WSJ, other combination schemes are slightly better, but, focusing more on

		Recall	Precision	F-score	Accuracy
Average	threshold = 0.8	0.92	0.92	0.92	0.87
	threshold = 0.9	0.76	0.92	0.83	0.76
	threshold = 0.95	0.29	0.92	0.44	0.40
AND	threshold = 0.8	0.84	0.93	0.89	0.82
	threshold = 0.9	0.27	0.94	0.42	0.39
	threshold = 0.95	0.02	1.0	0.05	0.21
Voting	threshold = 0.8	0.93	0.91	0.92	0.87
	threshold = 0.9	0.81	0.93	0.87	0.80
	threshold = 0.95	0.58	0.95	0.72	0.63

Table 3.30: The results for three classifier ensembles with different decision thresholds (ELP).

the difficult WFP dataset, we choose the voting scheme with the decision threshold at 0.95 as the best option so far.

3.3.4.4 Final evaluation

We have described a number of techniques that were tried out to improve the baseline model. They were all evaluated on the development set, and some were kept for further use, while others were discarded. Our main goal was optimizing precision and not letting recall drop so low that the system would make almost no positive predictions at all. The best model so far is a combination of three classifiers: random forest, gradient boosted trees and a support vector machine with standard scaling and calibrated by isotonic regression. Each of the classifiers has a shifted decision threshold, i.e. it only predicts a positive instance when the probability of the positive class is higher than 0.95. The final prediction is a majority vote of the three predictors, i.e. the instance is assigned a positive label if and only if at least two classifiers out of three predicted it.

Table 3.31 contains the results of testing this model on the proper test set, which does not include any instances used during model selection. Tuning, threshold moving and deciding which modules to keep and which ones to omit can all be looked upon as hyper-parameter optimization. That means it is important to test the optimized model on a fresh test set, just like learning classifier parameters and evaluating it has to be done on two distinct datasets.

The results are slightly lower, but similar to the ones we got on the development set, which is exactly what was expected.

3.3.4.5 Transferring the model

The ensemble of classifiers constructed in the previous Section gives good results, when it is trained and tested on the data from the same website.

	Recall	Precision	F-score	Accuracy
WSJ	0.15	0.94	0.24	0.83
WFP	0.04	0.97	0.07	0.70
ELP	0.53	0.92	0.67	0.58

Table 3.31: The results of testing the final model on a fresh dataset for all three sources.

	Recall	Precision	F-score	Accuracy
Trained on WSJ, tested on WFP	0.16	0.64	0.25	0.72
Trained on WSJ tested on ELP	0.11	0.94	0.19	0.27
Trained on WFP, tested on WSJ	0.02	0.66	0.04	0.80
Trained on WFP, tested on ELP	0.007	1.0	0.01	0.20
Trained on ELP, tested on WSJ	0.45	0.68	0.54	0.85
Trained on ELP, tested on WFP	0.36	0.60	0.45	0.72

Table 3.32: The results for training the model on one source and evaluating on another.

However, the reason that probabilistic cross-device tracking is researched at all is the possibility to use it on websites that do not have logins. That means training a model on one or several websites that have logins and then applying it to the ones that do not. Table 3.32 shows how the model behaves when it has been trained on one source, and then tested on another source, unseen during training.

The results drop drastically, and even though they stay above random guessing, they become impossible to use without further refinement. While low recall was expected, because that the decision threshold had been set high, there is a big drop in precision as well, which makes the high threshold unjustified.

These are several hypotheses for why the results are low:

1. All three sources have different class size ratios (see table 3.1). Normally a classifier operates under the assumption that training and test set have the same class distribution. The assumption is violated here, which can be problematic (see 3.3.4.3.1). To test this hypothesis we can use undersampling and retrain the model so that in each case the class distributions for the training set and the test set are the same. This is done in 3.3.4.5.1.

2. If undersampling does not bring the results close to what they were before transferring the model, it means that the differences

between the sources are not only in class sizes, but in feature value distributions. This can be due to dataset shift or covariate shift, and is explained and illustrated in Section3.3.4.5.2.

3. If there are differences in feature value distributions, it might be caused by a specific group of features that behaves very differently for different sources. In that case there would be a simple way to fix it: removing that group of features would make the classifier test slightly worse on the same distribution, but become more robust to transferring to a different source. In 3.3.4.5.3 we look at feature value distributions, as well as which features are most important for different sources.

4. If the low results are not caused by any particular feature or group of features, it is possible that something similar to overfitting takes place. The classifier that we chose as the best one at the end of the previous Section (see 3.3.4.4) produced high results due to learning very precisely the distribution of the source it was trained on, so it is ill-fit to classify a different source. To test it, we can see if any simpler model is more robust to transferring (see 3.3.4.5.4).

3.3.4.5.1 Dealing with class ratios

As was mentioned in 3.3.4.3.7, most classifiers in one way or another account for the probability of each class label. This unavoidably leads to lower results if the class distributions of the training set and the test set are different, especially in cases like ELP and WSJ, where the class ratios are almost the opposite. To see if that is the biggest difference between the sources, we undersample the training set to have the same class ratio as the test set and retrain the classifier (table 3.33).

In some pairing undersampling made a great difference. For instance, a classifier trained on an undersampled set from ELP works better on both two other sources. Other transfers are still very problematic, like WSJ to WFP. Overall, the model cannot be used on a different source even if class distribution differences are taken into account.

3.3.4.5.2 Dataset shift and covariate shift

Apparently, there are more distinctions between the sources than the different class distribution. In setups like this, when train and test data have been collected using slightly different methods or sources,—in our case, they come from different websites,—there might occur such problems as a covariate shift or a dataset shift. A dataset shift is when the joint distribution of input feature vectors and output labels changes between training and test set. A lighter version of dataset shift is called covariate shift: in that case it is only a distribution of some input values that change. For an ideally fit classifier covariate shift would not be a problem: though the distribution of feature values has changed, conditional probabilities of

		Recall	Precision	F-score	Accuracy
trained: WSJ	threshold 0.5	0.87	0.52	0.65	0.72
tested: WFP	threshold 0.95	0.25	0.60	0.35	0.72
trained: WSJ	threshold 0.5	0.97	0.86	0.91	0.85
tested: ELP	threshold 0.95	0.80	0.92	0.86	0.78
trained: WFP	threshold 0.5	0.38	0.76	0.51	0.85
tested: WSJ	threshold 0.95	0.04	0.78	0.08	0.81
trained: WFP	threshold 0.5	0.94	0.88	0.91	0.85
tested: ELP	threshold 0.95	0.30	0.93	0.44	0.41
trained: ELP	threshold 0.5	0.27	0.70	0.40	0.83
tested: WSJ	threshold 0.95	0.08	0.73	0.15	0.81
trained: ELP	threshold 0.5	0.50	0.54	0.52	0.71
tested: WFP	threshold 0.95	0.10	0.81	0.18	0.72

Table 3.33: The results for training on a set, undersampled to fit the distribution of the test set.

labels given feature values remain the same. However, in reality a model does not always fit the true distribution perfectly: if it is misspecified, covariate shift might render it suboptimal on the test set [51].

The most straightforward way to check if the data is shifted is to train a classifier that would learn to separate two classes: training set and test set. If feature value distributions are the same for both sets, then the classifier is not going to be able to distinguish them any better than by random guessing. The idea is illustrated by Figure 3.11.

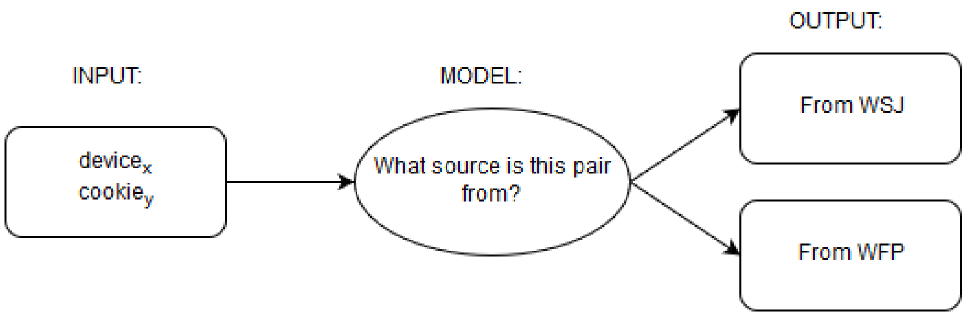


Figure 3.11: Detecting dataset shift by classification.

The biggest drop in results occurred between WSJ, a global financial newspaper, and a smaller regional all-purpose WFP. We tried to separate them, making a set with equal number of instances from both sources. Table 3.34 shows the results for such a classifier, trained to distinguish one source from another. It clearly works better than random guessing. The

	Recall	Precision	F-score	Accuracy
All features	0.85, 0.76	0.77, 0.84	0.81, 0.80	0.80
Without individual features	0.79, 0.64	0.68, 0.75	0.73, 0.69	0.71

Table 3.34: Predicting the source of the instance. Precision and recall for both classes.

most likely candidates to cause a covariate shift are individual features: language and geographical characteristics are bound to differ, depending on the newspaper. However, even after removing them from the vectors, it is still possible to distinguish between the two sets better than randomly. That means the differences between the sets lie deeper and concern the IP history as well.

Dataset shifts can sometimes be fixed if the distribution after the shift is known. Unfortunately, it would not be of particular use, if we wanted to make a model that works for new unlabelled sources, since we wouldn't know the true input-output distributions without labels. Nevertheless, in the next Section we look at features and their properties and try to determine whether the cause of the shift can be removed on these datasets.

3.3.4.5.3 Feature value distributions

One possible explanation for why models cannot be transferred is that because of different feature value distributions different features work well for different sources. To check that one needs to be able to tell which features are important in a given model.

Conceptually estimating feature importance is ranking features by how important they are for the performance of the classifier. Since we used a combination of several different classifiers, it would be more informative to look at how important features are for them individually, rather than take an average over all three. Gradient tree boosting produced good results close to the voting ensemble, so we will demonstrate the most important features for this algorithm alone.

One way to find out what features are important is to randomly reassign the values for one feature and see how much the performance was lowered. This method is also known as mean decrease accuracy, or mean error increase, first proposed in [13]. Scikit-learn uses a different method for tree-based algorithms: mean decrease impurity. Instead of measuring how accuracy is improved by this feature, it measures the reduction of impurity at the tree node that corresponds to the given feature. By definition, impurity is the probability of a random instance to be labelled incorrectly if the class label is assigned randomly according to the distribution of classes (see [32] for more details on decision tree feature importance). This is done for every tree in the model, and then the result for each feature are averaged over all trees.

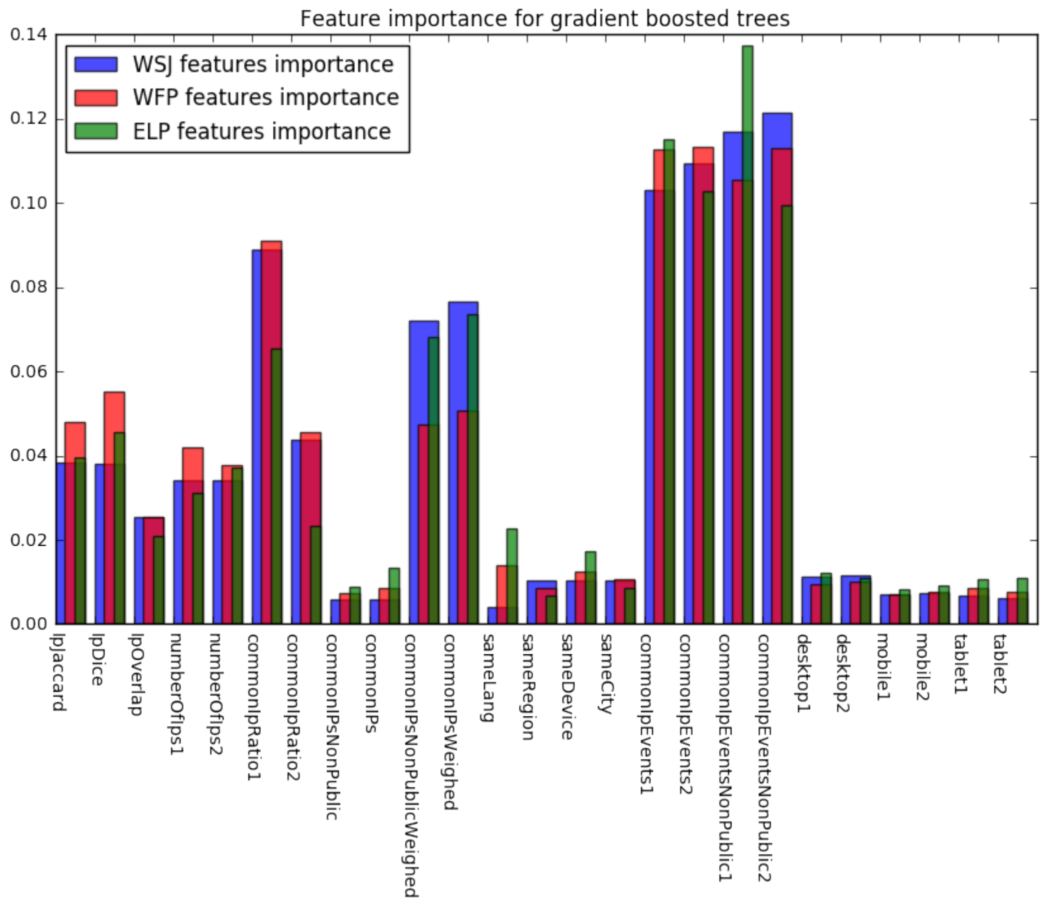


Figure 3.12: Feature importance for gradient boosted trees classifier on the three sources.

However, a graph of feature weights (Figure 3.12) seems to show that the feature importance is similar for all three sources.

Moreover, the distributions of values depending on class label have similar shapes and do not differ critically across websites. Figures 3.13 and 3.14 show the distribution of values for the most important feature as an example. To be able to compare samples of several thousand and several hundred thousand on the same graph, we scaled the values so that they lay in the same interval.

The spike at 1.0 in Figure 3.14 is likely due to a certain number of devices that have only been seen on one IP address. There was no particular group of features that showed different behaviour for different sources; therefore, the small variations in all feature value distributions put together are enough to confuse a classifier trained on another source and make it untransferable. These variations could be caused by a number of reasons. Geographical profiles are different for the audience of a big international newspaper and a smaller local one, which means the patterns of IP sharing also differ. Variations in user behavior between regions of the world, or in how traffic data is recorded and stored could also in

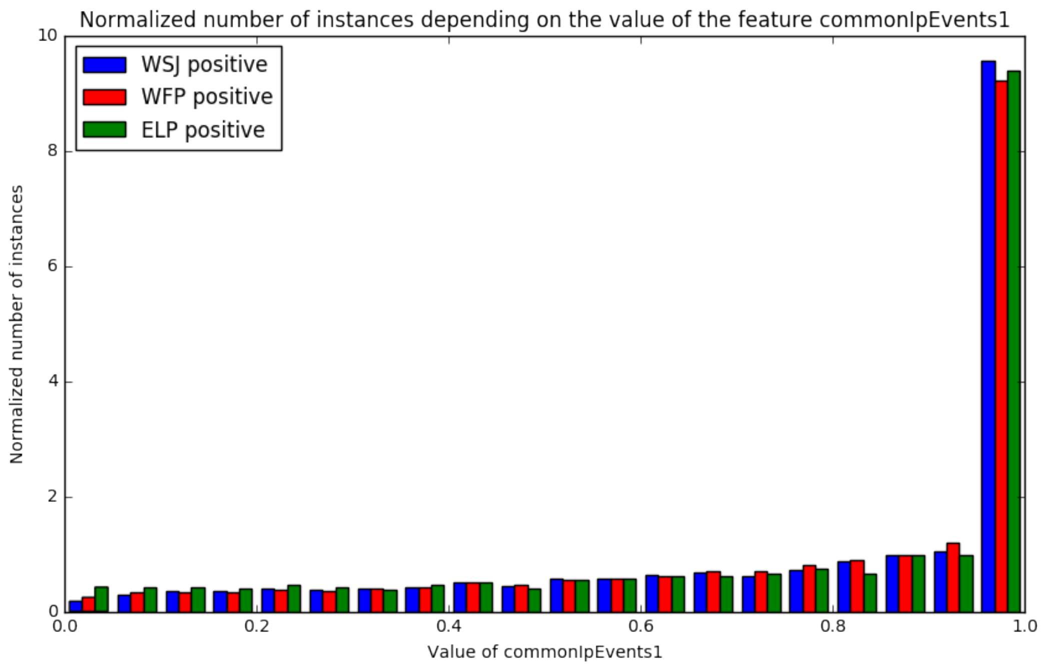


Figure 3.13: Feature value distribution among positive instances for the feature commonIpEvents1 for the three sources. Numbers are normalized to lie in the same intervals for all three sources.

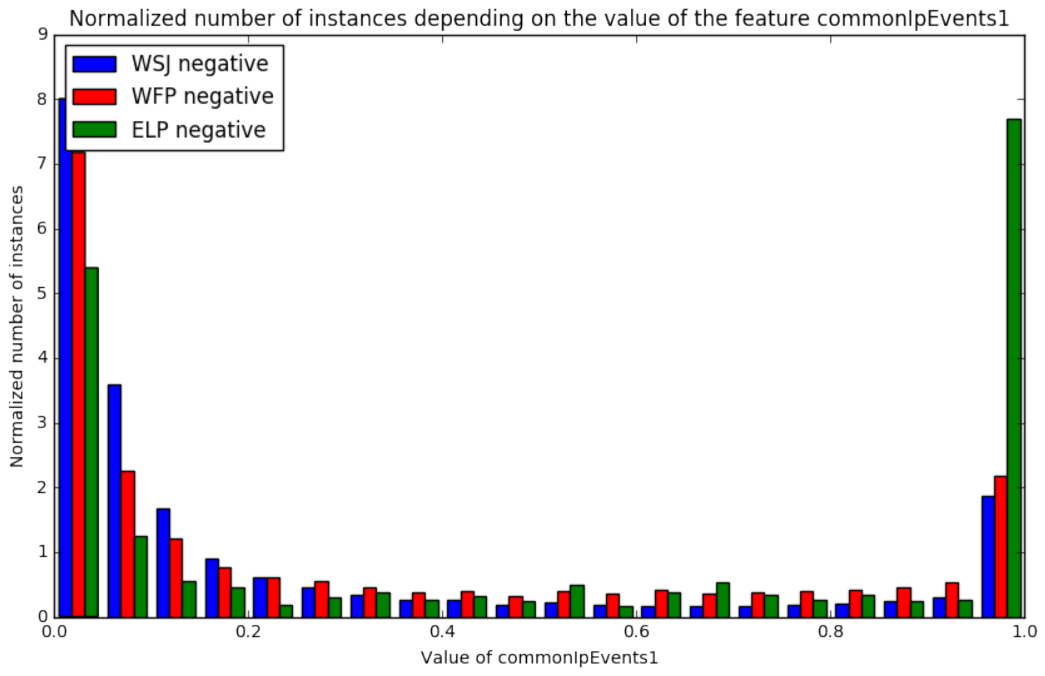


Figure 3.14: Feature value distribution among negative instances for the feature commonIpEvents1 for the three sources. Numbers are normalized to lie in the same intervals for all three sources.

theory contribute to the distinctions between datasets, but those are just speculations without a thorough analysis of each data source, which likely requires more information than what was stored in the traffic logs.

3.3.4.5.4 Avoiding "overfitting"

Since there is no apparent way to remove the differences between the datasets we can instead try to modify the classifier to make it more transferable. One possible reason the performance after transferring to another source is low can be similar in nature to overfitting.

There are two main types of erroneous behavior in classifiers: underfitting (caused by high bias) and overfitting (caused by high variance). Underfitting is when the algorithm fails to capture the relations in the data and does not learn enough information to classify instances correctly. Overfitting is the opposite: the algorithm becomes too sensitive to noise and variations in the data. As a result the model builds a very good boundary between classes, so that it describes the training set well, but fails to fit the distribution of the general sample. It usually happens when there are few instances with a lot of features, so there are many ways to separate the classes, and the most accurate boundary does not correspond to the real distribution of instances. As a result, at test time an overfit model would classify some instances falsely because it had focused on local variations in the training data. Figure 3.15 is an illustration of how overfitting might look like in 2-dimensional space.

When our model underperforms on a new source, it cannot be called "overfitting" in the usual sense: first, when trained and tested on the same website it can produce good results, second, instances from a new source are not from the same general sample, and finally there are no common conditions for overfitting like shortage of instances or numerous features. The situation in general, however, does appear close to overfitting: a distribution is learned quite well from a training set so that the model cannot work on a different set with similar, but slightly different distribution of feature values.

Table 3.35 compares performance of different classifiers trained on WSJ and tested on WFP. The first two of them have been tuned and produce better results when trained and tested on the same source. The second two are not tuned and have a low number of estimators.

T

The results at decision threshold 0.5 are almost the same for all three classifiers, but the advantage of the simple, un-tuned tree models with relatively few estimators is that moving the threshold helps, i.e. false positives are not distributed evenly, but are concentrated closer to the boundary.

It is not possible, however, to raise the decision threshold higher than 0.85 for robust gradient boosting classifier, because then the number of true positives become zero. It means that they are pushed closer toward the class boundary than they should be, i.e. that the classifier is badly

		Recall	Precision	F-score	Accuracy
Voting classifier	threshold 0.5	0.78	0.55	0.65	0.74
(SVM + RF + GTB)	threshold 0.95	0.17	0.67	0.27	0.72
Gradient tree boosting,	threshold 0.5	0.78	0.55	0.65	0.74
100 estimators, tree depth 10	threshold 0.85	0.45	0.53	0.49	0.63
Random forest,	threshold 0.5	0.78	0.55	0.65	0.74
50 estimators	threshold 0.95	0.14	0.74	0.23	0.72
Gradient tree boosting,	threshold 0.5	0.74	0.56	0.64	0.74
20 iterations, tree depth 5	threshold 0.85	0.06	0.82	0.12	0.71

Table 3.35: The results of several classifiers/classifier ensembles trained on WSJ and tested on WFP.

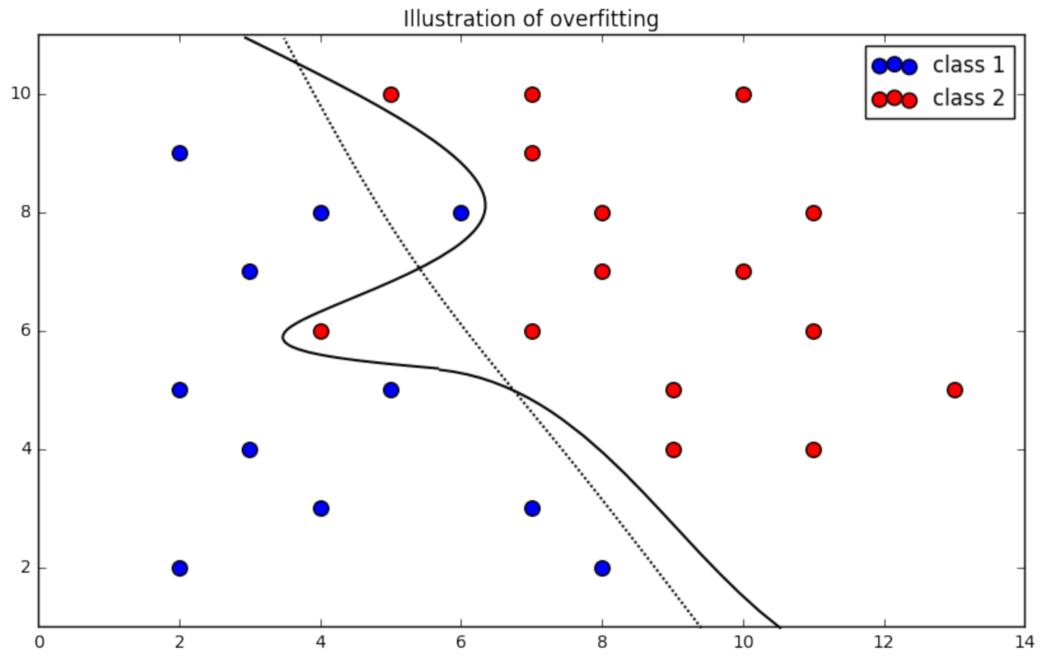


Figure 3.15: An example of overfitting. The dotted line represents the proper boundary between classes; the continuous line illustrates the behavior of an overfit classifier: it fits this particular set better, but will most probably lead to more mistakes on new instances.

calibrated (see 3.3.4.3.5). Though the difference was hardly noticeable before we tried transferring the model and decreasing the number of iterations, now calibration appears necessary. It is possible that different distributions lead not only to poorer performance, but also to poorer calibration.

In the following sections we do not use the voting classifier and instead switch to the robust version of gradient tree boosting with calibration (lowest row in table 3.35).

3.3.4.5.5 Combining datasets

Another way to make the classifier learn more generally is to mix data from different sources into the training set. It might help overcome the slight differences between the websites and get better results for new instances. Table 3.36 shows results for training on several sources and testing on instances from sets included in training, as well as from unseen sets. It should be noted that all mixed datasets have equal number of instances from every source included in them, and as a result the training sets that include ELP are smaller, since we have relatively few instances from ELP.

Naturally, classification works better on instances from a source that has been included into the training set. The threshold had been set at 0.85, which is lower than before, because recall falls faster here than when the

		Recall	Precision	F-score	Accuracy
Trained on: WSJ+WFP+ELP	Tested on: WSJ	0.22	0.87	0.34	0.84
	Tested on: WFP	0.18	0.66	0.30	0.73
	Tested on: ELP	0.34	0.94	0.50	0.45
Trained on: WSJ+WFP	Tested on: WSJ	0.04	0.98	0.08	0.82
	Tested on: WFP	0.07	0.86	0.13	0.72
	Tested on: ELP	0.03	1.0	0.05	0.21
Trained on: WSJ+ELP	Tested on: WSJ	0.49	0.80	0.61	0.88
	Tested on: WFP	0.48	0.61	0.53	0.75
Trained on: WFP+ELP	Tested on: WSJ	0.04	0.73	0.08	0.81

Table 3.36: The results of a robust gradient boosting classifier trained on data from several sources. The decision threshold is at 0.85.

model was tested on the same source. However, in most cases we could move the threshold even higher, and still get a number of true positives, because the classifier is properly calibrated now. But then recall would be too low and make it impractical.

When the source has not been seen in the training set, WFP remains the biggest problem, and the fact that it is classified worse than other sources in the same situation leads to believe that these problems likely stem from individual peculiarities of WFP dataset, and not from the transfer itself.

As for other sources, using a mixed training set is another step towards a model that can be used in practice, but the performance is still in most cases not high enough for that. It is possible that with a large number of different sources combining them all in the training set would help more, but this would not be an ideal solution: even if on a certain number of sources the model works, there is no guarantee that the next website does not have usage patterns slightly different from everything that has been seen before. Unfortunately, it is not always easy to check whether it is the case for the same reason that cross-device tracking is interesting at all: we would need labelled data, i.e. login information, which does not exist in sufficient amounts for a lot of media websites.

3.4 Unsupervised learning

The main problem with supervised learning in our experiments was that the model is not transferable from one source to another. In other words, the exact location of class boundaries can vary from source to source. An unsupervised approach, however, would solve that problem: if there were a way to infer the boundaries from the data directly, then we would not need to rely on the labelled data that is not always available in sufficient amounts. This Section briefly touches on unsupervised and semi-supervised methods.

3.4.1 Preprocessing

There are two possible strategies for preprocessing for unsupervised learning. The first one is to follow the same steps as in supervised learning to arrive at a set of pairs. We could then switch from a supervised algorithm to an unsupervised one, and the goal would change from classification to clustering. The set of pairs would need to be split into two clusters which corresponded as well as possible to the positive and negative class. Since we are not aiming for complete coverage, it would be enough to find *some* cluster in the dataset that consisted of matching pairs with good precision. This approach is described in Section 3.4.2.

Another strategy is to represent individual devices as vectors in multidimensional space in a way that would make the devices from the same user be closer to each other than from different users. This technique is widely used in language processing, namely—distributional semantics. The idea is to model words as vectors in a space, where the dimensions are other words, that co-occur with them. One of the most famous systems, exploiting this idea, is word2vec, developed in 2013 by Tomas Mikolov [35].

Figure 3.16 shows the preprocessing steps for the second strategy. It starts in the same way as preprocessing for supervised learning, and devices with few events and with several logins are filtered out. But then instead of assembling devices into pairs we create a feature vector for each individual device, where dimensions are IP addresses.

Since a great portion of information in classification turned out to come from the features based on IPs, we can try to model a device by its IP "context", similar to how context words are used in distributional semantics. Each IP address is a dimension, and an instance scores as much on that dimension as many times that device_id has been seen from that address. The number of dimensions is going to be large: the longer the time period, the larger. However, the data is going to be very sparse, i.e. most values on most dimensions are going to be zero, because most devices have only been seen on a couple of IPs. Of course, those zeros are not stored in the vectors. Both high dimensionality and sparseness are characteristic of language processing as well. Finally, the feature vectors are transformed to have unit length.

Intuitively this approach makes good sense: the bigger the overlap between the addresses, the more likely it is that the devices are used by the same person. Assuming it is a good representation of devices, we can measure the distance between them to estimate how likely it is that the devices are from the same user. This approach is further described in Section 3.4.3.

3.4.2 Clustering pairs

It would be logical to assume that if it is possible to classify pairs for cross-device tracking, then positive pairs are in some aspects closer to each other than to negative ones. We could therefore try to cluster them into two clusters and see if they correspond to the positive and negative class. But

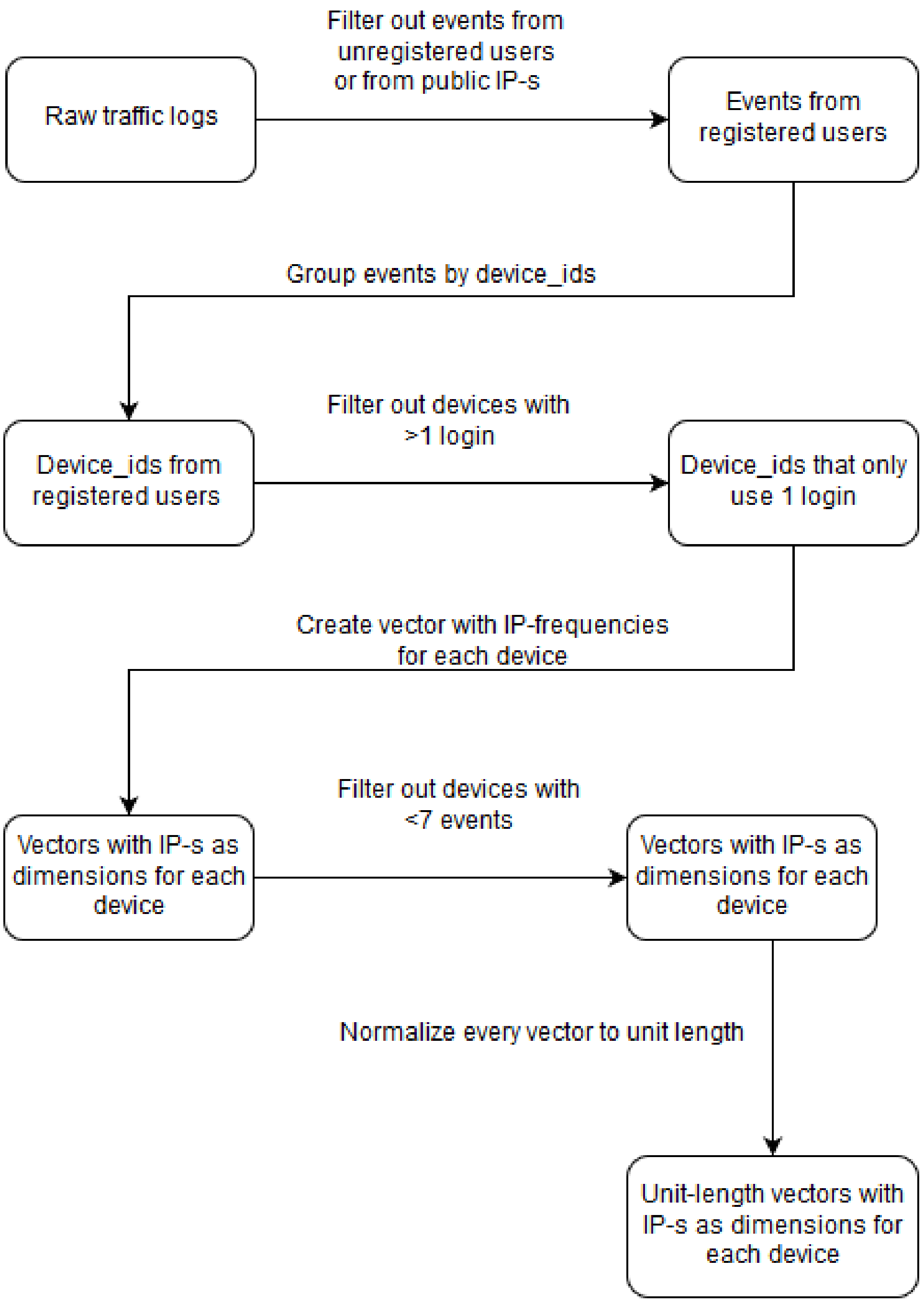

Filter out events from unregistered users or from public IP-s
Raw traffic logs
Events from registered users
Group events by device_ids
Filter out devices with >1 login
Device_ids from registered users
Device_ids that only use 1 login
Create vector with IP-frequencies for each device
Filter out devices with <7 events
Vectors with IP-s as dimensions for each device
Vectors with IP-s as dimensions for each device
Normalize every vector to unit length
Unit-length vectors with IP-s as dimensions for each device

Figure 3.16: Steps of preprocessing for unsupervised learning.

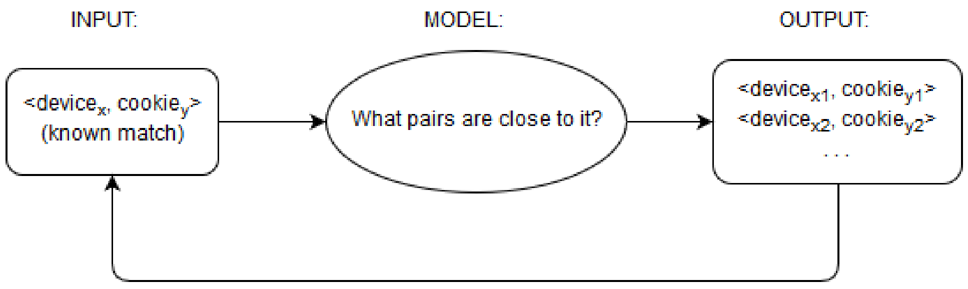

INPUT:
MODEL:
OUTPUT:
<device$_x$, cookie$_y$> (known match)
What pairs are close to it?
<device$_{x1}$, cookie$_{y1}$>
<device$_{x2}$, cookie$_{y2}$>
. . .

Figure 3.17: Expanding the set of matches through semi-supervised learning.

there is a number of problems with that approach. First, since the model is not aware of which variable it should split by, it is quite probable that there are other properties that divide the dataset better, and the clusters will be based on them, instead of on whether an instance is a match. Second, a lot of clustering algorithms are poorly suited for the task, because they aim at finding clusters with similar density, or of approximately equal sizes, or make some other assumptions about the data distribution that we have no way of estimating in advance, if the data is unlabelled.

We have tried to use several clustering algorithms, such as agglomerative clustering and K-means with $K = 2$, on the pairs in a completely unsupervised way and see whether positive instances were clustered together, but there was no correlation between clusters and class label. Instead, we can use a semi-supervised approach with a k-nearest-neighbors algorithm.

K-nearest-neighbors is a technique used both in supervised and unsupervised learning. The main idea is to find out which instances are closest to each other in the multidimensional space, where dimensions are features. It is explained in more detail in the next section (3.4.3). In the end, we have information about how far all instances are from each other.

If a small number of labelled instances is available, it is possible to use them as seeds to make the initialization of the algorithm less random and more relevant to the variable we want to predict [12]. Suppose we have a way of obtaining a small set of matching pairs (the seed set): whether it is through logins, or after using a previously trained classifier, or through some manually written heuristic. In order to expand this set we can find which pairs are closest to the ones in the set. The idea is shown in Figure 3.17.

The simplest way to find new matches is to take the nearest neighbor of each instance in the seed set and label it as positive. Of course, it is difficult to talk about recall in this situation: if the general set has thousands of pairs, and the seed set that the algorithm starts with is, for example, 100 instances, then by adding one nearest neighbor for each of those 100 instances we can find at most 100 new positives. Instead we only measure precision, i.e. what percentage of the newly found instances is in fact positive.

First we need to obtain the seeds. Taking a subset of 100 random positive instances can produce very different results, depending on how representative the subset turned out to be. It is preferable to take the pairs that are representative of the positive class and have a smaller chance of being close to negative pairs. A good way to estimate the degree of ”positivity” for an instance is to use a trained classifier on it, so that it returns a score or a probability that the instance belongs to the positive class. In short, we trained a model on a training set, applied it to a different labelled set and randomly chose a hundred true positive instances that got a score over 0.9 from the model. These instances would be the seeds. We then extracted the nearest neighbor for each seed, filtered out the cases when the nearest neighbor already was one of the seeds, and computed precision on the extracted neighbors.

Tables 3.37 - 3.38 compare results for taking the nearest neighbors of 100 random positive instances and of 100 instances that had received a high score from a classifier. The numbers were averaged over five randomized seed sets of each type. Since the results from ELP would not be very informative, as it has a much bigger positive class than negative, we only provide the results for WSJ and WFP.

Seeds	Number of distinct new instances among the set of nearest neighbors	Precision (fraction of positives in the set of the neighbors)
100 random positive instances	98	0.52
100 positive instances with >0.9 score from a classifier	91	0.72

Table 3.37: Precision for positive pairs among the nearest neighbors of 100 seed instances from WSJ.

Seeds	Number of distinct new instances among the set of nearest neighbors	Precision (fraction of positives in the set of the neighbors)
100 random positive instances	98	0.59
100 positive instances with >0.9 score from a classifier	97	0.86

Table 3.38: Precision for positive pairs among the nearest neighbors of 100 seed instances from WFP.

Choosing the more ”certain” instances of the positive class as seeds predictably produces better results, though not high enough for a practical application. Ideally we would like to repeat this extraction several times, but it will obviously stop finding positive pairs as soon as the set gets closer to the class boundary, where the nearest neighbors are negative pairs. That is why instead of taking the nearest neighbor we can set an absolute threshold on the distance to the neighbors that we add to the positive set. In that case we can also lift the limitation of one neighbor per seed and simply add every instance that is inside the maximum allowed radius from any seed. Tables 3.39 - 3.40 show the results for different maximum distances for WSJ and WFP.

	Number of distinct new instances among the neighbors in the radius	Precision (fraction of positives in the set of the neighbors)
radius = 1	8121	0.59
radius = 0.5	5298	0.72
radius = 0.1	1785	0.90

Table 3.39: Number of neighbors of seed instances from WSJ in a fixed radius and precision of positives among them.

	Number of distinct new instances among the neighbors in the radius	Precision (fraction of positives in the set of the neighbors)
radius = 1	1407	0.73
radius = 0.5	1206	0.76
radius = 0.1	670	0.82

Table 3.40: Number of neighbors of seed instances from WFP in a fixed radius and precision of positives among them.

A small threshold provides better results than many supervised methods have. It might be especially useful for WFP, which is not very successfully classified the by supervised models. It is possible to repeat this neighbor extraction more than one time on the same seed set, or on several different ones. In the latter case we would find several different small clusters of positive pairs, instead of expanding the same one further. In this project we did not have the time to explore how performance changes in those situations.

This semi-supervised technique can be combined with the classifiers that were developed in the previous sections and have a high precision and low recall. In that case it would not be difficult to obtain a small set of representative positive seeds to expand with *k*-nearest-neighbors. It is true that the need for a seed set makes the system more dependent on labelled data, but the seed set does not need to be large. It can be extracted as long as the website has a few registered users, possibly without even involving machine learning, but some hand-written rules for picking the more prototypical matches. The size of the seed set, the distance threshold and the hyper-parameters of the *k*-nearest-neighbors algorithm still have to be tuned, but overall this approach looks promising, especially for problematic datasets like WFP, where the classes are not well-separable.

3.4.3 Clustering devices

After following the preprocessing steps in Figure 3.16, we have a set of devices, each represented by a long sparse vector, where the dimensions are IP addresses. In order to compute how close they are to each other, we need to determine what distance to use in this high-dimensional space. Popular options include Euclidean distance and cosine distance, or similarity, as it is usually called, because it is defined on a scale from zero to one, where one corresponds to the points being closest. Cosine similarity is usually preferred in language processing over Euclidean distance, because it normalizes the vectors first, i.e. transforms them to be of unit length. That way the words with similar meanings but very different frequencies get a better score than unrelated words that are close in the Euclidean sense only because they are both very rare.

This consideration is just as applicable to devices: the input from their IP profiles should be more important than how frequently they were seen.

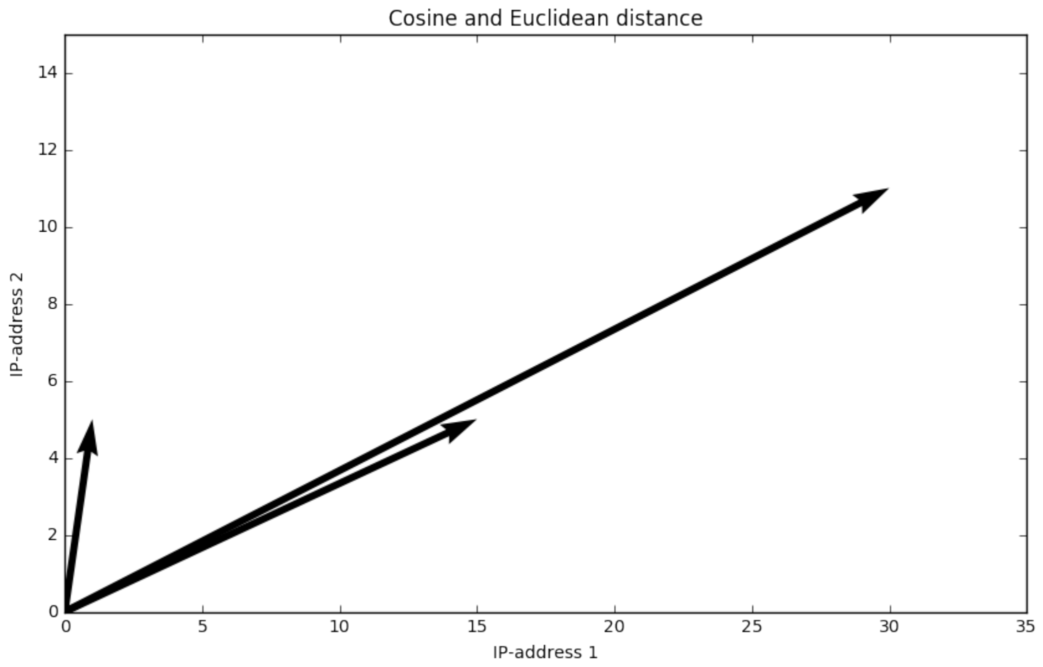


Figure 3.18: A toy example of three instances in 2-dimensional space.

Figure 3.18 shows a hypothetical example when using Euclidean distance might lead to suboptimal results: suppose there are two IP addresses in the set, and the vectors represent three devices. The left-most vector (1, 5) has only been seen on IP address-1 once, possibly as an isolated incident, while the other two vectors—(15, 5) and (30, 11)—have used both addresses many times, yet (1, 5) is closer to (15, 5), than

	WSJ	WFP
Number of devices	657 044	46 178
Number of >0 pair-wise similarities	1 683 886	102 734

Table 3.41: Number of devices with logins and >7 events for 1 month, and number of pairs between them that have a >0 similarity (= at least 1 common IP address).

	Precision for WSJ	Precision for WFP
$sim(d_1, d_2) > 0.95$	0.85	0.78
1 nearest neighbor	0.69	0.60

Table 3.42: Precision of positives among closest pairs and pairs with >0.95 similarity for WSJ and WFP.

(30, 11) in Euclidean sense. Cosine similarity measures the cosine of the angle between the vectors, ignoring their length, so it appears to be a more sensible choice in our case than Euclidean distance.

Conceptually the next step is creating a matrix of size $N \times N$, where N is the number of devices in the dataset, and the values are pair-wise similarities. Of course, only half of the table has to be stored, since the similarity is a symmetric function, i.e. $sim(A, B) = sim(B, A)$. With several thousand devices it is still a huge number of values, but in practice we need much less space. Since the vectors are very sparse and have 0 in most dimensions, a large number of devices will not have any common IP addresses, and so their similarity will be zero. There is no need to keep those values either, in the same way as we did not include pairs without common IPs during preprocessing for supervised learning. In the end the number of values becomes manageable: Table 3.41 shows how many non-zero similarities there are between devices after the preprocessing step from WSJ and WFP.

Like when classifying pairs, there are two straight-forward ways to find matches, making use either of absolute or relative similarities. The first way is to choose a threshold and label any pair that is more similar as a match, and the second way is to only label one (or more) nearest neighbor as a match for every device. The results of both methods are shown in Table 3.42. There is not much use in reporting recall, because this clustering model works its way up from the individual devices to find some of the positive instances, unlike a classifier, that has a general overview of the data and is expected to pay attention to both classes. That is why only precision is reported (the fraction of true positives among the uncovered device pairs).

The task is easier for WSJ, and, just as in the previous section, an absolute threshold works better than taking the first neighbor. An absolute threshold has its short-comings too, one of which is related to what was discussed in [55]: it is more difficult to establish a boundary between

	WSJ	WFP
Fraction of devices with min. 1 match that have a match among three closest neighbors	0.91	0.83

Table 3.43: Precision among the devices that are known to have at least 1 match in the set.

classes that is applicable to every user, than to choose for each user individually which candidate is most likely to be a match. Unsupervised learning faces a similar situation: a user that moves around a lot might have a relatively low similarity between his phone and his laptop, even if they are each other's closest neighbors. The second method solves this problem by counting only the closest neighbor as a match. However, its disadvantages appear to outweigh its benefits. First, it is guaranteed to miss a lot of matches, since it is known in advance that some users have more than two devices. Second, it leads to a certain number of false positives, since a lot of users still only have one device. This last problem is new compared to the supervised learning experiments, where the dataset only consisted of devices that had at least one match.

Despite the problems described above, this setup—representing the data in the IP-space—seems like a viable way to approach cross-device tracking. Even though the devices without a pair can be a problem and contribute to the number of false positives, in the cases when devices do have a pair, that pair is normally among their three closest neighbors (see Table 3.43). Overall, it is a promising approach, if it can be refined further, because it avoids the problem of transferring the model from one source to another.

To improve precision we can take a conjunction of the two methods, or a more complicated combination of them, but due to time limitations it has not been tried out in this project.

Chapter 4

Discussion and conclusions

4.1 Summary of results

In this project cross-device tracking has been approached as a classification problem and as a clustering problem. For the supervised part we constructed a training set consisting of two classes: the positive class, that contains pairs of devices belonging to the same user, and the negative class, with pairs that belong to different users. We then trained several classifiers, enhancing them by feature scaling, tuning and other pre- and post-processing modules. The most informative features proved to be the ones related to IP ”footprints” of devices. The main evaluation metrics were recall and precision of the positive class, with a much higher focus on precision than on recall. Best performance was achieved by ensembles from several different classifiers, and non-linear tree-based algorithms. The main problem of supervised models is that they do not necessarily work as well on data from a different website as the one where the training set came from. Another important observation that can be drawn from the classification experiments is how some websites are less suitable for cross-device tracking with our methods than others, either because of their geographical profile or for other reasons.

For the unsupervised and semi-supervised part we introduced a metric on devices and on pairs. The unsupervised model worked with individual devices: it used IP addresses as dimensions and represented devices as vectors in the IP-space, similar to how words are represented in distributional semantic models. It then counted close pairs of devices as positive pairs. The semi-supervised model worked with pairs: it started with a small number of known positive pairs and found the close neighbors for each pair in the feature space. Both approaches received less attention in this project than the supervised approach, but they appear worth researching further, since they avoid the problem of transferring the model from one website to another.

On the whole, only a supervised model trained and tested on the data from the same website achieves results high enough for practical applications on some of the data sources. Neither transferring a supervised model nor using unsupervised (or semi-supervised) learning produces good precision with so few false positives that they can be overlooked. The next Section provides suggestions on what could be done to improve them further.

4.2 Further work

Some of the goals that had been set for this project have been achieved, but there are still unsolved problems, especially regarding the differences between datasets. In cross-device tracking, unlike other, more established topics, there are no commonly used guidelines on how to represent the data or what machine learning tools work best. As a result, there are a lot of directions to go from this point, which have not been explored yet.

Optimizing the hyper-parameters of the system

There have been a lot of cases when a value had to be fixed based solely on intuitive assessement. This includes the threshold to remove widely used IP address, the minimum number of events for a device to be included into the dataset, the time period that the traffic logs were taken from, etc. It is possible to optimize those hyper-parameters of the system, although it is costly both in terms of time and memory.

Feature engineering

The feature set that was used in the project has room for improvement. We had 25 features, which is a rather small number, especially considering some of them were highly correlated. A possibility that has not been explored in this project at all is temporal features, which make use of the time stamps from the events.

Expanding data

Our data had a number of significant limitations. For instance, every identifier was limited to one website. If that restriction is lifted, then the user's timeline would be more complete, and the co-occurrences in their internet history might also be helpful in finding matches. It might also be worth processing more sources than just three. It would be challenging to find a lot of sources with enough labelled data, and preprocessing them would take a lot of time. But it might help find out how much sources differ and whether a model trained on examples from dozens of various sources would perform better on a new unseen one. Research into data distributions might also help explain why on some sources classification performs poorly, even when training and testing is done on the same source, like in the case of WFP, and whether it is possible to recognize such cases without access to labelled data.

Unsupervised and semi-supervised methods

This project deals with supervised learning in more detail than unsupervised and semi-supervised methods, due to the absence of earlier research on applying those method to cross-device tracking, and time constraints. Meanwhile those approaches seem promising. In our opinion, they might be more promising, than the supervised methods, considering how unsupervised models do not have to be transferred from one source to another, and how they useful they are when labelled data is scarce.

Bibliography

[1] *1.1.11. Logistic Regression*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (visited on 25/04/2017).

[2] *1.11.2. Forests of randomized trees*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/ensemble.html#forest (visited on 25/04/2017).

[3] *1.11.4. Gradient Tree Boosting*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting (visited on 25/04/2017).

[4] *1.13. Feature selection*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/feature_selection.html (visited on 25/04/2017).

[5] *1.16. Probability calibration*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/calibration.html (visited on 25/04/2017).

[6] *1.4. Support Vector Machines*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/svm.html (visited on 25/04/2017).

[7] *1.9. Naive Bayes — scikit-learn 0.18.1 documentation*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/naive_bayes.html (visited on 25/04/2017).

[8] *3.2. Tuning the hyper-parameters of an estimator*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/grid_search.html (visited on 25/04/2017).

[9] *4.3. Preprocessing data*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/modules/preprocessing.html (visited on 25/04/2017).

[10] *Adbrain: Demystifying Cross-Device*. Whitepaper. Adbrain. URL: https://iabuk.net/resources/white-papers/adbrain-demystifying-cross-device (visited on 25/04/2017).

[11] M. Balakrishnan, I. Mohomed and V. Ramasubramanian. 'Where's that phone?: geolocating IP addresses on 3G networks'. In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM. 2009, pp. 294–300.

[12] S. Basu, A. Banerjee and R. J. Mooney. 'Semi-supervised Clustering by Seeding'. In: *Proceedings of the Nineteenth International Conference on Machine Learning*. ICML '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 27–34. URL: http://dl.acm.org/citation.cfm?id=645531.656012 (visited on 26/04/2017).

[13] L. Breiman. 'Random forests'. In: *Machine learning* 45.1 (2001), pp. 5–32. URL: http://link.springer.com/article/10.1023/A:1010933404324?LI=true.

[14] J. Brookman et al. 'Cross-Device Tracking: Measurement and Disclosures'. In: *Proceedings on Privacy Enhancing Technologies* 2017.2 (2017), pp. 133–148. URL: https://www.degruyter.com/view/j/popets.2017.2017.issue-2/popets-2017-0020/popets-2017-0020.xml?format=INT (visited on 25/04/2017).

[15] X. Cao, W. Huang and Y. Yu. 'Recovering Cross-Device Connections via Mining IP Footprints with Ensemble Learning'. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015, pp. 1681–1686.

[16] R. Díaz-Morales. 'Cross-Device Tracking: Matching Devices and Cookies'. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015, pp. 1699–1704.

[17] *Drawbridge Cross-Device Connected Consumer Graph Is 97.3% Accurate*. WebWire. URL: http://www.webwire.com/ViewPressRel.asp?aId=197231 (visited on 25/04/2017).

[18] C. Drummond and R. C. Holte. 'C4.5, Class Imbalance, and Cost Sensitivity: Why Under-sampling beats Over-sampling'. In: *Workshop on learning from imbalanced datasets II*. Vol. 11. 2003, pp. 1–8.

[19] T. Fawcett. *ROC Graphs: Notes and Practical Considerations for Researchers*. 2004.

[20] *FTC Issues Warning Letters to App Developers Using 'Silverpush' Code*. Press release. Federal Trade Commission. URL: https://www.ftc.gov/news-events/press-releases/2016/03/ftc-issues-warning-letters-app-developers-using-silverpush-code (visited on 25/04/2017).

[21] D. Goldberg et al. 'Using Collaborative Filtering to Weave an Information Tapestry'. In: *Commun. ACM* 35.12 (1992), pp. 61–70. URL: http://doi.acm.org/10.1145/138859.138867 (visited on 25/04/2017).

[22] H. Grassegger and M. Krogerus. 'The Data That Turned the World Upside Down'. In: *Motherboard* (28th Jan. 2017). URL: https://motherboard.vice.com/en_us/article/how-our-likes-helped-trump-win (visited on 25/04/2017).

[23] H. He and E. A. Garcia. 'Learning from Imbalanced Data'. In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (Sept. 2009), pp. 1263–1284.

[24] C.-w. Hsu, C.-c. Chang and C.-j. Lin. 'A practical guide to support vector classification'. 2010. URL: http://www.datascienceassn.org/sites/default/files/Practical%20Guide%20to%20Support%20Vector%20Classification.pdf.

[25] *ICDM 2015: Drawbridge Cross-Device Connections*. Kaggle. URL: https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections (visited on 25/04/2017).

[26] N. Japkowicz and S. Stephen. 'The Class Imbalance Problem: A Systematic Study'. In: *Intell. Data Anal.* 6.5 (2002), pp. 429–449. URL: http://dl.acm.org/citation.cfm?id=1293951.1293954 (visited on 25/04/2017).

[27] G. Kejela and C. Rong. 'Cross-Device Consumer Identification'. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015, pp. 1687–1689.

[28] M. S. Kim et al. 'Connecting Devices to Cookies via Filtering, Feature Engineering, and Boosting'. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015, pp. 1690–1694.

[29] M. Kosinski, D. Stillwell and T. Graepel. 'Private traits and attributes are predictable from digital records of human behavior'. In: *Proceedings of the National Academy of Sciences* 110.15 (4th Sept. 2013), pp. 5802–5805. URL: http://www.pnas.org/content/110/15/5802 (visited on 25/04/2017).

[30] M. Landry, S. Rajkumar and R. Chong. 'Multi-layer Classification: ICDM 2015 Drawbridge Cross-Device Connections Competition'. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015, pp. 1695–1698.

[31] R. Longadge and S. Dongre. 'Class Imbalance Problem in Data Mining Review'. In: *arXiv:1305.1707 [cs]* (7th May 2013). arXiv: 1305.1707. URL: http://arxiv.org/abs/1305.1707 (visited on 25/04/2017).

[32] G. Louppe et al. 'Understanding Variable Importances in Forests of Randomized Trees'. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems*. NIPS'13. USA: Curran Associates Inc., 2013, pp. 431–439. URL: http://dl.acm.org/citation.cfm?id=2999611.2999660 (visited on 26/04/2017).

[33] 'Managing associations between device identifiers'. Patent Application 705/14.49. TapAd, Inc. URL: http://www.google.com/patents/US20130124309 (visited on 25/04/2017).

[34] 'Method and System for cross-device targeting of users'. Patent application 14/606,227. Silveredge, Inc. URL: http://www.google.com/patents/US20150215668 (visited on 25/04/2017).

[35] T. Mikolov et al. 'Efficient Estimation of Word Representations in Vector Space'. In: *arXiv:1301.3781 [cs]* (16th Jan. 2013). arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781 (visited on 26/04/2017).

[36] H. Mordt. *Tiden er over for armbåndsur på eksamen*. NRK. 31st Mar. 2017. URL: https://www.nrk.no/ostlandssendingen/tiden-er-over-for-armbandsur-pa-eksamen-1.13454212 (visited on 25/04/2017).

[37] A. Niculescu-Mizil and R. Caruana. 'Predicting Good Probabilities with Supervised Learning'. In: *Proceedings of the 22Nd International Conference on Machine Learning*. ICML '05. New York, NY, USA: ACM, 2005, pp. 625–632. URL: http://doi.acm.org/10.1145/1102351.1102430 (visited on 25/04/2017).

[38] A. Øhrn. Personal communication. 16th Apr. 2016.

[39] E. Pariser. *Beware online "filter bubbles"*. URL: https://www.ted.com/talks/eli_pariser_beware_online_filter_bubbles (visited on 25/04/2017).

[40] M. Pazzani and D. Billsus. 'Learning and Revising User Profiles: The Identification of Interesting Web Sites'. In: *Machine Learning* 27.3 (1st June 1997), pp. 313–331. URL: https://link.springer.com/article/10.1023/A:1007369909943 (visited on 25/04/2017).

[41] J. C. Platt. 'Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods'. In: *Advances in Large Margin Classifiers*. MIT Press, 1999, pp. 61–74.

[42] F. Provost. *Machine Learning from Imbalanced Data Sets 101*. Technical report. AAAI Workshop on Learning from Imbalanced Data Sets, 2000.

[43] F. Provost and T. Fawcett. 'Robust Classification for Imprecise Environments'. In: *Machine Learning* 42.3 (1st Mar. 2001), pp. 203–231. URL: https://link.springer.com/article/10.1023/A:1007601015854 (visited on 25/04/2017).

[44] A. Ramesh et al. 'Audience Segment Expansion Using Distributed In-database K-means Clustering'. In: *Proceedings of the Seventh International Workshop on Data Mining for Online Advertising*. ADKDD '13. New York, NY, USA: ACM, 2013, 5:1–5:9. URL: http://doi.acm.org/10.1145/2501040.2501982 (visited on 25/04/2017).

[45] *Republic*. URL: https://republic.ru/ (visited on 25/04/2017).

[46] M. Robinson. *Nielsen Confirms Tapad Cross-Device Accuracy at 91.2%*. Tapad. 2nd Dec. 2014. URL: https://www.tapad.com/news/blog/nielsen-confirms-tapad-cross-device-accuracy-at-91-2 (visited on 25/04/2017).

[47] J. Rucker and M. J. Polanco. 'Siteseer: Personalized Navigation for the Web'. In: *Commun. ACM* 40.3 (1997), pp. 73–76. URL: http://doi.acm.org/10.1145/245108.245125 (visited on 25/04/2017).

[48] A. L. Samuel. 'Some Studies in Machine Learning Using the Game of Checkers'. In: *IBM J. Res. Dev.* 3.3 (1959), pp. 210–229. URL: http://dx.doi.org/10.1147/rd.33.0210 (visited on 25/04/2017).

[49] *Scikit-learn: machine learning in Python — scikit-learn 0.18.1 documentation*. Scikit-learn 0.18.1 documentation. URL: http://scikit-learn.org/stable/ (visited on 25/04/2017).

[50] L. R. Selsaas et al. 'AFFM: Auto feature engineering in field-aware factorization machines for predictive analytics'. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 2015 IEEE International Conference on Data Mining Workshop (ICDMW). 2015, pp. 1705–1709.

[51] H. Shimodaira. 'Improving predictive inference under covariate shift by weighting the log-likelihood function'. In: *Journal of statistical planning and inference* 90.2 (2000), pp. 227–244.

[52] 'System to group internet devices based upon device usage'. US patent 9514248. Drawbridge, Inc. 6th Dec. 2016. URL: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&co1=AND&d=PTXT&s1=9,514,248.PN.&OS=PN/9,514,248&RS=PN/9,514,248 (visited on 25/04/2017).

[53] *The New Multi-Screen World Study*. Research study. Google. URL: https://www.thinkwithgoogle.com/research-studies/the-new-multi-screen-world-study.html (visited on 25/04/2017).

[54] *US Ad Spending: eMarketer's Updated Estimates and Forecast for 2015–2020 - eMarketer*. Financial report. eMarketer. URL: https://www.emarketer.com/Report/US-Ad-Spending-eMarketers-Updated-Estimates-Forecast-20152020/2001915 (visited on 25/04/2017).

[55] J. Walthers. 'Learning to Rank for Cross-Device Identification'. In: *Proceedings of the 2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. ICDMW '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1710–1712. URL: http://dx.doi.org/10.1109/ICDMW.2015.246 (visited on 25/04/2017).

[56] J. Walthers and A. Davis. *Learning to rank for cross-device identification*. 12th Jan. 2016. URL: https://www.youtube.com/watch?v=7UHOa2JvC9k (visited on 25/04/2017).

[57] X. Wu et al. 'Probabilistic Latent Semantic User Segmentation for Behavioral Targeted Advertising'. In: *Proceedings of the Third International Workshop on Data Mining and Audience Intelligence for Advertising*. ADKDD '09. New York, NY, USA: ACM, 2009, pp. 10–17. URL: http://doi.acm.org/10.1145/1592748.1592751 (visited on 25/04/2017).

[58] W. J. Youden. 'Index for rating diagnostic tests'. In: *Cancer* 3.1 (1950), pp. 32–35. URL: http://dx.doi.org/10.1002/1097-0142(1950)3:1%3C32::AID-CNCR2820030106%3E3.0.CO;2-3.

[59] B. Zadrozny and C. Elkan. 'Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers'. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 609–616. URL: http://dl.acm.org/citation.cfm?id=645530.655658 (visited on 25/04/2017).