Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Exploring Migrations and Spread of Chagas Disease in Latin America with Machine Learning

Explorando Migraciones y Difusión del Mal de Chagas en América Latina con Aprendizaje Automático

---

Juan Mateo De Monasterio

Director: Carlos Sarraute
Co-director: Alejo Salles
Jurados: Matthieu Jonckheere, Pablo Groisman

Diciembre 2017

This document was typeset in Latin Modern font using LaTeX.

To all.

Resumen

En la actualidad, la enfermedad del Chagas sigue siendo una epidemia de escala global. Ésta se encuentra extendida en gran medida a lo largo de todo el continente americano y con métodos de control que se concentran sobre las regiones infestadas por el vector de transmisión. Los patrones de movilidad humana representan un factor importante en la propagación geográfica de esta enfermedad. El movimiento de áreas de alta a baja infestación es fortalecida por migraciones de tipo estacionales o de larga duración. Usando datos anonimizados de llamados por telefonía celular, y en colaboración con la Fundación Mundo Sano, en este trabajo se busca evaluar la relación entre los patrones de uso celular y el de las migraciones humanas, explorando las relaciones sociales subyacentes en las interacciones de telefonía móvil. Este análisis servirá para ayudar en los esfuerzos de prevención del Chagas en Argentina y México, donde se lograron identificar posibles focos endémicos de esta enfermedad fuera de la zona de infestación vectorial. Para hacer esto, utilizamos diferentes modelos probabilísticos del subcampo de Aprendizaje Automático. Para cada usuario más de 150 atributos fueron extraídos de los datos y analizados para evaluar la probabilidad de que el usuario haya migrado desde la zona endémica. Los resultados aquí expuestos sirven para discriminar aquellos atributos más relevantes en la detección de estos movimientos, especialmente en usuarios de fuertes lazos sociales con la región endémica, de alta movilidad y en usuarios con un alto grado de movilidad. A través de visualizaciones geográficas, logramos agregar estas relaciones sociales para mostrar locaciones tradicionalmente no endémicas que potencialmente sean focos endémicos y que aún no son reconocidos como tales.

Abstract

The Chagas disease continues to represent a global epidemiological problem, particularly for the South American continent. Current control methods focus solely on the vector-infested regions. Still, human mobility patterns represent an important factor in the geographical spread of this disease, since its dissemination from high to low infested regions is strengthened by seasonal and long-term migrations.

Using anonymized mobile phone data, and in collaboration with the Mundo Sano Foundation, the objective of this work is to assess the relationship of calling patterns and user behavior, with migrations. This analysis, in turn, helps for Chagas prevention efforts in Argentina and Mexico, where we identified possible endemic foci outside of vector-infested regions.

To do this, we evaluated different Machine Learning techniques as probabilistic models. More than 150 features were extracted from the data and related to the probability of having lived or moved from an endemic region.

The results here presented identify key features for the detection of these movements, especially in users with strong ties to the endemic regions or with high mobility patterns. By presenting geographic visualizations of social ties, we tag locations outside the endemic region with a hypothetical higher prevalence rate.

Contents

Chapter 1

Introduction

Chagas disease is a tropical parasitic epidemic of global reach, spread mostly across 21 Latin American countries. The World Health Organization (WHO) estimates more than six million infected people worldwide [World Health Organization, 2016]. Caused by the *Trypanosoma cruzi* parasite, its transmission occurs mostly in the American endemic regions via the *Triatoma infestans* insect family. It is also known as "kissing bug" and "vinchuca" in the local variation for Argentina, Bolivia, Chile and Paraguay. In Central America the bug is more commonly known as "chinche". In recent years and due to globalization and migrations, the disease has become an issue in other continents [Schmunis and Yadon, 2010], particularly in countries that receive Latin American immigrants such as Spain [Navarro et al., 2012] and the United States [Hotez et al., 2013a]. As a consequence this disease has become a global health problem.

A crucial characteristic of the infection is that it may last 10 to 30 years in an individual without being discovered [Rassi and de Rezende, 2012], which greatly complicates effective detection and treatment. About 30% of individuals with chronic Chagas disease will develop any type of symptoms which include life-threatening cardiomyopathies or gastrointestinal disorders. Long-term human mobility, particularly seasonal and permanent rural-urban migration, play a key role in the epidemic spread [Briceño-Leon, 2009]. Other relevant routes of transmission include blood transfusion, congenital contagion –with an estimated 14,000 newborns infected each year in the Americas [OPS, 2006]–, organ transplants, accidental ingestion of food contaminated by *Trypanosoma cruzi*, and laboratory accidents.

The spatial dissemination of a congenitally transmitted disease sidesteps the available measures to control risk groups, and shows that individuals who have not been exposed to the disease vector should also be included in detection campaigns. To the best of our knowledge, current studies on human migration patterns in Latin America can provide only coarse and outdated information on user flows at specific municipal levels, without national territory coverage.

In this work, we discuss and explore the use of mobile phone records –also known as Call Detail Records (CDRs)– for the analysis of mobility patterns and the detection of possible risk zones of Chagas disease in two Latin American countries. We rely on key health expertise on the subject, provided by the *Mundo Sano* Foundation. To do this, we will explore the data and rely on a set of different *Machine Learning* tools. In the remainder of this chapter we will give some key aspects of Chagas disease, and introduce some particular aspects of the countries of our study: Argentina and Mexico. We will also give a very brief overview of the field of *Machine Learning*.

1.1 Mobile Phone Records, Mobility and Epidemiology

The scientific analysis of mobile phone datasets, collected by operators within their network, is a recent field of study, with a corpus of published works starting in 2005, and a noticeable increase of publications from 2012 onward, as reported in the survey of [Naboulsi et al., 2015]. Other milestones of this increased interest of the scientific community are the NetMob conferences, focused on mobile phone data analysis (see [NetMob, 2016]).

Mobile traffic contains information about the movement, interactions, and mobile service consumption of individuals at unprecedented scales. This attracted scientists from multiple disciplines: sociologists, epidemiologists, physicists, transportation and telecommunication experts found in these datasets a clear opportunity to bring their analyses to an unprecedented scale while retaining a high level of detail on each individual. Thanks to the growing availability of datasets, collaborations between academic research groups and network operators based on the analysis of real-world mobile traffic have been flourishing. A significant example is the Data for Development Challenges by Orange [Orange D4D, 2016] in which CDRs from the telecommunication operator was provided. This let international research laboratories compete under an innovation challenge whose objective is to contribute to social welfare.

In CDR analysis one of the main research subjects is mobility analysis, where calls can be located by means of knowing cellphone tower geolocalization and by then matching calls to one of these points. This provides fine granularity projections of users trajectories and in this way users have been found to show a strong regularity in their movement patterns, both in space and time, as described in [Gonzalez et al., 2008].

In another study, [Song et al., 2010] explore the limits of predictability in human dynamics by studying the mobility patterns of anonymized mobile phone users. By measuring the entropy of individuals trajectories, the authors found a 93% potential predictability in user mobility across the whole user base.

Human mobility is also influenced by social components: [Cho et al., 2011] observe that social relationships can explain about 10% to 30% of all human movement, while periodic behavior explains 50% to 70%; whereas [Ponieman et al., 2016] show how social ties can be used to improve mobility predictions, for the case of people

attendance to large social events.

Mobility analysis has applications in diverse areas such as urban planning (see [Wang et al., 2012]) and disaster recovery (see [Lu et al., 2012]). In particular, mobile phone records contain information about the movements of large subsets of the population of a country, and make them very useful to understand the spreading dynamics of infectious diseases, while preserving the privacy of users.

For instance, CDRs have been used to understand the diffusion of malaria in Kenya by [Wesolowski et al., 2012] and in Ivory Coast by [Enns and Amuasi, 2013], including the refining of infection models performed by [Chunara and Nsoesie, 2013]. The cited works on Ivory Coast were presented using the "Data for Development" ([Orange D4D, 2016]) challenge datasets released in 2013. [Tizzoni et al., 2014] compare different mobility models using theoretical approaches, available census data and algorithms based on CDRs interactions to infer people movements. They found that the models based on CDRs and mobility census data are highly correlated, illustrating CDR use as mobility proxies.

Mobile phone data has also been used to predict the geographic spread and timing of Dengue epidemics by [Wesolowski et al., 2015]. This analysis was performed for the country of Pakistan, which is representative of many countries on the verge of countrywide endemic dengue transmission. Other works directly study CDRs to characterize human mobility and other sociodemographic information. A complete survey of mobile traffic analysis articles may be found in [Naboulsi et al., 2015], which also reviews additional studies based on the Ivory Coast dataset mentioned above.

In this work we intend to show how these data can help in the problem of Chagas disease, by means of accurately describing people migrations and movements at a national level. We expect to add more relevant information to the problem of detecting *foci* of communities with high rates of prevalence, out of the endemic region.

It is relevant to the purpose of this work to note that specifically tagging disease-carriers is out of scope. We have no reason to believe that this dataset can provide such information with accuracy and we have not found any data sources to cover this missing information.

1.2 Machine Learning

Machine learning is a sub-field of computer science with broad *theoretical* intersections with statistics and mathematical optimization. At present time, it has a wide range of applications. A non-comprehensive list of its uses includes self-driving cars, Spam detection systems, face and voice recognition, temperature prediction in weather, AI opponents in games, disease detection in patients, text and audio language translation, stock pricing, movie recommendation systems, etc. Examples of these Machine Learning programs are now widespread to the point where their use today has a direct impact on the daily lives of millions of people. Due to this, Machine

Learning has *practical* intersections with data and software engineering.

[Mitchell, 1997] gives the following definition of Machine Learning: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E".

For our purposes it is clear though that this definition[1] is not a mathematical one. However it serves to convey the idea of algorithms that automatically *learn* to do a specific task better over time and with more data. Note that the "goodness" of their performance is inherently subjected to the evaluation criteria chosen for the task. Because of this, the concept of *learning* within this context is less associated with the cognitive definition and more related to an optimization based approach.

In this thesis we break down our purpose of work into different tasks to generate predictions of population movements between regions. We give a broad presentation of tools that are necessary to carry out this task and, additionally, we give an introduction to the field's vocabulary, theory and practical notes on how it can be used for this kind of problems.

We later start solving the tasks as we introduce a set of supervised classification algorithms. All along this work, the idea will be to exemplify the concepts as much as we can with actual work on the CDR data. Also, we will try to assess what are the best attributes that can be extracted from the data for the prediction tasks.

The problem of long-term human migrations will provide a proxy to understand the spread of Chagas disease. This work's objective is to show that geolocalized call records are rich in social and individual information, which can in turn be used to determine whether an individual has lived in an epidemic area. We present two case studies, in Argentina and in Mexico, using data provided by mobile phone companies from each country. To the best of our knowledge, this is the first work that leverages mobile phone data to better understand the diffusion of the Chagas disease.

1.3 Key Facts on Chagas Disease

1.3.1 Chagas Disease in Argentina

For more than 50 years, vector control campaigns have been underway in Argentina as the main epidemic counter-measure. The *Gran Chaco*, situated in the northern part of the country is home to the disease-carrying triatomines. This region is hyper-endemic for the disease [OPS, 2014]. A map of this ecoregion is shown in Figure 1.3.1.

The ecoregion's low socio-demographic conditions further support the parasite's life-cycle, where domestic interactions between humans, triatomines and animals foster the appearance of new infection cases, particularly among rural and poor areas. This region is considered as the endemic zone E_Z in the analysis described in

1. Other authors might reference to Machine Learning as *statistical learning*. See [Hastie et al., 2009] as an example.

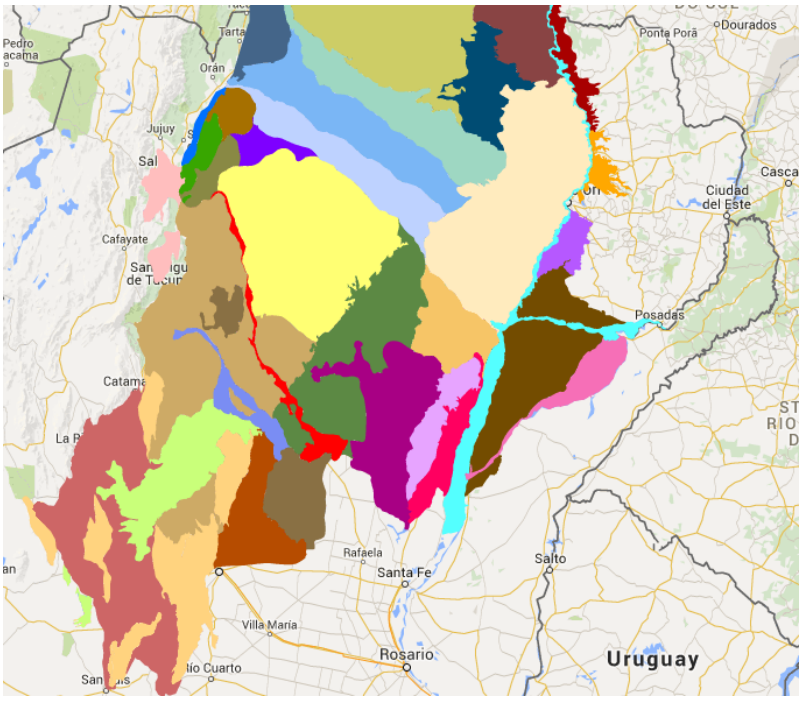


Figure 1.3.1 The *Gran Chaco* ecoregion in South America.

Chapter 2 and Chapter 3.

The dynamic interaction of the triatomine infested areas and the human mobility patterns create a difficult scenario to track down individuals or spots with high prevalence of infected people or transmission risk. Available methods of surveying the Chagas disease state in Argentina are nowadays limited to screenings of individuals.

Recent national estimates indicate that there exist between 1.5 and 2 million people carrying the parasite, with more than seven million exposed. National health systems face many difficulties to effectively treat the disease. In Argentina, less than 1% of infected people are diagnosed and treated (the same statistic holds at the world level).

Even though governmental programs have been ongoing for years now [Ministerio de Salud Argentina, 2016], data on the issue is scarce or of difficult access. This presents a real obstacle to ongoing research and coordination efforts to tackle the disease in the region.

1.3.2 Chagas Disease in Mexico

In 2004, the joint work of *Instituto Nacional de Cardiología "Ignacio Chávez"* and *Instituto de Biología de la UNAM* resulted in a Chagas disease database for Mexico ([Cruz-Reyes and Pickering-Lopez, 2006]). Reviewing positive serology in blood banks and human reported cases per state, an epidemic risk map description was produced to geographically situate the disease. Based on this data, we defined the

Mexican epidemic area, selecting the states having the top 25% prevalence rates nationwide. The resulting risk region is shown in Figure 1.3.2. It covers most of the Southern region of the country and includes the states of Jalisco, Oaxaca, Veracruz, Guerrero, Morelos, Puebla, Hidalgo and Tabasco. This region is considered as the endemic zone E_Z for the Mexican case in the analysis described in Chapter 2 and Chapter 3.

Figure 1.3.2 Endemic region E_Z for Mexico.

The authors of [Carabarin-Lima et al., 2013] provide an extensive review of the research reports on Chagas disease in Mexico. The review is very critical, stating that there are no effective vector control programs in Mexico; and that the actual prevalence of the disease can only be estimated because no official reporting of cases is performed.

According to [Dumonteil, 1999], there are a total of 18 endemic areas in Mexico, located in the southeast, and these areas include the states of Oaxaca, Jalisco, Yucatán, Chiapas, Veracruz, Puebla, Guerrero, Hidalgo, and Morelos, all of them with rural areas. Chiapas, Oaxaca, Puebla, Veracruz and Yucatán are among the most affected states (where the prevalence may exceed 10%), although cases have been reported in most areas of the country ([Cruz-Reyes and Pickering-Lopez, 2006, Dumonteil, 1999]). Despite the lack of official reports, an estimate for the number of *Trypanosoma cruzi* infections by state in the country indicates that the number of potentially affected people in Mexico is roughly 5.5 million (see [Carabarin-Lima et al., 2013]). Mexico, together with Bolivia, Colombia, and Central America, are among the countries most affected by this *neglected tropical disease* (NTD), as reported by [Hotez et al., 2013b]. For what we know, nowadays the disease spreads across borders: Chagas and other neglected tropical diseases present in the north of Mexico remain highly endemic in the south of Texas as well ([Hotez et al., 2012]).

In recent years there has been a focus on treating the disease with two available medications, benznidazole or nifurtimox. A study that explores the access to these two drugs in Mexico shows that less than 0.5% of those who are infected with the

disease received treatment in Mexico in years ([Manne et al., 2013]).

People from endemic areas of Chagas disease tend to migrate to industrialized cities of the country, mainly Mexico City, in search of jobs. In accordance with this movement, [Guzman-Bracho, 2001] showed that infected children under 5 year of age are frequently distributed in urban rather than in rural areas, indicating that the disease is becoming urbanized in Mexico. Therefore, as in the Argentinian case, the study of long-term mobility is crucial to understand the spread of Chagas disease in Mexico.

1.4 Thesis Structure

Chapter 2 will present the data sources and give the main idea behind how we process the data. It will also show some basic descriptive aspects of the information that can be found in the CDR logs and how it compares to other data sources. Finally, the chapter will develop the methodology used to construct chagasic national disease risk maps from the data logs. Examples of maps will be shown to illustrate the results found.

In Chapter 3 we will give a more in-depth introduction to Machine Learning theory, describing use cases, ideas and concepts from this field. Our long-term migration prediction tasks will be formulated in a way that can be used in a binary classification problem that can be later input to a statistical model. Added to this, we will introduce Machine Learning theory along with examples of applications on the processed dataset. We will use various examples with the Logistic Regression classifier in order to illustrate the concepts involved.

Chapter 4 will further expand concepts from the field of Machine Learning that will be used throughout our analyses. We will also give an overview of different strategies to select the best statistical models.

Chapter 5 will concentrate on specific tree-based algorithms for Supervised Machine Learning Classification tasks. We will introduce the minimal formulations for the models, as well as their applications. For each of these, we will dive into their main benefits and drawbacks, and we will compare all of them under a common classification problem. Finally, along with the presentation of each method, we will show specific long-term human migration experiments on the CDR data, For each case we will show the results of these applications and the discoveries made. Additionally, we will introduce the Naive Bayes as a model on which to benchmark.

We close this work with a summary of results in Chapter 6 where key information from the previous chapters' output is recollected. We then give this work's conclusions in Chapter 7, along with drawbacks from the methodology used and possible lines of future work.

Chapter 2

Data Description and Risk Maps

This chapter will give a technical explanation of what raw Call Detail Records (CDR) are and how they will be used throughout this work. The preprocessing and manipulation of the data will be explained in depth, with all the details necessary to build the epidemic risk maps that are presented in this chapter. All necessary vocabulary and definitions will be introduced, along with the ideas used to build the user-level covariates from the CDRs. Finally, some descriptive facts on both datasets will be given, concerning data size, volume of transactions, feature correlations and a comparison of telecommunication (TelCo) user distribution with national population distributions.

2.1 Mobile Phone Data Sources

Our data source is anonymized traffic information from two mobile operators, in Argentina and Mexico. Both companies service phone calls at the national level and to a number of customers that amounts to a user base that sizes in the order of millions. All of this information is contained in two different CDR datasets.

We make an important remark here that in the data the users' privacy is ensured by not providing a direct identifier of the persons. All users are distinguished with their *user_id* which is information given by the TelCo with a salted hash transformation. In this way, we are unable to use these identifiers in other data sources other than the specific one used for this work.

For our purposes, each record is represented as a tuple $\langle i, j, t, d, l \rangle$, where user i is the caller, user j is the callee, t is the date and time of the call, d is the call direction (incoming or outgoing, with respect to the mobile operator client), and l is the location of the tower that routed the communication.

The dataset does not include personal information from the users, such as name, phone number, home address, etc. The users privacy is assured by differentiating users by their salted and hashed ID, where encryption keys were managed exclusively

by the telephone company.

All of the data was preprocessed excluding users whose monthly cellphone use either did not surpass a minimal number of calls μ or exceeded a maximal number M. This ensures we leave out outlying users such as call-centers or dead phones and, for both datasets, we used $\mu = 5$ and $M = 400$.

We then aggregate the call records for a five month period into an edge list $(n_i, n_j, w_{i,j})$ where nodes n_i and n_j represent users i and j respectively and $w_{i,j}$ is a boolean value indicating whether these two users have communicated at least once within the five month period. This edge list will represent our mobile graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$ where $\mathcal{N}$ denotes the set of nodes (users) and $\mathcal{E}$ the set of communication links. We note that only a subset $\mathcal{N}_C$ nodes in $\mathcal{N}$ are clients of the mobile operator, the remaining nodes $\mathcal{N} \setminus \mathcal{N}_C$ are users that communicated with users in $\mathcal{N}_C$ but themselves are not clients of the mobile operator.

Since geolocalization information is available only for users in $\mathcal{N}_C$, in the analysis we considered the graph $\mathcal{G}_C = \langle \mathcal{N}_C, \mathcal{E}_C \rangle$ which results to communications only between clients of the operator.

Datasets Information. The Argentinian dataset contains CDRs collected over a period of 5 consecutive months. The raw data logs contain, in average, more than 65 million calls per day. The calls were placed through a network of over 4000 geolocalized antennas. In total, data amounts to almost 10 billion calls.

The Mexican data source is an anonymized dataset from a national mobile phone operator. Data is available for every call made within a period of 24 consecutive months and the raw logs contain at least 11 and at most 47 million daily calls. Each day, up to 8 million users accessed the telecommunication company's (*TelCo*) network. Note that users from other companies are logged, as long as one of the users registering the call is a client of the operator. Again, we only considered CDRs between users in $\mathcal{N}_C$ since geolocalization was only possible for this group.

Information logged for each call included other aspects of the interaction. We had data for the duration and timestamp of the call, the users participating in the call and the antenna id that transmitted the call to the *TelCo* client.

Adding all data elements, the project involved working with more than 1.5 terabytes of data. The data was compressed using *gzip* format. The call logs, in this data format, were parsed, processed, transformed and loaded to create the datasets which were built as needed by this project.

Data Limitations. Although a lot of information is available in the CDR datasets, there may be limitations in their representations of the whole national population. In both cases, the data sources from a single mobile phone operator, and no information is given on the spatial distribution of its users, relative to the national average. In principle, we do not know if TelCo's users are over- and under-represented across national jurisdictions, in comparison to the nation's national population distribution. Therefore, calls might not accurately represent social interactions and movements

between two given jurisdictions, given that we are provided with a biased sample of users.

The same argument can be used to note that not all *real* population movements will be captured by the logs. However, these limitations are offset by the huge datasets' sizes, from which we think we can safely assume that the amount of users observed in each set is sufficient to correlate CDR usage with human mobility or social links between different areas.

2.2 Risk Maps Generation

In this section we describe how the Chagas disease risk maps were built for Argentina and Mexico, and we give an overview of their uses. As an overview, we use the risk maps to hypothesize on the possibility of locating specific communities of higher disease prevalence outside of the endemic regions. They are based on the assumption that when we have stronger communication ties from one community to the endemic region, we will find a higher chance of disease risk. Earlier versions of this project were presented by [Sarraute et al., 2015b] at *Simposio Internacional sobre Enfermedades Desatendidas*, and [de Monasterio et al., 2016] at *International Conference on Advances in Social Networks Analysis and Mining*. These were based on the results contained in this chapter.

The generated heatmaps display a geographical representation with the TelCo's antennas situated on the map. For each antenna, a circle is drawn which represents, graphically, the volume of usage of that antenna and the vulnerability of its users. We will expand on this last concept later by analyzing the importance of this variable in the detection of long-term migrations.

2.2.1 Home Detection

In order to build the risk maps, the first step is to infer the antenna in which each user lives. The antenna's neighborhood will define their home area of influence and with this, their risk condition associated with being inside or outside the endemic region.

Having the geolocalized data granular at the antenna level, we can match each user $u \in \mathcal{N}_C$ with their *home antenna* H_u. To do so, we assume H_u as the antenna in which user u spends most of the time during weekday nights. This, according to our categorization of weekday's types, corresponds to Monday through Thursday nights, from 8 pm to 6 am of the following day. Note that this home antenna analysis will not precisely locate users yet it will define their area of influence. This will enable us later to detect long range migrations by looking at changes in these home antennas.

This *home* characterization is based on the assumption that on any given day, users will be located at home during night time. The implications of this assumption for CDRs are explored by [Sarraute et al., 2015a] and by [Csáji et al., 2012]. There, the authors explain that given the large user base, this assumption proves helpful when trying to predict migration patterns at large scale. For our case we need not

detect specific agent movements but are more interested in movement of large amount of people.

Given a user's H_u,they will be considered endemic if their inferred home antenna is located in the endemic zone E_Z. These tagged users are considered to be the set of *residents of* E_Z. In the case of Argentina, the risk area is the *Gran Chaco* ecoregion, as described in Section 1.3.1; whereas in the case of Mexico, we used the region described in Section 1.3.2.

2.2.2 Detection of Vulnerable Users

Given the set of inhabitants of the risk area, we want to find those with a high communication pattern with residents of the endemic zone E_Z. To do this, we get the list of calls for each user and then determine the set of their neighbors in the social graph $\mathcal{G}_C$. For each resident of the endemic zone, we tag all their graph neighbors as potentially vulnerable. We also tag the calls to (from) a certain antenna, or *cell*, from (to) residents of the endemic area E_Z as *vulnerable calls*. With our definition, a user with at least one call from (to) the endemic area is enough to qualify as vulnerable.

The next step is to aggregate this data for every antenna. Given an antenna a, we will have:

- The total number of residents N_a (this is, the number of people for which a is their home antenna).
- The total number of residents which are vulnerable V_a.
- The total volume of outgoing calls C_a from every antenna.
- From the outgoing calls, we extracted every call that had a user whose home is in the endemic area E_Z as a receiver VC_a (*vulnerable calls*).

These four numbers $\langle N_a, V_a, C_a, VC_a \rangle$ are the properties we extracted for each antenna in the studied countries.

2.2.3 Heatmap Generation

With the collected antenna properties, we generated heatmaps to visualize the mentioned antenna indicators, overlapped with political maps of the corresponding region. In them, a circle is generated for each cell, where:

- the **area** depends on the number of TelCo users living in the antenna N_a.
- the **color**, is related to the fraction V_a/N_a of vulnerable users living there.

We used two filtering parameters to control which antennas are plotted.

- β: The antenna is plotted if its fraction of vulnerable users is higher than β.
- m_v: The antenna is plotted if its population is bigger than m_v.

When displaying the maps, these parameters were helpful to identify special small antenna's circles which were covered under the circles of larger volume antennas. Having more than four thousand antennas for each plot means that a high number of these would graphically overlap with small neighboring ones. This *noise* is unhelpful for the detection of smaller, more vulnerable, antennas.

These parameters were tuned differently when zooming into different regions. For example: an antenna whose vulnerable percentage would be considered low at the national level can be locally high when zooming in at a more local level. The filtering parameters helped us explore these cases in more detail, and provided great insight to the Mundo Sano Foundation. The following section presents these maps.

2.3 Risk Maps Visualizations

2.3.1 Risk Maps for Argentina

As a first visualization, maps were drawn using a provincial or national scale. Advised by *Mundo Sano* Foundation's experts, we then focused on areas of specific epidemic interest.

Figure 2.3.1 shows the risk maps for Argentina, generated with two values for the β filtering parameter, and fixing $m_v = 50$ inhabitants per antenna. After filtering with $\beta = 0.15$, we see that large portions of the country harbor potentially vulnerable individuals. Namely, Figure 2.3.1 (b) shows antennas where more that 15% of the users has social ties with the endemic region E_Z.

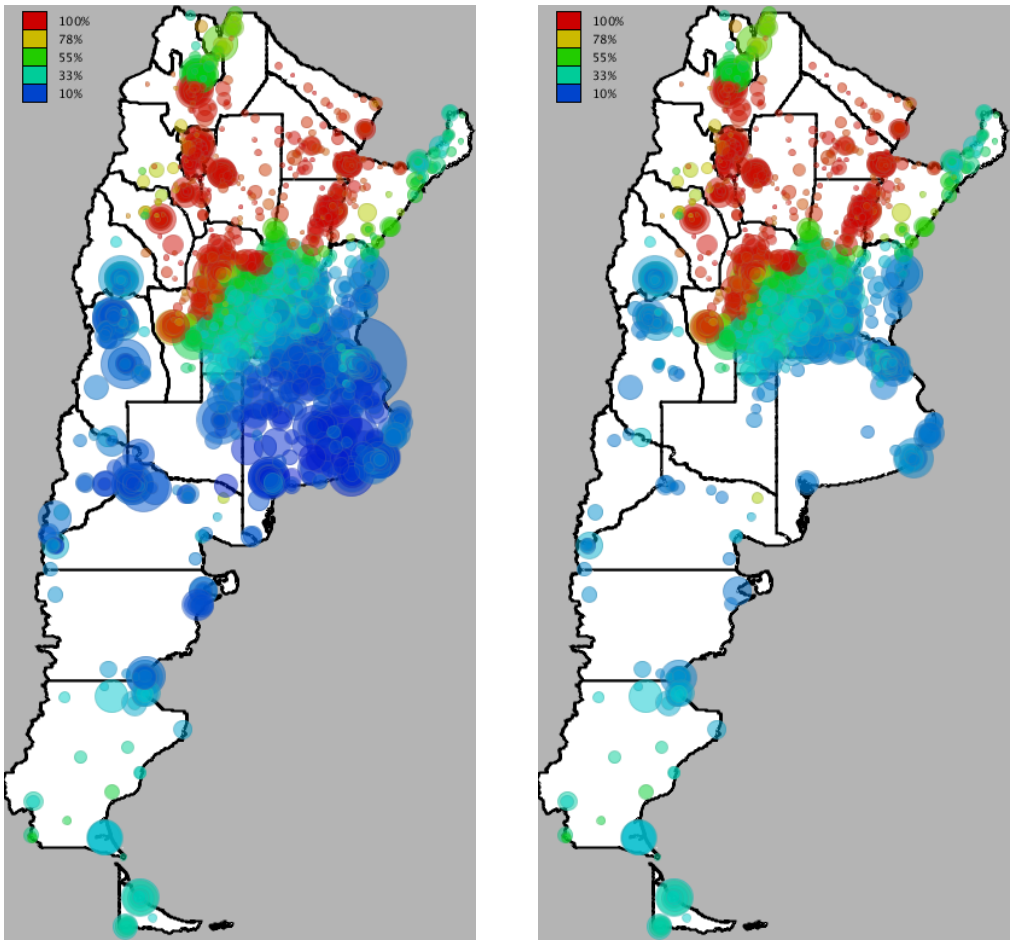
Figure 2.3.2 shows a close-up for the Cordoba and Santa Fe provinces, where we can see a gradient from the regions closer to the endemic zone E_Z to the ones further away.

2.3.2 Detection of Vulnerable Argentinean Communities

As a result of inspecting the maps in Figure 2.3.1, we decided to focus visualizations in areas whose results were unexpected to the epidemiological experts. These areas included the provinces of Tierra del Fuego, Chubut, Santa Cruz and Buenos Aires, with special focus on the metropolitan area of Greater Buenos Aires whose heatmap is shown in Figure 2.3.3.

In some cases, antennas stood out for having a significantly higher link to the epidemic area than the adjacent ones. Our objective here was to enhance the visualization in areas outside of Gran Chaco looking for possible host communities of migrants from the ecoregion, inferred by the social links shown in the CDRs. Our assumptions were that if, in average, there was a higher percentage of vulnerable users in that non-epidemic antenna, this would mean that there's a high possibility of having more infected users in that community.

As an outcome of analyzing these visualizations, we were able to locate special antennas which stand out over their neighboring ones. These high risk antennas were then located and separated for manual inspection along with the *Mundo Sano*



(a) $\beta = 0.01$ (b) $\beta = 0.15$

Figure 2.3.1 Risk maps for Argentina, filtered according to the value of β.

Foundation collaborators. They used this information it as an aid for their campaign planning and as education for community health workers.

This data exploration allowed us to specifically detect outlying communities in the focused regions. Some of these can be seen directly from the heatmap in Figure 2.3.3, where the towns of Avellaneda, San Isidro and Parque Patricios have been pinpointed.

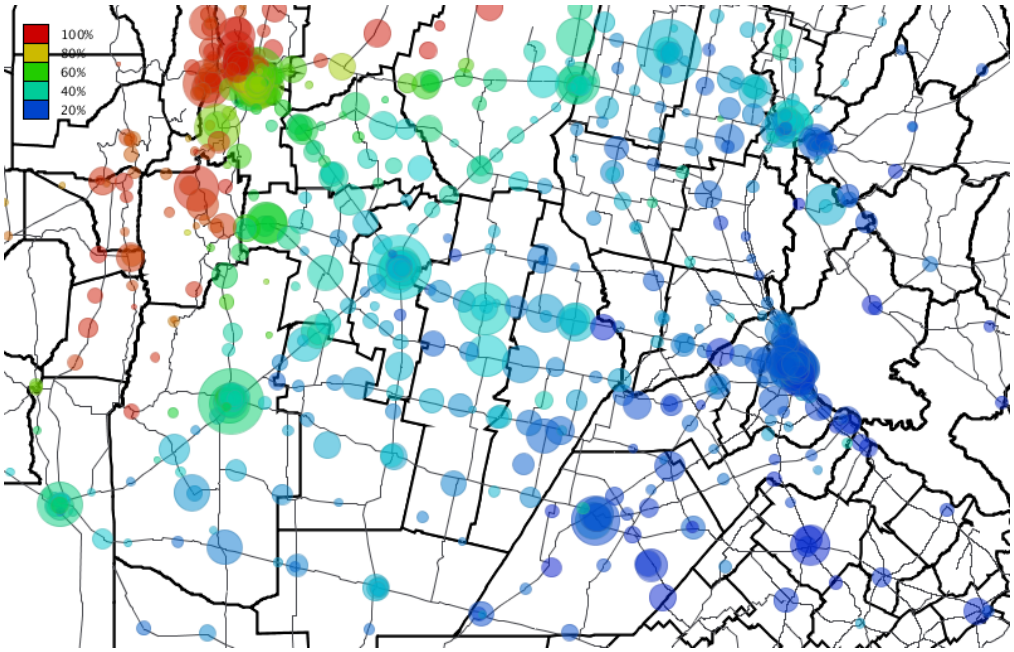


Figure 2.3.2 Risk map for Cordoba and Santa Fe provinces, filtered according to $\beta = 0.15$.

2.3.3 Risk Maps for Mexico

With the data provided by the CDRs and the endemic region defined in Section 1.3.2, heatmaps were generated for Mexico using the methods described in Section 2.2.1. The first generated visualizations are depicted in Figure 2.3.5, which includes a map of the country of Mexico, and a zoom-in on the South region of the country. We used $m_v = 80$ inhabitants per antenna, and a high filtering value $\beta = 0.50$, which means that in all the antennas shown in Figure 2.3.5, more that 50% of inhabitants have a social tie with the endemic region E_Z. For space reasons, we don't provide here more specific visualizations and analysis of the regions of Mexico.

2.4 Data Features

The quality of any Machine Learning task on CDRs relies heavily on the ability to characterize the users and their communication patterns, in ways that are as relevant as possible to the task. In general, the features constructed reflect calling and mobility patterns, segmented by different time periods during the week, and tagging whether the actions or subjects are 'endemic'.

Our goal in building a Machine Learning model is to evaluate and analyze if long-term migrations can be accurately predicted from the CDR data. Added to this, we would like to know what type of information extracted from the data has most predictive power to our problem's variable. This result would tell us which data features are informative in detecting migrations, which in turn, would test the

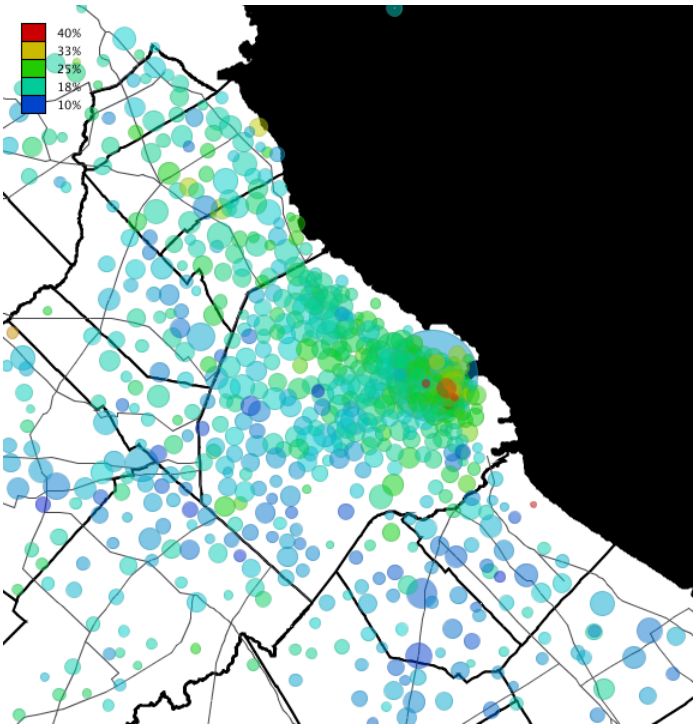


Figure 2.3.3 Risk map for the metropolitan area of Buenos Aires, filtered with $\beta = 0.02$.

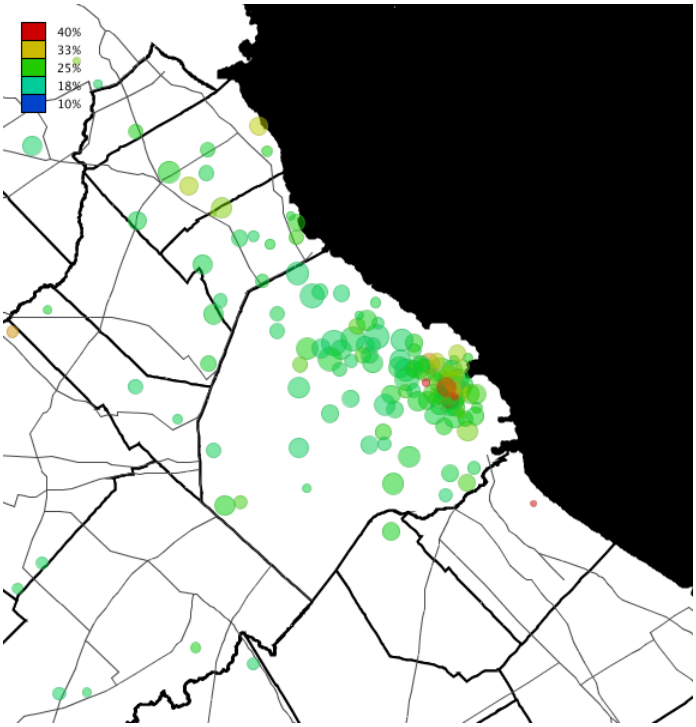
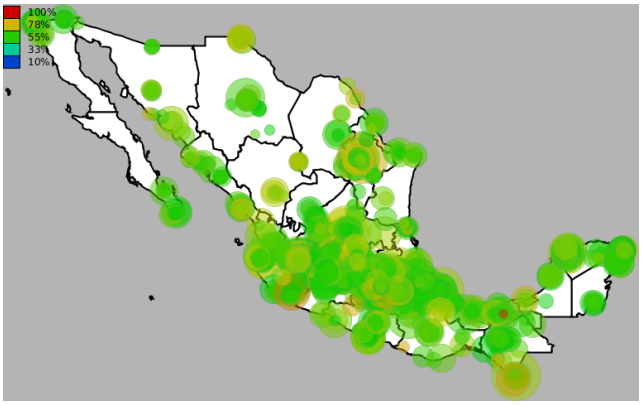
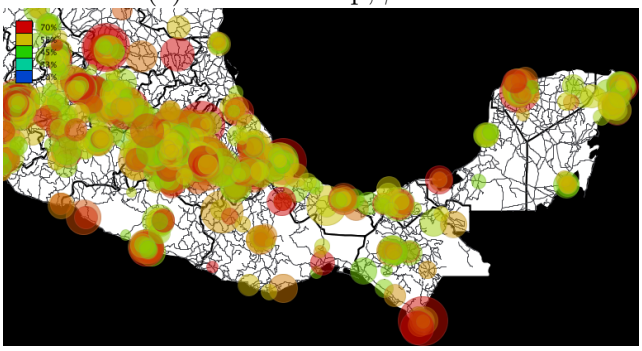


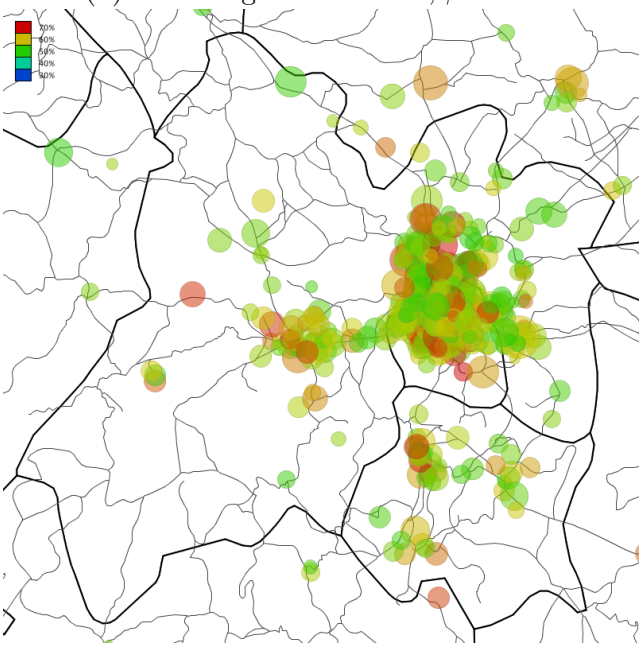
Figure 2.3.4 Risk map for the metropolitan area of Buenos Aires, filtered with $\beta = 0.2$.



(a) National map, $\beta = 0.50$



(b) South region of Mexico, $\beta = 0.50$



(c) State of Mexico, $\beta = 0.50$

Figure 2.3.5 Risk maps for Mexico, filtered according to the value of β.

assumption on which we build the risk maps in Section 2.3. The CDR logs available for Argentina span a period of 5 months, whereas the Mexican dataset includes 24 months, from January 2014 to December 2015, making it more suitable for this study.

To begin with, we divide the available data into two distinct periods: T_0, from January 2014 to July 2015, considered as the "past" in our experiment; and T_1, from August 2015 to December 2015, considered as the "present". Our dataset will only run from August 2015 to December 2015 (period T_1), where all CDRs are processed to extract information by user and by link. This is done because we can't introduce data from T_0 for a model trying to predict this same information.

To begin building attributes from the data, each week is divided into time periods:

Definition 2.4.1 (i) the period *weekday* is from Monday to Friday, on working hours (from 8hs to 20hs); (ii) *weeknight* is from Monday to Friday, between 20hs and 8hs of the following day; and (iii) *weekend* is Saturday and Sunday.

The model consists of the following features, which can be grouped into 4 categories:

a) **Used and Home Antennas:** For each user $u \in \mathcal{N}_C$, we register the top ten most used antennas, during each month of the training period, together with the number of calls made through each antenna. We tag all users having their home antenna in the epidemic region as *EPIDEMIC*. In our dataset user antennas are ordered with 0 being the most used antenna and 9 the least. We ignore those users for which we don't have ten used antennas and discard users that have no calls on weeknights[1]. We also register the most used antennas considering only calls made during the *weeknight* period, as defined in Definition 2.4.1. As explained before, a user's home antenna is defined by the most used antenna during weeknights and, with this, users were tagged as 'endemic' if their home antenna is in the endemic zone E_Z and 'exposed' if any of the ten antennas logged is in the risk area. Finally, we added the user's province by referring to their home antenna's membership.

b) **Mobility Diameter:** The user's logged antennas define a convex hull in space and the radius of this hull is taken to be the user's mobility diameter. This length is representative of the area of influence of that individual. We expect this feature to be correlated with long-term migrations. We registered the mobility diameter of each user, as the diameter in kilometers of the convex hull defined by their top 10 used antennas. Again, we generate two values for this attribute, considering (i) all antennas used and (ii) only those used during *weeknights*.

c) **Graph Data and Communications:** We look at the social graph $\mathcal{G}_C$ built from the CDRs, and the communications between nodes in $\mathcal{N}_C$. For each edge $\langle n_i, n_j \rangle \in \mathcal{E}_C$, we dive into each of their interactions, segmenting call data with different criteria. For each month and pair of users $\langle i, j \rangle$, we gather the tuple $\langle time_{ij}, calls_{ij}, direction, period \rangle$ where *time* is the sum of all calls (in seconds), *calls* is the number of calls exchanged, *direction* is a boolean variable indicating whether the calls were incoming or outgoing (from user i's point of view) and *period*

1. This is because we wouldn't be able to detect $H1_u$.

corresponds to a segmentation of the week into the periods *weekday*, *weeknight*, and *weekend*. In this sense, two users $u_i, u_j \in \mathcal{N}_C, u_i \neq uj$ are **neighbors** in the social graph if $time_{ij} > 0$.

Since the samples in our dataset are users, we have to aggregate all of these variables, by grouping interactions at the user level. Combinations of all of these different variables amount to more than 150 features per user.

To illustrate the point, in Table 2.4.1 we show a small example of how two calling features could look like, where we apply groupings and filters only on the user's calls. From here we extract the calling features by aggregating these variables up to the user's level. In practice, this means grouping either by j or i, and aggregating across all of the different components.

Table 2.4.1
An example list of features that were built from the social graph dataset. Each attribute has its code name and a description of the codes used to construct its name.

Feature name	Call/Time	Time Period	Direction of call	Endemicity	Month
Calls Weekend InVul08	Count	Placed on Weekends	Incoming	Edges with endemic neighbors only	August
Time Weeknight Out12	Sum of duration in seconds	During weekdays and on out-of-office hours	Outgoing	No endemic filtering	December

We also label each edge $\langle n_i, n_j \rangle \in \mathcal{E}_C$ if one or both users is endemic and count each user's amount of neighbors in the communication graph, as well as the endemic neighbors. This labeling defines user i as *vulnerable* whenever they have any edge with another user j who lives in the endemic region E_Z.

2.4.1 Antenna Distribution

As described before, the training data belongs to period T_1, from August 2015 to December 2015; whereas the ground truth that we use to validate the predictions belongs to T_0, from January 2014 to August 2015. Raw data logs contain between 11 million and 30 million calls per day and the volume of calls increases over the months, where most recent months have higher rates. After preprocessing and cleaning the dataset, we obtained a dataset with 1.6 million users. To create our model's input data, by means of relational database operations and algorithms that scrap the whole dataset, we transform raw CDRs into a dataset where individuals correspond to a single rows in the dataset.

To compare how this sampled population compares to country-wide distribution estimates, Table 2.4.2 shows the percentage of antennas, the population (according to INEGI census 2014) and of TelCo users per state, for the top 10 Mexican states. This table describes the similarity between the population distribution of Mexico versus the TelCo's users to highlight possible sources of bias in the statistical model.

Table 2.4.2
Table showing the Mexican distribution of antennas, total population and TelCo users by state.

State	Number of antennas	Population	TelCo users
Distrito Federal	28.2%	8.5%	20.1%
Mexico	21.2%	13.9%	23.8%
Jalisco	10.7%	6.4%	8.3%
Nuevo Leon	9.6%	4.9%	2.9%
Guanajuato	6.1%	4.8%	5.9%
Puebla	5.8%	5.3%	4.3%
Veracruz	5.4%	6.8%	4.2%
Baja California	4.3%	2.8%	1.1%
Yucatan	4.1%	1.7%	2.9%
Sinaloa	4.1%	2.5%	0.4%

As we see in Table 2.4.2, there are differences in the state distribution of TelCo users in comparison with the population distribution from census data. This unbalance is remarkably high for both the states of Mexico and the "Distrito Federal" where the TelCo has a better coverage than in other states.

Note that during this work we considered only postpaid users, i.e., users which have a monthly plan rate. This filtering was done because prepaid users have a higher churn rate, thus meaning that phone lines are not necessarily associated with one single person during the two years of analysis, making them less suitable for the purpose of this study.

2.4.2 User Migrations

We performed an analysis similar to the home antenna detection previously described in Section 1.3.2, but considering the time period T_0 (from January 2014 to July 2015), in order to determine the home antenna of users during T_0.

The number of people which maintain their home antenna between T_0 and T_1 is 1,012,416; whereas 580,425 users had a change in their home antenna. In terms of endemic condition, we observed that 1,551,560 users maintained their endemic condition, whether it be positive or negative, between T_0 and T_1, whereas 41,281 had a change.

Table 2.4.3 shows the matrix of changes C, such that $C_{i,j}$ is the number of users that were in group i during period T_0 (the past) and moved to group j during the training period T_1. As an example, lower left means was endemic, is now not endemic.

In relative numbers, this shows that only 2.59% of users had a change in their endemic condition over time. A similar count results in that 66.0% of users have not changed their home antenna from T_0 to T_1 and that approximately 5% of past endemic users moved into non-endemic during this time period. This is not surprising given that we wouldn't expect a large number of people migrating in a time lapse of only two years.

Table 2.4.3
Matrix of user endemicity change across time periods T_1 and T_2.

	Not endemic in T_1	Endemic in T_1
Not endemic in T_0	1140360	18330
Endemic in T_0	22951	411200

Table 2.4.4
Correlations of dataset attributes and a user's endemic condition in T_0.

Feature	Correlation
Endemic in T_0	1.0
Endemic in T_1	0.933801
Call Count OUT 08/2015	0.730294
Vuln. calls 08	0.732498
Vuln. calls 09	0.714566
Vuln. calls 12	0.715448
Vuln. calls 10	0.694106
Vuln. calls 11	0.694265

2.4.3 Feature Correlations

The dataset used in this work shows significant correlations between communication and mobility patterns. Some features are essentially highly correlated across time periods. Thus knowing a user's current endemicity will be highly correlated to their past endemicity since a user being endemic in T_1 is very correlated to being endemic in T_0. The same property extends to attributes of user's interaction with vulnerable neighbors during T_1, where their relationship to the user's endemicity in T_0 is expected. In these cases, it is important to know the value of their correlation to the past endemicity of the user. In Chapter 5, we will leverage this observation to improve our performance in predicting long-term migrations.

Table 2.4.4 quantifies these relationships, where only features with an absolute correlation value higher than 0.25 are shown. It is notable that the correlation is slightly increasing as we get closer to the split month i.e. the month chosen to separate T_0 from T_1. This behavior might be an indicator of seasonality in the data, where events from T_1 more closer to T_0 carry more intrinsic value. Added to this, there is a very small indication that a user's calling volume in the month of December is more related to a user's past endemicity in the sense that it carries a higher correlation with their past endemicity. Again, this suggests data seasonality in the data, where December is a festive month by tradition in Latin America.

Chapter 3

Machine Learning

This chapter will present an elementary introduction to Machine Learning where the concepts and ideas developed here will be exemplified on concrete applications of the processed CDR data, using a Logistic Regression classifier. Also, the chapter will formulate the tasks that will be solved in later chapters, as means of casting the prediction of the long-term human migrations as a supervised classification problem.

Most of the discussions here are based on information extracted from two distinguished textbooks: [Bishop, 2006] and [Hastie et al., 2009]. Some fundamental concepts and material were also referenced from [Pedregosa et al., 2011]. When other texts were used, they have been cited correspondingly in the text, and, where not specified, the reader can assume that the proof is given in the textbooks above mentioned.

3.1 Overview of Supervised Classification Applications

Machine Learning is divided in two main categories: Supervised and Unsupervised Learning. If we consider $Y \in \mathbb{R}^n$ be the output vector of a model and $X \in \mathbb{R}^{n \times p}$ to be the input matrix, supervised learning algorithms produce outputs from input data i.e. for each instance $x \in X$, the computer has access to examples of outputs, $y \in Y$, and tries to reproduce them based on information contained in X. In this context, the algorithm is generally referred to as a *learner*. The second class of problems is where there is no output Y in the data. For this scenario the most common objectives are clustering samples, estimating densities and compressing data.

In supervised learning, tasks are sub-categorized by the nature of the problem. If the type of the output variable Y takes a (generally *small*) discrete set of values, then it is said that it is a supervised *classification* problem. On the other hand, when the output takes continuous (or dense in an open set of $\mathbb{R}$) range of *quantitative* values, it is said to be supervised *regression* problem. Note that regression problems can be encoded into classification problems by grouping the output values into categories in

which each takes a range of output values.

Suppose now that aim is to predict y given a new sample x. We denote this prediction as $\hat{y}$ and note that for supervised regression problems, y will comprise a continuous variable. On the other hand for classification problems y will represent a label for a certain class. For the case of K classes, we can say that y takes values in the ranges 0 through $K-1$ or 1 through K. In both cases, the joint probability distribution $p(X, Y)$, called the *true* distribution, gives all the information we need on these two variables. Yet the characterization of this probability is most often unknown.

Noting this, the idea is to use estimations and predictions on the *most likely* values $\hat{y}$ for new samples and take decisions based on past information. These decisions will be based on a probabilistic characterization of the problem, from the data we have. In this work we will focus on the classification aspect of Machine Learning.

We also note that when dealing with these problems, the theoretical and the computational aspects are both of interest. It is reasonable to say that the data, taken as an input to the problem, limits the available solutions and the same goes for the algorithms used in the solution process. These methods need to take into consideration aspects such as the software and hardware available, as well as time or resource constraints imposed by the problem itself. As such, these are expected to be executed in a reasonable amount of time, as established by the task's specification, and limited by the computing power available.[1] There can be problems which require that the algorithm outputs predictions in *real-time*, to the resolution of milliseconds.

For example, if we picture a system where credit-card transactions need to be approved or labeled as a fraud, we would expect the system needs to quickly respond if the transaction is approved or not. Other use cases might require the system to process a big volume of data at once, not a single event but a batch, and produce an answer. The production system needs to be prepared to run *lean* with a big inflow of data, without exceeding the hardware capacity.

These examples show that for a given problem, there are multiple algorithms available for use, but while all of them are theoretically doing the same task, we must also consider the practical advantages. Computational efficiency and *scalability* are relevant when working in this kind of problems. Even though we won't delve into these aspects in this work, they are important in Machine Learning applications. These aspects will also drive the way research is conducted in this area.

In its essence, a Machine Learning method is a probabilistic model built from data and in this way it is very similar to a statistical model. However, it differs in that its focus is generally on the models' predictive abilities more than in the model's parameter estimates [Breiman, 2001b]. The algorithms will be built and used for a given phenomena, to try and replicate it as best as possible without really identifying the true nature of the mechanisms behind this phenomena. As such, most applications will try to *imitate* the task's behavior rather than try to identify the

1. Here the word *reasonable* is used in a broad sense. It will depend entirely on time constraints, computational capacity, usage and other aspects of each learning application.

real system behind it.

These subtle differences in the Machine Learning approach of a problem is also reflected in the terminology used by the field. For the methods introduced here, we know that other disciplines often speak about them in different terms and this difference is certainly notable in classical statistics. We will be identifying these differences along the text and, as a start, the *dependent* Y is called the target or label and the *independent* variables, *covariates* or *input variables* are named *features* in this case.

3.2 Long-term Human Migrations Classification

Starting from the two-year CDR dataset we have described in Chapter 2, we introduce our set of tasks defined to predict the long-term human migrations problem under a supervised classification framework.

We intend to develop a set of Machine Learning methods that will aid us in further characterizing and analyzing the mobility of users between regions. These will help us establish to what extent the CDRs allow us to predict a user's past home and migration movement across both endemic and non-endemic regions. More precisely, we would like to explore the relationship of the dataset attributes based on calling patterns across regions with the movement of humans. Where possible, we would like to establish which features are most relevant for our predictors.

To begin, we need to introduce some notation to describe different sets of users. In order to do this, we will divide them into distinct sets, marked by their behavior in each time period:

Definition 3.2.1 (i) EU_1 are the users that lived there in the endemic region during period T_1 while EU_0 are the users that lived in the past (period T_0); (ii) their set complements are noted as ${EU_1}^{\complement}$ and ${EU_0}^{\complement}$ respectively and (iii) the user's home antenna in each period will be referred to as $H0_u$ and $H1_u$, following the same time-index convention as before.

Keeping the above in mind, we divide our task into four problems. These will provide different perspectives of the mobility issue and all of them together will give us a principled understanding of the problem.

It is important to note that all problems are solved with user data extracted from their communications during T_1. This means that no information from T_0 will be used as input to the statistical models.

We process the raw data to build our target variable which will be noted as Y and will be defined for every user u. The only difference among all of the problems will thus be in how the target variable is defined[2].

We present here the list of problems that will be used throughout this work:

2. In reality, certain variations of the data were considered too, where best performing features were filtered from the dataset to explore how this decreased the overall prediction ability of the algorithm.

Problem 1 We want to infer which users lived in the endemic region in the past:

$$Y_u = \begin{cases} 1 \text{ if } u \in EU_0 \\ 0 \text{ if not.} \end{cases}$$

With this definition and using the description of the features extracted from the data in Chapter 2, we observe that here the information of a user's endemicity EU_1 is used to predict their membership to EU_0. It is not surprising that knowing about a user's present endemicity is indeed highly correlate with his past endemicity, as these two variables have a 91% correlation value. This means that, overall, the present endemicity feature is a very good predictor of the target variable, causing algorithms to be heavily weighted towards this attribute. We could use in fact use this single feature as a basic algorithm to build a simple predictor. To prevent this from happening and to capture the change in endemicity for a small groups of users, we decided then to hide this feature when solving Problem 1.

Problem 2 We want to infer which users lived in E_Z in the past and then migrated:

$$Y_u = \begin{cases} 1 \text{ if } u \in EU_0 \cap {EU_1}^{\complement} \\ 0 \text{ if not.} \end{cases}$$

In this task, we have not excluded any attributes from the data since this problem does not have the feature-target correlation issue as in Problem 1. Recall from Table 2.4.3 that we have at most 23 thousand users that satisfy this condition out of more than a million users. This strong unbalance in the target class makes a much harder problem to solve than Problem 1 since these users become harder to find and the class unbalance is notably stronger. If not correctly taken into account, the error over the small class would be dominant of the overall prediction.

It is relevant to note here that we are ignoring any relevant information about the user's endemicity or current state of residence. Otherwise we would have a perfect correlation between $Y_u = 0$ and being currently endemic, or living in an endemic state.

Problem 3 We want to predict those users that migrated between regions, in any direction. This implies we need to see a change in the user's endemicity from positive to negative or vice versa:

$$Y_u = \begin{cases} 1 \text{ if } u \in (EU_0 \cap {EU_1}^{\complement}) \cup (EU_1 \cap {EU_0}^{\complement}) \\ 0 \text{ if not.} \end{cases}$$

This task is similar to Problem 2 in difficulty, yet given that the condition is slightly higher, there are more users satisfying it. The difference relies in that cross-migrations to and from the endemic regions are not as important for our study as migrations directly from the endemic region. Still, we define this task hoping to eventually find some unexpected insights.

Problem 4 This problems determines to predict those users that migrated from the endemic region, but only allowing users which are currently not endemic. This means that we will only analyze users such that $u \in EU_1^{\complement}$ as a base set:

$$Y_u = \begin{cases} 1 \text{ if } u \in (EU_0 \cap EU_1{}^{\complement}) \\ 0 \text{ if } u \in (EU_1{}^{\complement}/EU_0). \end{cases}$$

An important observation is that at this point we are looking at a reduced instance of our CDRs, because we are only considering currently non-endemic users and not the whole sample population. Still, we are observing a total of more than a million users for this case.

We can see from what was mentioned before that not all problems can be represented by the same input features from the data. Each problem has certain restrictions on what input features can be used and these limitations will be enforced by constructing variants the input set's (X) attributes. For example, we said that in Problem 1 we have no limitations to use a user's current endemicity, $u \in EU_1$, as a feature, yet we decided to not do so as it was a feature highly correlated with the target variable. Also, in Problem 2 we omit using a user's current territory of residence as this feature would leak information that is directly related to the target variable: knowing the user's current state will allow us to know directly if he has migrated out of the endemic region in the past. If we choose not to, we would be creating an ill-conditioned problem as we implicitly introduce the target variable in the form of an attribute. Taking these issues into consideration, for each problem we must carefully discard those attributes that ill-pose the models.

With Problems 1 to 4 defined, we then continue to illustrate techniques for supervised classification. Throughout Chapters 4 to 7 we will come back to these problems by applying them with the techniques and methods we discuss.

3.3 A working example of a Machine Learning setup

In our work we take advantage of the volume of samples, where, as explained in Chapter 2, amounts to almost 1.5 million users. This allows us to randomly split the dataset into training and test sets that do not overlap and which contain 70% and 30% of all samples respectively. The test set will not be a part of the model building process because we will only use it at the end of the process to evaluate the quality of the built models. No information from the test set should be accessed during the construction of the models.

We will note the training and the test sets by $\mathcal{T}$ and $\mathcal{T}_f$ respectively. Both $\mathcal{T}$ and $\mathcal{T}_f$ will take the form of a paired couple of datasets (X, Y) where $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^n$. The difference between both sets will lie in the way each is used to computationally build a probabilistic model. The test set will act as an estimation of our algorithm's performance, by allowing us to have *new* samples over which to evaluate models. This makes sense knowing that the objective is to build a

probabilistic model which has the capacity to correctly predict the output class instances for new data objects, based on having seen information of objects from $\mathcal{T}$. The samples from $\mathcal{T}_f$ will take on this role.

As an example, let's consider a reduced training dataset built from Call Detail Records (CDRs), where samples are calls being made by users who can belong to any of the following provinces: *Buenos Aires*, *Cordoba* and *Santa Fe*. Five measurements were taken from all of the observations to account for the users' number of calls and total minute duration of calls and all data was extracted from a week of logging measurements. In Table 3.3.1 we show a short overview of this dataset:

Table 3.3.1
Head of the raw CDR dataset. three-row mock example of calls.

User	CallsWeekend	TimeWeekend	CallsWeekDays	TimeWeekday	Province
BA343E	15	89	8	24	*Santa Fe*
73F169	10	121	2	98	*Cordoba*
EA23AD	12	43	5	154	*Buenos Aires*

In X a row is representing the available data processed for each user, and the columns represent the features which are the different types of measurements or information on that user. In other settings the features would be known as covariates or independent variables. It is not uncommon to represent data with rows as observations or *samples* and columns are measurements or *features* of our samples.

More specifically, the jth column of X, or equivalently the jth feature of the dataset, is denoted by X^j. In a similar way, the ith row or sample of X, is denoted by X_i, or x, when it is clear from context or when we are referring to a generic sample. A similar notation is used with the outputs, where Y_i or y will be used to denote the target of a specific sample, depending on context.

In this way, the training set $\mathcal{T}$ is used to algorithmically create a function which maps inputs to outputs, whilst the test set $\mathcal{T}_f$ will be used to measure how *well* this mapping would behave in real situations, given a way of measuring this performance.

In this particular example, even though the last column of the dataset in Table 3.3.1 is not a measurement *per se*, it provides information on each user's province of residence. With this we can map users to *classes* which conform the partition of the set of provinces.

Hereon, there are various questions one could try to answer using this dataset. Examples of problems that a Machine Learning algorithm could tackle could be:

- Predict a user's province when given information on only the first four features.
- Predict a user's number of calls made on weekends when given information on the last four features.
- Give an estimate of the probability density function of user's calls duration, during weekdays.

These questions can be shaped into the form of a target variable to define a concrete Machine Learning problem.

The first two of the problems defined in Section 3.2 are examples of a supervised learning task. For each, the Y variables or *labels* are respectively the last and first features (columns in the dataset). The first problem is a classification one since users are to be categorized according to one of the three possible provinces, whilst the second one is a regression problem for which the output could be any of a range of numerical values. The labels in classification problems are usually numerically encoded with a finite range of numbers, with $\{0, 1\}$ or $\{-1, 1\}$ being usually used in the binary class problem.

Note that we take the data as given and that we do not making any assumptions on it which means that the data was not collected previous to the problem but rather used to try to tackle it. This is common in Machine Learning applications and because of this, algorithms tend to be designed to account for this lack of structure among variables. The type of problems and questions that could be posed, then depend entirely on the available data.

The last example problem belongs to the unsupervised learning category where there is no need to have a label on the data. A priori, there are no observations in the data that serve as examples of what is an expected output i.e. samples are not tagged as correct or not since we do not have a target feature. In this setting, the question is on the structure of a specific column, namely the estimation of the true probability distribution of the calling time. As such, there is no output expected from new data.

Given that in this work we will be talking about supervised classification scenarios, we will introduce in the next section an example of such an algorithm.

3.4 A first approach with a Logistic Regression classifier

Let's suppose that we want to build a predictive model for Problem 1. A classification algorithm should then assign samples to their past endemic condition, one in which the user lived in the endemic region in the past.

We let $\{C_1, .., C_k\}$ denote the set of possible target classes (in this simple case $k = 2$). Our aim can be defined to maximize the probability of belonging to a certain class, given the phone data:

$$P(C_k|X) = \frac{P(x|C_k)P(C_k)}{P(x)}. \tag{3.4.1}$$

In this interpretation, $P(C_k)$ is known as the prior probability of belonging to that certain class and $P(C_k|x)$ as the posterior probability, given the sample data.

Our classifiers will partition the input space into decision regions R_k for which a class C_k is uniquely associated to the class' condition. It makes sense to try and *minimize* the chances of assigning samples to incorrect classes. For example, take a problem with K classes and a sample x_i, then the probability P of a correct

classification, with probability density function p, is measured by

$$\begin{aligned} P(C_k|X) &= \sum_{k=1}^{K} P(x_i \in R_k, C_k) \\ &= \sum_{k=1}^{K} \int_{R_k} p(x_i, C_k) dx \\ &= \sum_{k=1}^{K} \int_{R_k} p(C_k \mid x_i) p(x_i) dx. \end{aligned} \tag{3.4.2}$$

Observe that the measure is completely characterized by the posterior probabilities because the factor $p(x)$ is common to all integrals so we only need to maximize the posteriors. And even when $p(x)$ might not be known or accurately estimated, the algorithm can only introduce a modification in the decision regions R_k. This will have a direct effect on the probability of correctly classifying samples. We can say then that the goal of our algorithm will be to choose the best possible regions for the problem.

For our example Problem 1, we are in the binary classification problem since there are two possible output classes. So a first approximation to predict the target variable for each sample could be to build a learner f from a transformed linear combination of the input features.

Adding this assumption, we would say that for every sample

$$y \approx f(x) = h\left(\sum_j \theta^j x^j\right) \tag{3.4.3}$$

where $h(\cdot)$ is an activation function transforming the feature's linear combinations with θ, the model's parameters. Here θ is the unknown and can also be referred to as the coefficients of the problem.

For analytical convenience, a tractable transformation $h(z)$ for this task is the logistic function:

$$\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}} \tag{3.4.4}$$

The logistic function has the advantage of being smooth, well defined for all real numbers and with the image of $\mathbb{R}$ under it being $(0, 1)$. This property lends itself to reading outputs as probabilities of the target belonging to a certain class. It is also an approximation of the Heaviside step function (see Section A.1 for more details on this relationship).

If we let $t_i = \theta \cdot x_i \; \forall \, i$ be the linear decision boundary of each sample and use the logistic function as our activation, we have that for the supervised binary classification

setting the ideal situation, one in which all samples are correctly classified, appears when we have that

$$\begin{cases} \exists\, \theta \text{ such that } \forall\, i \\ \qquad\qquad\qquad t_i > 0 \text{ if } y_i = 1 \\ \qquad\qquad\qquad t_i < 0 \text{ if } y_i = 0. \end{cases} \tag{3.4.5}$$

If the goal of the model is to maximize $P(Y_i = y_i|X_i = x)\ \forall\, i$ under the assumption that we found parameters θ that satisfies Equation (3.4.5), then our algorithm would correctly classify all samples to their corresponding target values. However this situation is hardly ever the case. The common approach is to rely then on optimization procedures to minimize the amount of misclassification given by our algorithm. This would be analogous to maximizing the probability of a correct classification in equation (3.4.2).

Another benefit of using this function is that its derivatives can be put in terms of the logistic function itself, where

$$\sigma'(a) = \sigma(a)(1 - \sigma(a)) \tag{3.4.6}$$

The function is also bijective, with the inverse given by the *logit* function

$$\sigma^{-1}(z) = \log\left(\frac{z}{1-z}\right) \tag{3.4.7}$$

For the scenario characterized in Equation (3.4.3), we would have to finally assign each output to a specific class. A common approach for this is to categorize each output whether $\hat{y} > 0.5$. Notice that having $h(x \cdot \theta) = 0.5$ implies that $x \cdot \theta = 0$ and thus our classifier is separating samples in feature space (the space of the inputs) with the hyper-plane characterized by parameter θ.

Having a higher $\hat{y}$ for a given sample implies that it is further away from the hyper-plane and the same goes for low predicted ($\hat{y}$) target values. If we were to read this as a probability, we can interpret that the algorithm is modeling the posterior probability $P(Y_i = y|X_i = x)$ and would interpret as having a higher confidence in the classification.

Recall here that in a classification problem we are interested in maximizing a sample's probability of belonging to a certain class, given the input data:

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)} \tag{3.4.8}$$

Then for the binary classification, the two-class problem, we have that

$$\begin{aligned} P(C_1|x) &= \frac{P(x|C_1)p(C_1)}{P(x|C_1)p(C_1) + P(x|C_2)p(C_2)} \\ &= \frac{1}{1 + \exp\left(-\log\left(\frac{P(x|C_1)p(C_1)}{P(x|C_2)p(C_2)}\right)\right)} \end{aligned} \tag{3.4.9}$$

Here we see the close resemblance to the logistic function which is acting on the *log-odds ratio* $\log\left(\frac{P(x|C_1)p(C_1)}{P(x|C_2)p(C_2)}\right)$. This equivalent form of the posterior distribution of one class with the log-odds one property defining the model.

The second defining property of this model is that the log-odds are described as a transformation on a linear combination of inputs, restricted to the sum of the posterior probabilities being one.

Let j be a fixed class and denote

$$\log\left(\frac{P(C_i|x)}{P(C_j|x)}\right) = \theta_{i0} + \theta_i^\intercal \cdot x \tag{3.4.10}$$

for $i, j \in \{1, 2, \ldots, K\}, i \neq j$.

With these same indices we have that

$$P(C_i|x) = \frac{\exp(\theta_{i0} + \theta_i^\intercal x)}{1 + \exp(\theta_{j0} + \theta_j^\intercal x)} \tag{3.4.11}$$

In this way the model is specified in terms of the log-odds for each class with respect to the base class j with the model completely defined by the parameter θ.

Here, we can structure the target as a Bernoulli random variable when conditioned on the input variables. Formally we have that,

$$\begin{aligned} Y_i \mid X_i &\sim \text{Bernoulli}(p_i) \\ \mathbb{E}[Y_i \mid X_i] &= p_i \end{aligned} \tag{3.4.12}$$

The probability function of the target given the features $\Pr(Y_i = y \mid X_i)$ is given by

$$\Pr(Y_i = y \mid X_i = x_i) = p_i \; I_{(y=0)} + (1 - p_i) \; I_{(y=1)} = p_i^y (1 - p_i)^{(1-y)} \tag{3.4.13}$$

where this depends on the class of y.

Here the logit function is utilized to map log odds into conditional probabilities and the model will output for each sample the predicted probability of the target variable belonging to a certain class.

This is why we are specifying a model where the target is a linear function of the inputs, corrected by an error term. Our model's predictions will then be characterized

by:

$$Y_i = I(\theta_0 + \theta \cdot X_i + \epsilon > \frac{1}{2}) \ \forall i \tag{3.4.14}$$

where ϵ is the error of the approximation and is distributed with the standard logistic distribution.

If we take the conditional probability in Equation (3.4.14), given the features we will have that

$$logit(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = logit(\mathbb{E}[Y_i|X_i]) = \theta_0 + \theta \cdot X_i \tag{3.4.15}$$

By an abuse of notation, we absorb θ_0 into X_i. This can be safely done if we consider a dummy attribute X_i^0 which is always equal to 1 for every sample i. Then from the equation above we have that

$$p_i = \sigma(\theta \cdot X_i) = \frac{\exp(\theta \cdot X_i)}{1 + \exp(\theta \cdot X_i)} \tag{3.4.16}$$

Finally, using this p_i representation and plugging it into the initial conditional probability of Y_i in equation Equation (3.4.13) we will have

$$\Pr(Y_i = y \mid X_i = x_i) = p_i^y (1-p_i)^{(1-y)} = \frac{\exp(\theta \cdot X_i)}{1 + \exp(\theta \cdot X_i)} \tag{3.4.17}$$

Maximum likelihood is the most common method employed to fit the model. with Multinomial distributions modeling the features. Given a parameter θ, the probability of having a target vector y is

$$P(Y = y \mid \theta) = \prod_{i=1}^{N} P(y_1 \in C_1 \mid x_i, \theta)^{y_i} (1 - P(y_1 \in C_1 \mid x_i, \theta))^{(1-y_i)} \tag{3.4.18}$$

If we take into account that $P(y = 1 \mid x, \theta) = 1 - P(y = 0 \mid x, \theta)$ and the logistic function's derivative form Equation (3.4.7), then the estimation of θ for N samples is equivalent to minimizing the following form

$$l(\theta) = \sum_{i=1}^{N} (y_i \log(P(y_i \mid x_i, \theta)) + (1 - y_i) \log(1 - P(y_i \mid x_i, \theta))) \tag{3.4.19}$$

In Machine Learning this is called a *log loss function* and it is central to the theory because it is used as a metric or score of the quality of the model, in this case θ, in correctly assigning estimated targets $\hat{p_i} = P(y_i \mid x_i, \theta)$ for the input data X. It defines the space over which the learner will be evaluated.

Definition 3.4.1 Loss Function. The loss function of a model $l : \mathbb{R}^p \to \mathbb{R}$ quantifies the model's parameters (θ) error.

In all cases, the loss function will define the underlying optimization procedure behind the model that is fit. By comparing for each sample the target predictions versus their actual values, loss functions give a measure of how *good* classifiers are with respect to their predictions. These can be described as semi metrics on the target space, i.e., functions that are symmetric, non-negative and equal to zero only if both arguments are equal.

Loss functions are used to find the optimum θ^* as the $\text{argmin}_\theta\, l(\theta)$. Thus we say that loss or scoring functions give us a way to find the best parameters for our model.

Note however, that the optimum parameter will be dependent on the choice of the algorithm used to construct the model. That is, for our particular case we have selected the optimum θ for a Logistic Classifier with input data X. Other algorithms can be used for the same task, and these will have their own parameters and loss functions to optimize.

In our derivation, the scoring function was constructed from the assumptions given at the start. However, we could in fact build *any* type of algorithm which given an input x_i and a parametrization of the model θ, outputs a value $P(y_i \mid x_i, \theta)$. In this way we can compare the performance of that model to the one built by a Logistic Regression Classifier by using the same log loss function for both. Refer to Section A.2 to find more details on this loss function's properties and on what are the common optimization procedures to fit this function.

3.5 Model Regularization and Hyper-parameters

Definition 3.5.1 Model Regularization. Regularization of a model is the process in which restrictions and conditions are imposed on the model's loss function l through a functional $R(l)$. The regularization term is jointly optimized during the fitting procedure.

Regularization is used in problems which are ill-posed. Most often, this is well suited for situations where the model has very different performances in the training vs. the test set. In its most used form, regularization imposes restrictions to the model so as to reduce the total number of features used.

This *shrinkage* of parameters can be forced by penalizing large values for θ either by the number of non-null components of this value or by penalizing its distance to zero. By doing so, we hope that only the most relevant features are selected in the final model.

Also, a shorter model has the benefit of being more accurate in the estimation of the model's performance on unseen samples. In other cases, the benefit of a reduction in the number of features is that the overall variance of the predictions are reduced, only with a slight increase in error.

In addition, a more *heuristic* argument for model parsimony follows from Occam's razor principle. Here it applies in that any model can approximate the true underlying process only up to a certain level and if two models have the same predictive power the simpler model should be preferred. Better examples on this point are illustrated

in [Rasmussen and Ghahramani, 2001]. The authors give examples of how this principle applies in different Machine Learning settings.

In practice, most algorithmic implementations of regularization resort to use $l1$ and $l2$ norm penalizations on the θ vector. Both have its advantages and pitfalls, yet these details exceed the nature of this work. As a summary, benefits are related to the robustness of the solution, convexity of the minimization function, unique global minimum and model parsimony.

Of the two norms, the former one is more commonly known as *Lasso* regularization and the latter is known as *ridge* or *Tikhonov* regularization. Other variants include the *elastic-net* penalty which is a weighted average of the previous two norms, or the l_0 norm, which is the number of non null components of θ.

The following example shows the logistic regression's model loss function with an $l2$ regularization term on θ:

$$\begin{aligned}\theta^* &= \min_\theta \sum_{i=1}^{N} (y_i(\theta \cdot x_i) + \log(1 + e^{1-\theta \cdot x_i})) + \lambda \|\theta\|_2{}^2 \\ &= \min_\theta \sum_{i=1}^{N} (y_i(\theta \cdot x_i) + \log(1 + e^{1-\theta \cdot x_i})) + \lambda \sum_{j=1}^{P} \theta_j^2\end{aligned} \tag{3.5.1}$$

The value λ acts here as the weight that our optimization procedure will put to the regularization part of the minimization function and P is the number of features used in the model. Note that in an abuse of notation, the functional takes the form $R(l) = \lambda \|\theta\|_2{}^2$ where the model is identified by its θ parameter. In this work, we will only consider model regularization with lasso, ridge, or elastic-net regularizations. As an example we show here how regularization affects an instance of a Logistic Classifier on Problem 1.

In Figure 3.5.1 we show an example of different models loss function scores across varying levels of regularization. Here we try to predict which users were endemic in the past by making predictions over Problem 1. To fit the models we used all of the features from $\mathcal{T}$ and we generated multiple models $f(x, C)$ where we defined

$$C = \frac{1}{\lambda}$$

In detail, we have that each logistic model was built from a specific C in $[10^{-5}, 10^5]$ and that the model's negative log loss error ($NLL(C)$) was measured for both $\mathcal{T}$ and $\mathcal{T}_f$ sets separately. Then, the function of scores $1 - NLL(C)$ is graphed as curves over the C value.

As we can see, there is a clear tendency to have better performance across smaller values of C which is equivalent to having stronger regularization of the scoring function. This illustration serves to exemplify why regularization can help models' predictions as we see that in this case the $1 - NLL(C)$ score will output higher values for better classifiers. Thus we find that, as a simple case and with no other manipulation of the data, a regularized model helps in having a better performance

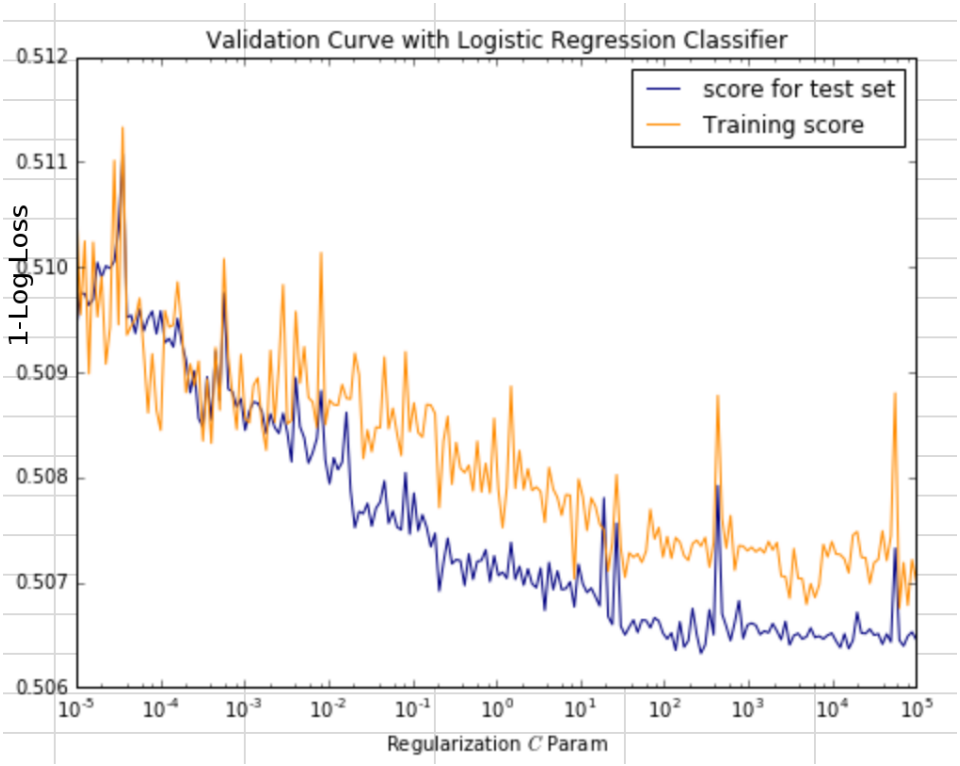


Figure 3.5.1 Problem 1's validation curve of a Logistic Classifier model on training and test sets. Here, for each value of the C regularization parameter, we show the score $1 - NLL$. Note that with this formulation, higher scores mean better results.

across the test and training sets error.

3.5.1 Hyperparameters

We can see that in the model built from equation Equation (3.5.1) there are two specific parameters which we need to predefine before starting the optimization procedure. Notably we chose the value of λ and the regularization method (by selecting the $l2$ penalizing norm, or ridge regularization, over other possible regularizing options). It is said these values are *hyper-parameters* of the models since they are not directly part of the theoretical construction yet they need to be previously set in order to find a solution θ^*.

In the literature some authors can also refer to the *hyper-parameters* of the model as *tuning parameters*. These are the values set to configure the different possible variants in the loss function and in the model's training phase. They will directly affect the estimated fit $f_{\hat{\theta}}$, yet are not the θ parameter themselves. They need to be instantiated before training the learner and, as such, they can't be learned from the dataset.

3.6 Chapter Summary

We introduced some basic vocabulary on supervised Machine Learning settings, with references to other types of other non-supervised settings which are not relevant to our general task. Also, the human mobility problems of this thesis were presented using the aforementioned notation.

A concrete Machine Learning example was introduced in which we constructed a logistic classifier, where we derivate its loss function from basic theoretical assumptions. This led us to optimize the hyperplane that best separates the two classes by estimating parameter $\hat{\theta}$. Given that we now have a problem of p degrees of freedom, we are left to find or fit the parameter θ by optimizing on certain criteria. The loss function will then shape our evaluation criteria to determine whether one parameter is preferable to another. Finally, we expanded on the concepts of hyper-parameters and regularization methods which will be reused later in the upcoming sections.

Chapter 4

Generalization Error and Model Selection

In this chapter we will extend some of the Machine Learning concepts introduced in the previous chapter and dive into the different tools that will help us evaluate which are the best models for our task. These tools will be based on the idea of the generalization error minimization with its decomposition into its bias and variance components. For our applications, we will settle around model hyperparameter's Cross Validation.

4.1 Prediction and Generalization Error

Historically, a convenient loss function used to optimize the model's parameters was to minimize the residual sum of squares. This measures the performance of our model $f(\cdot)$'s predictions on input samples x with respect to the data's known output classes y as

$$RSS(\theta_0, \cdots, \theta_p) = \sum_{i=1}^{n} [y_i - \hat{y}_i]^2 = \sum_{i=1}^{n} [y_i - h(\theta \cdot x_i)]^2 \tag{4.1.1}$$

We know that the objective is to have a good model in the generalization sense: one which can make a *good* prediction on any sample, even new ones from the true data distribution. This generalization concept is lacking in the preceding Equation (4.1.1) because it is built to fit the actual dataset in the best possible way. Given that the training set ($\mathcal{T}$) is a sample of the population, it is not straightforward that the model will perform well accurately any other given sample of the *true* distribution of the data.

We need a way of measuring and comparing the generalization power between models to select the best supervised estimator. For our test data, it means to have

a model that can correctly label new samples which were not used in the training phase, such as those from the test set T_s.

For now, let us assume that $f : X \to Y$ is a function which represents a true existent relationship in the data such as

$$Y = f(X) + \epsilon$$

.

In this mapping from feature to target space we assume ϵ to be the noise, with a null mean and fixed σ^2 variance i.e. $\epsilon \sim \mathcal{N}(0, \sigma^2)$. With this, Equation (4.1.1) can be read as an approximation of the expected prediction error given the training set $\mathcal{T}$.

Definition 4.1.1 Prediction Error: Given a loss function $L(\cdot, \cdot)$ and a function f, we say that the **prediction error** for the resulting classifier f for (Y,X) is

$$PE(f_\theta) = [L(Y, f_\theta(X))]$$

This is the error between our model's output and the target labels, as quantified by the loss function. In our specific example, with the parameters θ encoding the structure of f as $f_\theta(x) = h(x \cdot \theta)$ we would have that:

Definition 4.1.2 Squared Loss Prediction Error: Given a value for parameter θ and the *squared error loss*, we say that the **prediction error** for the resulting classifier f_θ under the squared loss is

$$PE(\theta) = L(Y, h(X \cdot \theta)) = (Y - h(X \cdot \theta))^2 \tag{4.1.2}$$

Our interest is now in minimizing this error, or loss, for all possible values from the true distribution, and not only for those in the dataset. We would like to quantify the expected error over occurrences of **X** and **Y**. This means we need to account for the variables randomness and that we need to think of a way to consider measuring this error independently of the current sample $\mathcal{T}$, which is the data used to fit the model. The Generalization or *Test* error capture this idea

Definition 4.1.3 Generalization Error

$$Err_{\mathcal{T}_f} = \mathbb{E}_{\mathbf{X},\mathbf{Y}} \left[L(Y, f(X)) | \mathcal{T}_f \right] \tag{4.1.3}$$

The problem appears when considering that in this scenario, the models' fit is done on a fixed dataset and, in turn, the dataset is a fixed sample representation of the true distribution. Thus the generalization error is using only the model's error conditional to the test data that is available. This means that in practice we have to ensure that training ($(\mathcal{T})$) and test sets ($\mathcal{T}_f$) independence.

The generalization error intends to compare the performance of our algorithm trained on a training dataset, with the loss that this model would have had on the

true data distribution. If we consider the case of the squared loss function, we will have that this error now becomes

$$Err_{\mathcal{T}_f} = \mathbb{E}_{\mathbf{X},\mathbf{Y}}\left[L(Y,\hat{f}(X))|\mathcal{T}\right] = \int_{\mathcal{T}_f} (y - h(x\cdot\theta))^2 P(x,y)dxdy \tag{4.1.4}$$

Overall, we are ultimately wanting to know how the learner performs over the true distribution, or what is the average prediction error's for any sample of data. Note here that we also want to average out any specific influence of the training set on our fit model $\hat{f}$ which was trained from a specific dataset, but also on the $\hat{f}^*$ that would have resulted from training on the true distribution of the data. With this we can introduce a last definition:

Definition 4.1.4 Expected Prediction Error

$$EPE = \mathbb{E}_{\mathbf{X},\mathbf{Y}}\left[L(Y,\hat{f}(X))\right] = \mathbb{E}_{\mathbf{X}}\left[\mathbb{E}_{\mathbf{Y}|\mathbf{X}}\left[L(Y,f(X))\right]\right] \tag{4.1.5}$$

In practice we have finite access to samples, so we have to estimate the expected prediction error by means of the generalization error. From the data, we will build or *train* our model and from this reduced sample we will extract a training error to determine how well the model is performing.

Definition 4.1.5 Training Error: is the average loss over the sample prediction errors:

$$\overline{err}_{\mathcal{T}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i,\hat{f}(x_i))$$

and with this same way we are going to calculate the generalization error

$$Err_{\mathcal{T}_f} = \frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\mathcal{T}_f} L(y_i,\hat{f}(x_i)) \tag{4.1.6}$$

Both of these formulas will be thoroughly used throughout this work, and it is with the generalization error that we will estimate the EPE. Doing this will let us find the best model for our task, where we will turn to rank different learners according to their errors $\mathcal{T}$ and $\mathcal{T}_f$. Combinations of high or low values for these, across training and test data, will be used as model evaluation and by this we will start to find aspects of the algorithm which need improvement.

4.2 Bias and Variance

If we look closer at the EPE for the specific case of the squared loss function with

$$EPE = \mathbb{E}_X\,\mathbb{E}_{\mathbf{Y}|\mathbf{X}}\left[\left\|Y - \hat{f}(X)\right\|_2^2\right] \tag{4.2.1}$$

It is not difficult to see that the model that minimizes this error is

$$\hat{f}(x) = \mathbb{E}\left[Y|X = x\right]$$

We say that our model $\hat{f}$ is an estimate of the true relation in the data, and takes the form above, with the form structure from the data.

With the squared loss, the expected prediction error

$$\mathbb{E}\left[\left\|Y - \hat{f}(X)\right\|_2^2\right]$$

of the model can be decomposed in the following way:

$$\begin{aligned} EPE(\hat{f}) =& \mathbb{E}_{\mathbf{X}}\left[f(X) - \hat{f}(X)\right]^2 + \mathbb{E}_{\mathbf{X}}\left[\hat{f}(X)^2\right] \\ & - \mathbb{E}_{\mathbf{X}}\left[\hat{f}(X)\right]^2 + \sigma^2 \\ =& Bias(\hat{f})^2 + Var(\hat{f}) + \sigma^2 \end{aligned} \tag{4.2.2}$$

Equation (4.2.2) is hereby referred to as the *bias-variance decomposition for the squared loss* where the first term is called the square of the estimator's bias. This measures how good are our estimator's predictions compared to the true relational function. The second and third terms are the estimator's variance. This will measure how this random variable varies along its most expected value. The noise's variance term is that part of the prediction error which is irreducible. Note that we have already taken the expectation over the target and that is why we are left with the target's inherent noise. This part of the error we cannot reduce or control with our learner model since it is caused by the problem's random nature.

In the EPE's decomposition we are integrating over the joint distribution of inputs and outputs. As we've mentioned before, we have incomplete information on $P(x, y)$ given the limited amount of information in $\mathcal{T}$. We must assume then that calculating this integral is not possible for any θ so we must rely on estimation procedures.

Historically, the concepts of bias and variance where associated directly with the squared loss function and in the literature it is said that algorithms have ways of attacking these two sources of error. They are key elements in the expected prediction error and they point to different weak spots in the algorithms. At the same time, different methods are used to improve each of them, where ultimately our having control over both errors is central to our prediction task. Authors point that in practice it is usual that the improvement of one generally leads to a decrease in the other.

Conceptually the bias error represents the model's accuracy in labelling predictions correctly. It is the model's best attempt to capture the functional relationship among the feature and target variables. This could either mean it is correctly assigning the sample to its correct class in classification settings, or, in a regression setting, by

estimating a value for that sample which is *near* to the target value.

The bias is lower when models correctly learn the underlying structures of the data. However, as bias decreases, the model complexity increases and more data is needed to train it correctly because the model becomes very fit to the training set i.e. it loses the ability to extend this predictive accuracy to new samples because the model learns too much from the available samples only. In this situation, we have another type of error which is when we have an increase in variance.

In classification problems it is uncommon to use the residual sum of squares to fit the model's parameters and there are other *loss* functions to rely on. Specifically in the binary classification context, a common loss function used is

$$L(Y, \hat{f}(X)) = I(Y \neq \hat{f}(X)) \tag{4.2.3}$$

where Y will be taking any of value of the class set $\mathcal{G}$.

With this loss function, the classification prediction error can be decomposed in

$$P\left(\hat{f}(X) \neq Y\right) = Var(Y) + P\left(Y = \underset{1 \leq i \leq K}{\operatorname{argmax}} P(\hat{Y} = i)\right) - \sum_{i=1}^{K} P(\hat{f}(X) = i) P(Y = i)$$

where in this formula we have, once again, that the error is decomposed by using the classifier's variance. Here, the classifier's bias is not clearly defined as with the squared error. Yet the decomposition quantifies how similar is the learner's class probabilities with the true distribution in the data, and how this interaction differs from the most probable output value, which is the Bayes classifier.

Note that in this definition we do not have the bias of the model as a term of the error. This quantity is rather taking effect as part of the second and third summands in the equation and it tries to capture how close is the classifier's distribution with respect to this distribution in the data. Another relevant difference is that in this definition there are no noise terms For more detailed information on bias and variance decomposition in classifiers, one can refer to Section B.1 which is based on the survey work of [James, 2003].

4.2.1 Overfitting

For any given Machine Learning learner, we can have different combinations of high or low variance and biases. Naturally that makes three cases out of four model performance scenarios that need to be improved, where we have low bias and/or low variance. For each combination of bias and variance stages, there are different strategies to improve a model. For example, in one of the most common scenarios we might have a model that has a high overall variance and low bias. If we also have that the model has a good performance on $\mathcal{T}$, whilst having a poor performance on $\mathcal{T}\int$, it is said in the literature that the model is *overfitting* the data. The reasons

behind this scenario can be various and most of them are related to learners which are overly-complex or situations in which there is insufficient training data. This results in learners which are only suited for the training set.

Overfit models will also fail to generalize on new data since the variance structure of the training set is such that the number of training samples is not enough to lower the overall variance of this model. To illustrate this point, in Figure 4.2.1 we show the results from fitting a learner on our own CDR dataset. We fit a Decision Tree classifier on Problem 2 which has a high imbalance between the positive and negative target classes to illustrate the interaction between bias and variance. A discussion of this model's formulation is presented in Chapter 5. This is a tree based model which increases its complexity by growing the tree's depth. In turn, this increases the number of features used in by the leaner to decide on the predicted target.

To measure the performance of the fit learner, we take a scoring function for the model's performance which outputs a value in the range of $(0, 1)$. With this function we have that higher values means better scores. We will introduce later specific formulations for scores generally used. In this way, Figure 4.2.1 graphs how the model performs at different levels of model complexity. In the image, the test and training errors are given as a function of the tree's depth.

At first the model's bias is high both for the training and testing sets, and as a consequence, we would expect to have a high generalization error. We later see how the overall error decreases as complexity increases.

An estimate of the expected prediction error is shown, calculated by averaging over the test set' prediction error's. We note how at first it decreases when the model's complexity is increased. However, later when the model starts to overfit the data, we can see how the test error starts to rise whilst the train error keeps decreasing. This situation is important to our EPE estimation cause it hints that the model has lost predictive power due to an increase in variance.

In [Hastie et al., 2009] the authors give a rough heuristic to select the best model: stop increasing the model's complexity once the estimated EPE stops decreasing. That is, chose the point where the scoring error of the test set stops decreasing along with the training error. For Figure 4.2.1, this would roughly be when $MaxDepth = 10$.

In this example we are only looking for the best model over a search space of one dimension, namely the tree-depth hyperparameter only. In practice, the models will have more parameters to tune and this makes finding the lowest prediction error for all possible models difficult. Still, we might settle for near-optimal models which reach acceptable error scores. The following section introduces the assumptions to consider the test error as a good approximation of the generalization error. It also gives a formal argument in favor of finding models whose scores are not optimal for that dataset.

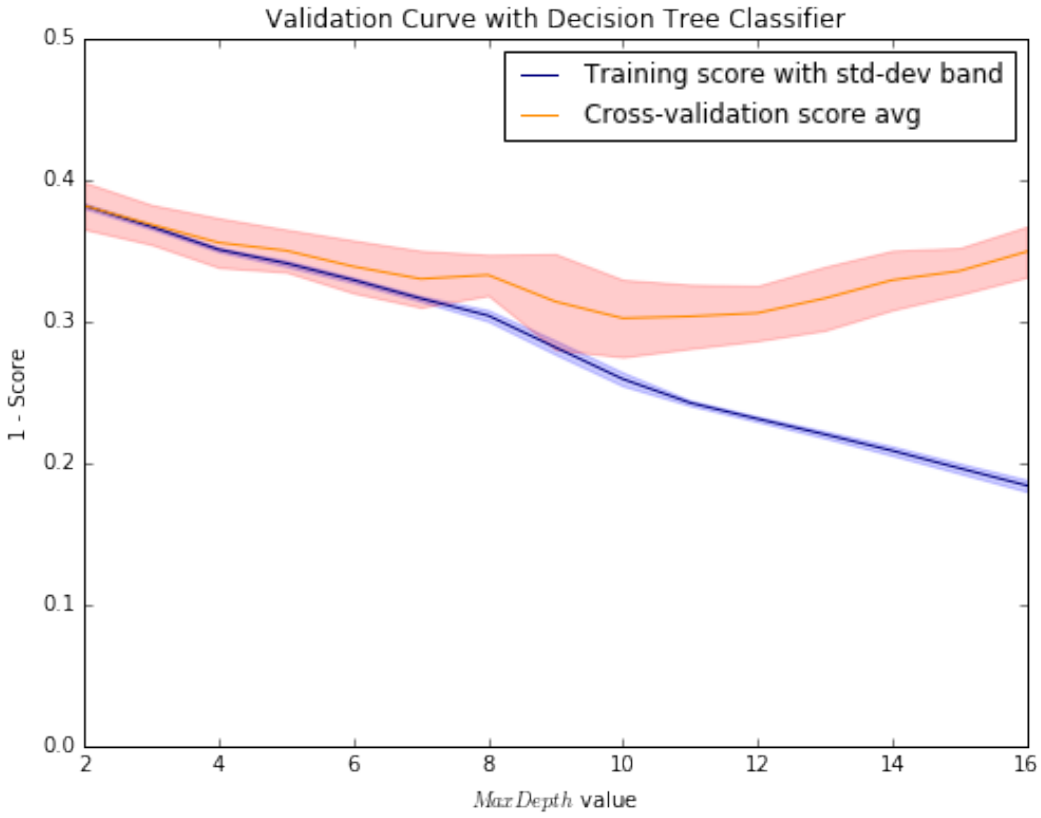


Figure 4.2.1 Training and cross-validated mean *Accuracy* score functions along the change in max tree depth. For Problem 2, one model is fit for each tree depth value on both the training and cross validation sets. Both score series are compared side by side, with their corresponding standard deviation band for the CV folds.

4.3 Cross Validation (CV)

Classification models are algorithms which intend to build approximating functions to a stochastic process, by means of algorithms. Due to the searching nature of this process, in the generally large space of hyper-parameters, we have that when models are fitted, we ultimately create different learners.

These differences between learners are controlled by the *parameters* or configuration of the algorithm. And the nature of each parameter can be due to reasons such as computational or statistical algorithmic variants. An example of this, which was introduced in Section 3.5, is the λ penalization parameter in a Logistic Regression. In this case we have that λ controls the amount of weight to be placed on the regularization term of the minimized function.

Under this setting Cross Validation (CV) is a technique introduced to systematically explore tuning parameters values and decide which learner is better for the task

at hand. It intends to help in provide a better way of estimating the EPE during the process of finding the best hyperparameters.

By comparing the generalization error of each model, where models vary accordingly with the hyper-parameters' values, the model's evaluation score is used to select the *best* model in the class.

The technique is the most widespread for evaluating the generalization performance of a set of learners. Given a number of possible configurations or values for the tuning (hyper) parameters of an algorithm, we would want to decide which selection of these fit the best estimator $\hat{f}$, as measured by the generalization error.

In most problems we will have that data is scarce and, at the same time, we have that prediction error estimates are hardly computable in practice. In part, this because estimates are based on assumptions that are not practical: they rely on the true distribution of the data, access to an *infinite* amount of data, or because they are based on analytical results which are difficult to compute. To cope with this, CV intends to prevent over-fitting problems and improve the EPE estimation procedure by applying a procedure which iteratively holds out a random split of the dataset. With this, the procedure will measure the predictive accuracy of the learner fit of data *in-sample* against values from the *hold-out* part of the dataset. In a sense, hold-out samples are acting as "test" to the learner fit with in-sample data. As usual, the evaluation measure here is given by the loss function, which we must select beforehand.

4.3.1 Formulation of the Cross Validation (CV) Procedure

In this method, we will denote a partition of the samples as a *fold*. CV will then partition the data into K random separate *folds*, where the number K is preset[1].

Let $\gamma : \{1, \cdots, N\} \mapsto \{1, \cdots, K\}$ be a function mapping samples to folds. Without loss of generality, let α be an index of the model's hyper-parameters configurations, where each distinct value combinations for all tuning parameters is identified by this index[2].

The idea behind the CV algorithm is to run an iteration over all the folds, denoting $k \in [1, \ldots, K]$ the iteration indexer. It will then take one fold $\gamma^{-1}(\{k\})$ to be the validation set and if we let $\hat{f}^{-k}$ be the fitted estimator on the training set and k-fold hold out, the classification performance will be tested against these *out of sample* estimates. Finally we will have that for every sample in this k-fold, we will measure the loss $L(y_i, \hat{f}^{-\gamma(i)}(x_i))$ of the in-sample model's prediction, against the true target's value. The average score over all samples would get us the Cross Validation error for this problem instance.

We see that Cross Validation intends to estimate the expected *out-of-sample* error $\mathbb{E}\left[L(Y, \hat{f}(X))\right]$, when the model is tested against **independent** samples from

1. We assume here that we are selecting samples from the training set $\mathcal{T}$ and not from the test set $\mathcal{T}_f$.

2. Note that the domain of α will vary with each model, which defines the type of algorithm used to learn.

the true distribution. To do this it is fundamental to ensure independence of the training set and the test set. Any data transformations that must be done on the input data X that jointly uses the output samples Y in the process, must be done or "learned" only on the training set. It is important that no information from our test set is introduced into the final estimator output from the CV procedure.

Models have to be agnostic to any information contained in $\mathcal{T}_f$. In a sense this means that the learner never "sees" any test data until we use it to evaluate our learner's performance.

4.3.2 Selection of tuning-parameters (hyper-parameters)

The CV procedure provides a reliable EPE estimate to select the best values for the algorithm's tuning parameters. These hyperparameter configuration will be *best* in the measure provided by the loss function. Let $\mathcal{A} = [\alpha_0, \alpha_1, \ldots, \alpha_l]$ be a list of hyperparameter settings and $\mathcal{K} = [1, \cdots, K]$ a list of folds with $\gamma(\cdot)$ a function which maps samples to their corresponding fold. To clearly depict how a full K-Fold Cross Validation procedure runs, we outline its pseudo-code.

Data: All hyper-parameter configurations $\mathcal{A} = [\alpha_0, \alpha_1, \ldots, \alpha_l]$ and a number K of folds.
Result: Data table mapping error scores to hyper-parameter configurations $CV(\alpha)\ \forall \alpha \in \mathcal{A}$
Initialize $\mathcal{A}$ and $\gamma(\cdot)$;
for $\alpha \in \mathcal{A}$ **do**
  if *data transformation* **then**
    Perform data transformation on the whole training set $\mathcal{T}$;
  end
  for $k \in \mathcal{K}$ **do**
    if *feature selection* **then**
      Perform feature selection on the $(T)^{-k}$;
    end
    fit $\hat{f}^{-k}(\cdot, \alpha)$;
  end
  compute $CV(\alpha) = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \hat{f}^{-\gamma(i)}(x_i, \alpha)\right)$;
end

Algorithm 1: Pseudo-code for K-Fold Cross Validation Estimation for an index of α hyper parameters.

Note that during the loop for α, each sample's prediction is tested on the model which was fitted without using that same sample. This is central to the idea of cross-validation in which training samples are used *as if* they were testing samples.

With this method, it makes sense to choose a final model $\hat{f}_\alpha$ with the lowest $CV(\alpha)$ score among all possible hyper-parameters. However, following ideas detailed in Section B.2, importance should also be given to the *complexity* of the approximating function. A common rule of thumb is to favor models in which we use a lower number

of hyper parameters or features. In practice however, a class of approximating functions might have a defined complexity which is not analytically computable. Thus in some cases, crude heuristic estimates or common sense is used to estimate model complexity, without making use of theoretical arguments.

4.3.3 CV run on CDR data

As an example, we consider here a cross validation procedure over a Logistic Regression model. Here we are using Problem 2 which is defined to find those users that used to live in the endemic area and eventually migrated out of it, becoming non-endemic during the training phase. This procedure serves as a benchmark to compare different hyperparameter configurations.

In this case we decide to optimize the model's regularization strength which is captured in the parameter $C = \frac{1}{\lambda}$. Recall from Equation (3.5.1) that this is the regularization strength of the model, and note that a smaller value in C is a stronger regularization. We perform this experiment and consider the difference in scores between $\mathcal{T}$ and $\mathcal{T}_f$ for each possible C value. It is important to note that all test scores were performed on models built only from $\mathcal{T}$.

In Figure 4.3.1 the x-axis represents the parameter C's shown on a $\log_{10}$ scale. The score used is the negative log loss function on a twelve fold cross validation set. Both the training and the mean cross validation scores are shown.

It is interesting to see that for low C values, the test and CV scores are very similar in that they have are better scored. This is probably due to the fact that being on a log scale, the algorithm's regularization dominates the optimization routine.

However, at some point near $\exp(-2)$, both lines separate differ in a notorious way. This continues to be notorious in the direction of increasing C values. Here it can be seen that the test score is barely in the one standard deviation band of the average CV score. This image serves as an example of how different the CV and test scores can be.

With this we have come to show an example of how Cross Validation approximates the test error of a model. The method required only minimal information from the data to do this, yet we have to preset a value for the number of K folds to use.

To determine this value a priori might not be possible and experimental results in the literature show the different results for CV routines with varying K number of folds. For more information on this please refer to Section B.3.

4.3.4 CV Scores in Classification Learning

In binary classification, the contingency table is a good tool to summarize a learner's training performance over samples in $\mathcal{T}$.

Let $\hat{f}$ be our model learned from the data after a CV procedure. To construct this table we note that for each sample, there can only be four possibilities when comparing the model's prediction and the sample's observed target value. The

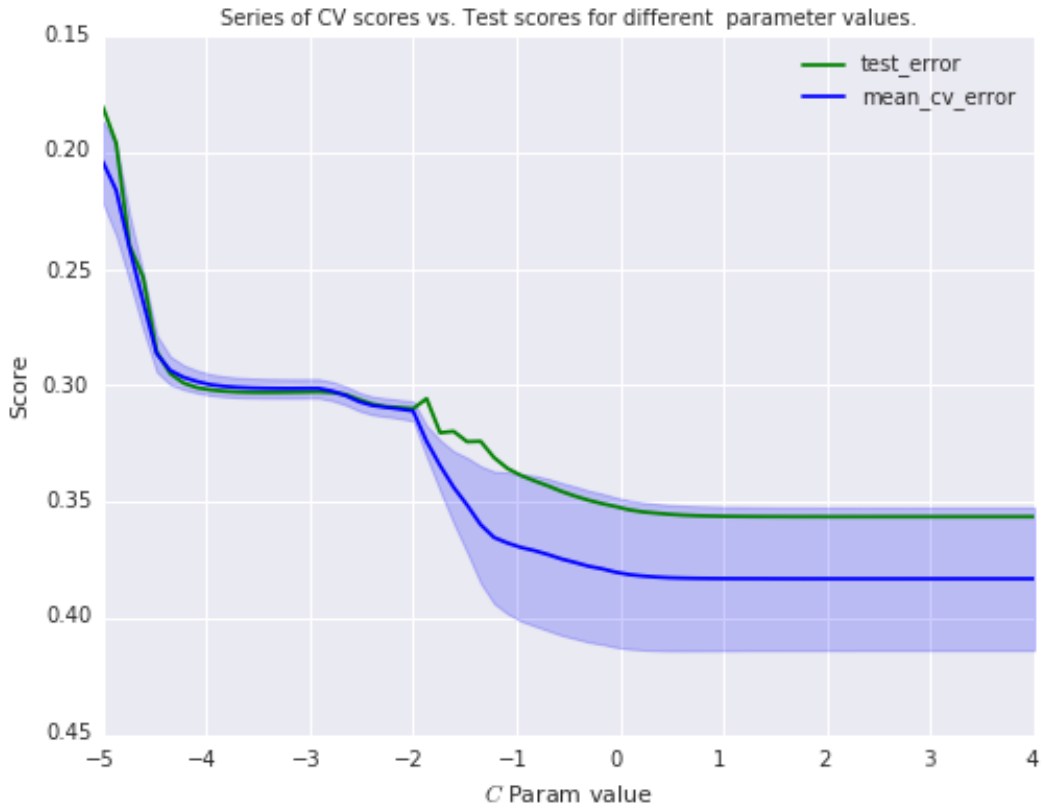


Figure 4.3.1 Comparison of 12-fold cross validated and test average errors for Problem 2. Negative log loss (NLL)a scores are shown for each value of the regularization parameter C. Standard deviation bands on the K fold errors are added to the CV series.

contingency table then shows the count of samples that fall into each of the these four groups.

We can express the models' target value $\hat{y}$ into the positive ($\hat{P}$) or negative ($\hat{N}$) categories. At the same time we can set actual target data into the positive (P) or negative (N) categories.

To assess the performance of the classification algorithm and pick the *best* model we must decide on how the CV procedure will value two different models. For this we must evaluate the mismatch between the target and the predicted value in a quantifiable way.

In this context these evaluating functions are also known as *scores* or *measures* and they are built by looking at how many times an algorithm misclassified instances, and in which situations do the misclassification happen. To visualize this, the *confusion* table presents these results in the following table:

		Predictive value $\hat{y}$	
		$\hat{\mathbf{P}}$ (0)	$\hat{\mathbf{N}}$ (1)
Target value y	**P** (0)	True Positive (TP)	False Negative (FN)
	N (1)	False Positive (FP)	True Negative (TN)

These cell values count the amount of instances that fall into each of the four possible outcomes. Building from this, we have metric scores constructed to provide values on the algorithm's performance. These counts are combined in different transformations that measure different aspects of an algorithm's classification performance.

Some of the most common metrics include the following:

- **True Positive Rate (Recall):** $\frac{TP}{P} = \frac{TP}{TP+FN}$
  This rate measures the percentage of real positive values captured by the algorithm. A high recall of the algorithm indicates that a high number of the real positive labels were classified as positive.
- **Positive Predictive Value (Precision):** $\frac{TP}{\hat{P}} = \frac{TP}{TP+FP}$
  This rate measures the *confidence* of the algorithm in its predictions of the positive class, where a high precision indicates value in its predictions.
- **True Negative Rate (Specificity):** $SPC = \frac{TN}{N} = \frac{TN}{TN+FP}$
  This rate measures the percentage of real negative values captured by the algorithm.
- **False Positive Rate (Fall-Out):** $FPR = 1 - SPC$
  This rate measures the percentage of false negative values misclassified by the algorithm.
- **Accuracy:** $\frac{TP+TN}{P+N} = \frac{TP+TN}{TP+FP+TN+FN}$
  This rate measures the *confidence* of the algorithm in all of its predictions.
- **F1 Score:** $\frac{TP+TN}{P+N} = \frac{TP}{TP+FP} = 2\frac{1}{\frac{1}{recall}+\frac{1}{precision}}$
  This is the harmonic mean of the recall and the precision metrics. It's advantage is that it can capture both of the scores with equal weight. Its values range in the $[0, 1]$ domain and are ordered in the sense that perfect classifiers have an $F1$ score of 1.

To illustrate the difference in these metrics we ran an experiment on Problem 2 where we intend to correctly classify users that moved out of the endemic region from

the past to the current time period. We took one logistic classifier with common settings and we ran a cross validation procedure over values of the regularization strength C.

In this run, we considered four metrics to cross-validate our models: *Accuracy*, *Precision*, *Recall* and $F1$. We compared all of them in a procedure which had fixed hyper parameters for all models. The number of folds, K, was set to eight and the lasso regularization ($l1$) configuration was optimized. At the same time, we fixed to 100 the maximum gradient descent iterations for every configuration tested. We also balanced the contribution of the positive and negative samples to the loss function, where samples of the , under-represented, positive class had a higher contribution to the loss. This was done by reweighing each individual sample's loss, where the weight was equivalent to the reciprocal of that sample's class percentage with respect to the total number of samples.

For each metric, the best model output from the CV procedure which was selected based on the score. All winning models were then evaluated against $\mathcal{T}_f$ to give a final evaluation metric on the model's performance. No other information from $\mathcal{T}_f$ was used during the whole process. In Table 4.3.1 we show a summary of these benchmarks.

Table 4.3.1
Results comparing an 8-fold CV on Problem 2, using a Logistic Classifier over varying regularization C values. Four metrics were cross-validated and compared in this experiment: *Accuracy*, *Precision*, *Recall* and $F1$.

Metric	Best C value	Mean CV score	Test score	Full CV time (s)
Accuracy	0.316	0.685	0.695	4628.3
Recall	0.107	0.637	0.612	4638.2
Precision	0.0006	0.038	0.035	4667.5
$F1$	0.0015	0.0071	0.005	4612.1

It is clear from these results that all metrics favor strongly regularized models, except for the *Accuracy* metric which favored the least regularized models of all CV procedures. As expected, calculating each score is straightforward from the contingency matrix and there are no significant runtime differences among them. CV procedures all take nearly the same time to compute.

Among all best-fit models, the biggest difference in test and average CV scores was of 4%, yet the algorithm had very poor performance when finding samples of the positive class. This is evidenced in the extremely low value for the *Precision* scores where only a handful of positive classes can be correctly captured by the model. The effect of the low precision is also evident in the $F1$ score which drags this metric to extremely poor levels, even when the *Recall* had an acceptable rate of 0.65. This is a strong indicator of a very ill-conditioned problem, where the class imbalance strongly affects the detection of positive cases over all positive predictions made. For

this case, the positive class samples were less than 1% of all samples.

4.3.5 ROC Curve

One last important metric that is generally used in classification tasks is one related to the "Receiver Operating Characteristic" (ROC) curve. The metric itself is a measure of the **Area under ROC curve** ($ROCAUC$) and it applies only to algorithms which output for each sample the probability of belonging to the positive class.

With these algorithms we would have that the output label will be defined as

$$\hat{y} = \begin{cases} 1 \text{ if } f(x) > \pi \\ 0 \text{ else.} \end{cases} \tag{4.3.1}$$

where π is a threshold value which will set the algorithm's decision level, that which separates positive from negative samples.

If we consider different values for π, we will see that the true-positive rate TPR, or recall, and the fall-out FPR of the algorithm will vary depending on this confidence level π. With this, we can define the ROC curve to be

$$\sigma(\pi) = (TPR(\pi), FPR(\pi)) \tag{4.3.2}$$

As expected, there exists a functional relationship between these two as the threshold is varied. The image of $\sigma(\cdot)$ is a curve defined in $[0, 1] \times [0, 1]$ which is referred to as the *ROC space*.

To find a balance between these two rates, the $ROCAUC$ metric measures the integral of this curve in ROC space. The score calculated is thus known as *Area Under the ROC Curve*.

This metric follows the same properties as the ones mentioned before, where the best classifiers have values closer to 1.[3]

The following figures are example ROC curves for two of the problems defined from our dataset.

Notice the algorithm's poor prediction performance where the $ROCAUC$ output is near to the 'random' line. This line constitutes the performance of a *random* classifier which arbitrarily labels samples as being to each possible class. Under normal circumstances we expect to construct learners that have better $ROCAUC$ scores than a *random classifier*. To do this fit, we configured the Tree's hyperparameters with a bad nature and, in effect, this outputs a learner which has a poor performance in the training error.

As another example, the same algorithm was run but on Problem 1 and using the same available data as in Figure 4.3.2. For this problem we are looking for those

3. As a side note, the $ROCAUC$ measure is proportional to the statistic of the Mann-Whitney U-test, where the classifier's mean output positive and negative classes are compared. More information on this can be found in [Mason and Graham, 2002].

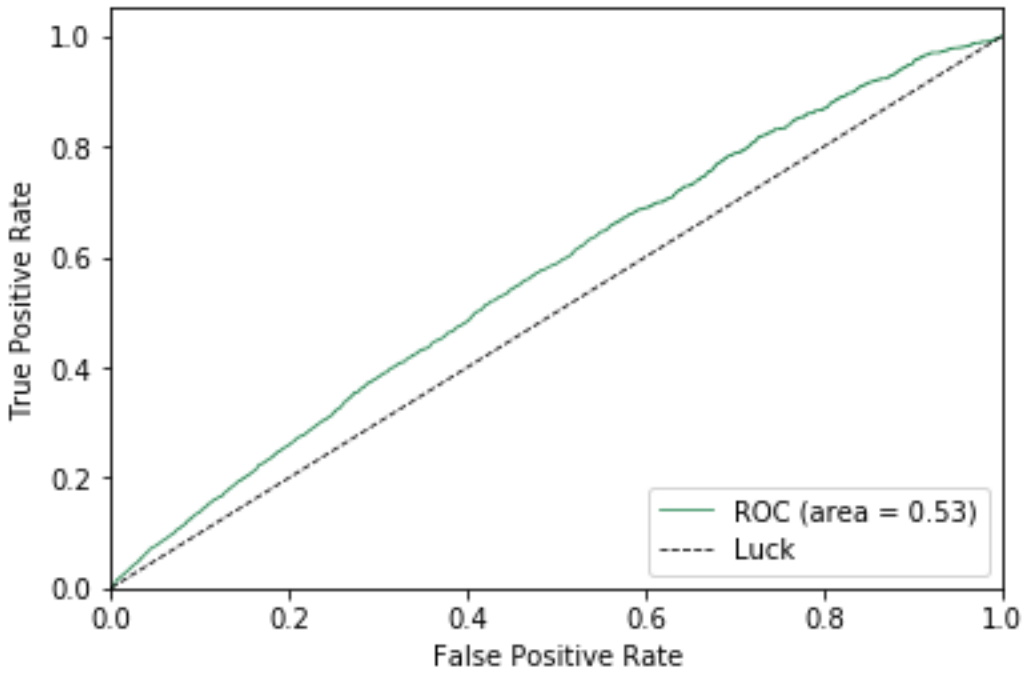


Figure 4.3.2 Example of a ROC curve from a Decision Tree classifier used on Problem 2. A 8-fold CV procedure is fit and then the best learner's performance is evaluated on $\mathcal{T}_f$ which gives score of 0.53. This model was specifically built to overfit the dataset by constructing a tree of 12 levels.

users that had lived in the endemic region in the past regardless of their current endemic situation. This problem does not suffer from class imbalance, where the positive class constitutes 30% of all samples. In Figure 4.3.3 we show a similar figure as in Figure 4.3.2 to illustrate the difference "ROC" curves that result from this optimization.

The results in this case is that the algorithm has a considerable better score by the *ROCAUC* metric. No data transformation has been applied to the dataset between runs. The difference in the two scores is notable and corresponds entirely to the problem chosen.

4.3.6 Final experiments on model selection

As a last remark for this chapter, we show here a brief experiment with a Logistic Regression Classifier. In this, we explore different hyperparameter configurations over a full CV procedure.

The idea is to predict which users lived in endemic regions in the past (again this is Problem 1), yet excluding the feature that indicates in which state the user is currently living. We also set the "ROC AUC" metric to compare the $l1$ vs the $l2$ regularization of the model on more than a hundred values for the regularization

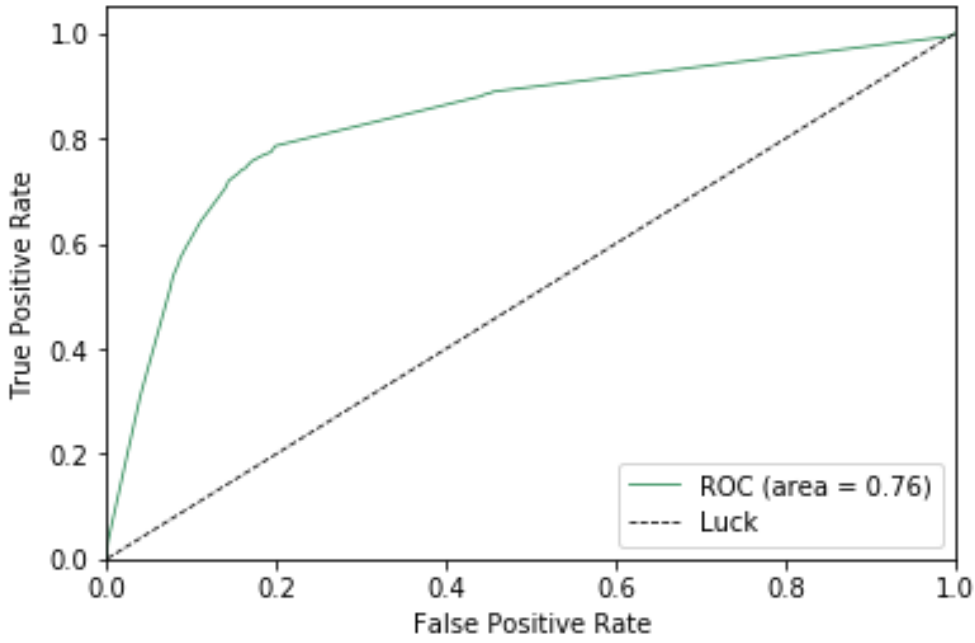


Figure 4.3.3 Example of a ROC curve from a Decision Tree classifier used on Problem 1. An 8-fold CV procedure is fit and then the best learner's performance is evaluated on $\mathcal{T}_f$ which gives score of 0.76.

parameter C.

For both regularization types, we set a value for C that ranged from $\exp(-2)$ to $\exp(5)$. Over all, we implement a cross validation routine of 8-folds.

Table 4.3.2 shows a summary of results. In these, our experiments resulted in higher values for $l1$ regularized models. This gap is gets wider when comparing test scores for both and the same happens to the time taken to fit the cross validation algorithm for all of the C values.

Table 4.3.2
Table of results comparing an 8 fold cross validation fit for a Logistic Classifier with varying regularization C values for Problem 1. The metric for this experiment was the "ROC AUC".

Regularization type	Best CV C value	Mean CV score	Test score	Full CV time (s)
$l2$	1e-5	0.805	0.744	17579.7
$l1$	1e-5	0.81	0.76	15892.9

Another interesting result is that in both experiments, the best CV scores were achieved for highly regularized models. To further explore this, we looked at their CV mean scores for each hyperparameter setting. Figures 4.3.4 and 4.3.5 both show

the series resulting from each type of regularization fit.

From these figures it is clear that both models improve the more regularized they become. The full extent to which this regularization would keep improving the score is not explored because both experiments' C parameter had a minimum at 10^{-5} which corresponds to the highest score for both. Also, there is a difference in the stability of the fitting, where the scores are more noisy for the $l2$ regularization at high C values as can be seen in Figure 4.3.5.

On the other hand, the $l1$ regularization has a more stable performance across C. This can be an indication that the $l1$ experiment required less iterations to fit and find an optimum.

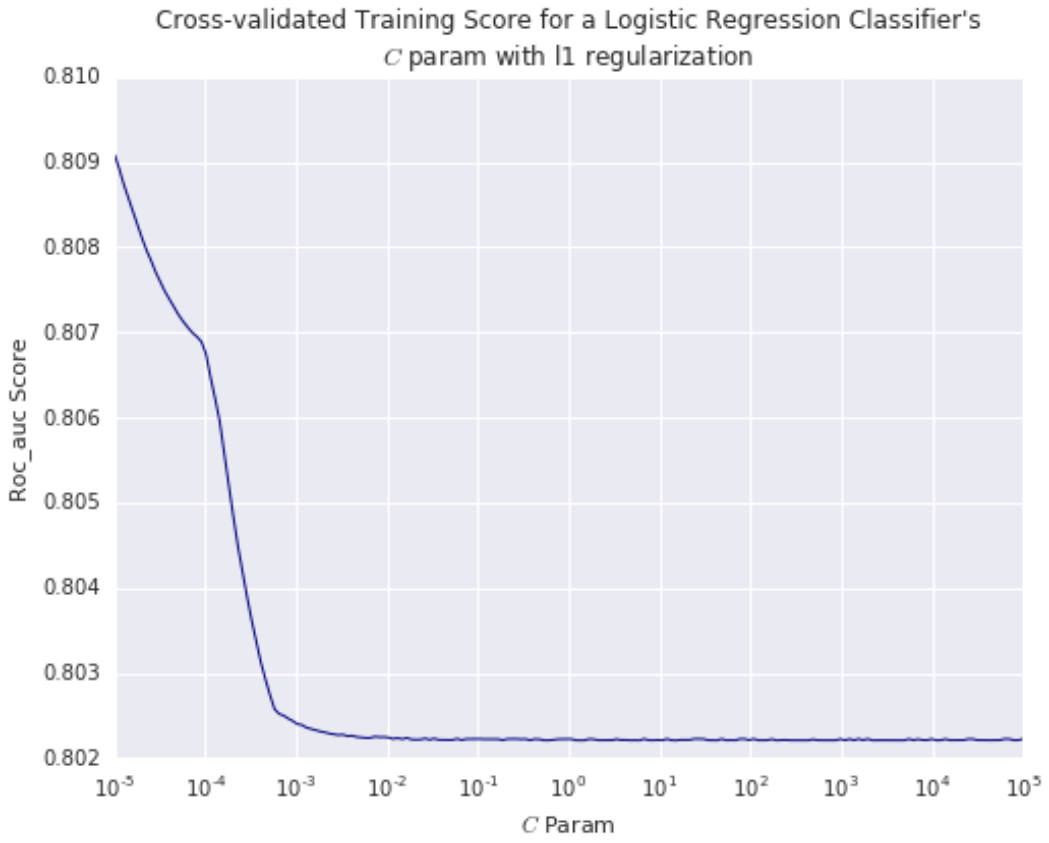


Figure 4.3.4 CV mean average scores of the "ROC AUC" metric for Problem 1. The $l1$ regularization method on a logistic classifier was tested, varying along the C hyperparameter.

With these experiments we give an extended overview of how a complete model-selection pipeline is defined and we give examples of how we must iterate and evaluate different hyper-parameter configurations by scoring the cross-validated learners.

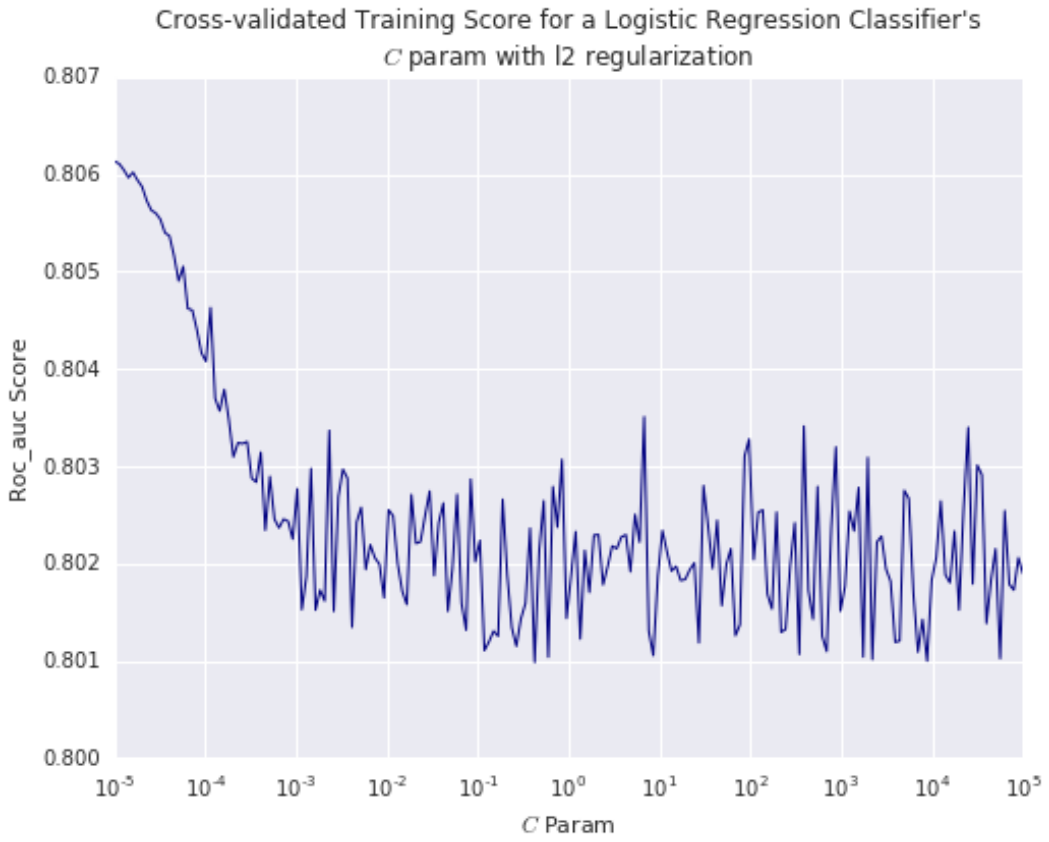


Figure 4.3.5 CV mean average scores of the "ROC AUC" metric for Problem 1. The $l2$ regularization method on a logistic classifier was tested, varying along the C hyperparameter.

Chapter 5

Ensemble Methods and the Naive Bayes Classifier

This chapter begins with formulations and introductory discussions of the most common tree-based classifiers, and finishes with the formulation of the Naive Bayes learner. For all of these models, we will present their advantages and disadvantages as predictors, along with a comparison of the model's performance. This assessment will compare models' benefits and drawbacks in a specific long-term migration experiment using processed CDR data.

5.1 Classifier: Decision Trees

Decision trees are models that can be understood as a tree-like graph structures in which each node of the tree contains a conditional statement. These conditional statements act as a rules which split input samples in a binary relationship of two disjoint sets. In this way, we have that at each node samples split in two possible branches which recursively continue to further nodes which again split the samples. The rules are propositions or statements that are either true or false and they are built as a linear (or similar) binary partition over feature space. In the algorithmic implementation each rule will be defined by assigning a threshold for a given feature which splits samples over those above and below this threshold. In this way, at each node the model evaluates each input sample x with the proposition by checking if $x^j \in U$, where x^j is the feature used for that node's rule and U is a subset of the feature's domain. After the split, the algorithm recursively partitions each of the resulting splits with another rule.

For numerical features, the input space X will be partitioned into L and R, namely left and right, where L takes the form $(-\infty, d]$. The value $d \in \mathbb{R}$ is a number predefined by the rule itself and is actually computed by the algorithm during the optimization procedure. In the same way, for categorical features L will be a subset

of the possible category values for that feature.

By iterating this process over more nodes, a tree defines a partition of feature space in multiple regions $A_1, \ldots, A_K$. Here the number of iterations $K \in \mathbb{N}$ is a hyperparameter which is dependent on the algorithm's convergence criterion. Each region will have an associated value c_k in such a way that the tree's predicted output $\hat{y}$, for a sample x, is c_k, when the sample belongs to A_k. Here c_k will be one of the possible values taken by the target variable y in the training set. If we take into consideration the way trees are built, we will have that each A_k is a hyper-rectangle in feature space.

5.1.1 Decision Trees Formulation

In short, the learner can be characterized by the following formula:

$$h(X) = \sum_{k=1}^{K} c_k I(X \in A_k) \tag{5.1.1}$$

where c_k is the output value that our model estimates for samples in the A_k region. Both of these will have to be learned by the model during optimization.

By the principles given above, the algorithm will need to determine an "optimal" way to split a set of samples, that flow through a decision node. In this learner, the *goodness of splits* are measured by specific functions called *node impurity measures*. Later in Section 5.1.2 we give specific examples of node impurity measures.

Most variations for this Machine Learning model build rules (tree nodes) in a sequential, greedy, fashion, where node impurity measures are locally optimized at each node to decide on which is the best splitting value. The reason for doing this is because the construction of optimal binary decision trees is NP-Complete [Hyafil and Rivest, 1976]. Doing otherwise would result in an algorithm whose computational complexity is infeasible.

At any splitting node we have to find the *best* feature $X^j \ / j \in \ [1, \ldots, p]$ and value split d for which to partition the data in

$$A_L(d, j) = \{x \in \mathcal{T} \ / \ x^j \leq d\}$$

and

$$A_R(d, j) = \{x \in \mathcal{T} \ / \ x^j > d\}$$

Let N_l and N_r be $|A_L|$ and $|A_R|$ respectively and note that we have taken (p, j) as given. However, the Decision Tree algorithm will find both in an optimization routine. To quantify the *best* feature $j \in \ [1, \ldots, p]$ for this split, the algorithm will minimize:

$$\min_{j,d} [\min_{c_L} \frac{1}{N_l} \sum_{x \in A_L(d,j)} L(y, c_L) \ + \min_{c_R} \frac{1}{N_r} \sum_{x \in A_R(d,j)} L(y, c_R)] \tag{5.1.2}$$

where y is the target associated to the samples x that are part of that node's split. Also, $L(\cdot)$ is the loss function used by the algorithm to measure the quality of the split in this optimization. Note that this can be done efficiently for a wide range of loss functions since the minimization can be done for each feature independently.

A tree is then grown in an iterative way from the top down[1], estimating the appropriate parameters at each rule split. All of the training set's samples would start at the top (the root node) and then travel down through the tree's branches, where at each step the node's rule will decide to what branch below does the sample move to. A branch of the tree would stop growing once the samples at a node have reached a certain *purity* i.e. when samples belong to the same class. Other stopping criteria include halting when a minimum number of samples has been reached.

We would finally have that the tree's leafs are the partition subsets over the input data and, once a tree is built, predicting targets for new samples is straightforward: the prediction of their target class will be the value given after traveling the sample down to its corresponding leaf node.

To illustrate a constructed tree using this method, an instance of a grown tree is shown in Figure 5.1.1. This classification tree example is built from a dummy gender prediction problem, using CDRs as input data. We must note here that there is not much interpretability in this toy example:

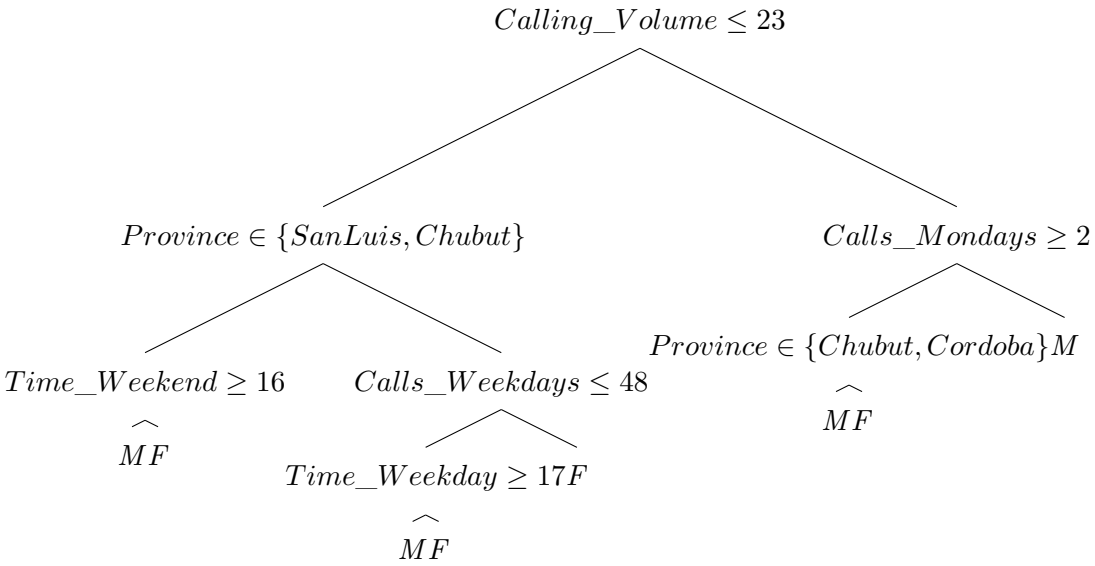


Figure 5.1.1 Classification tree example built for a toy gender prediction problem, using CDR available data.

1. In this context the *top* of a tree refers to the root of the tree.

5.1.2 Impurity Measures

The most used metrics to build each rule are the *Gini impurity measure* and the *entropy* or *information gain* criterion. The former minimizes the misclassification error in the output sets that result from the partition. It optimizes the model's accuracy as the resulting score of tagging all the leaf's samples with the majority label in that partition.

The latter measure optimizes for information entropy, which is analogous to minimizing *Kullback-Leibler divergence* of the resulting sets with respect to the original set prior to the split.

In addition, we have the misclassification measure. Following is listed formulation of the these three impurity measures for classification trees:

- Gini index: $\sum_{k \neq k'} \hat{p}_{jk}\hat{p}_{jk'} = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$
- Cross-entropy: $\sum_{k=1}^{K} -\log(\hat{p}_{jk})\hat{p}_{jk}$
- Misclassification error: $\frac{1}{N_j} \sum_{x \in R_j} I(y \neq c_j) = 1 - c_j$

For the case of binary (two class) classification, these measures can be expressed in simpler terms. If we consider p to be the probability of success, then we have

- Misclassification binary error: $1 - max(p, 1 - p)$
- Gini binary index: $2p(1 - p)$
- Binary cross-entropy: $-\log(p)p - \log(1 - p)(1 - p)$

5.1.3 Hyperparameters

For the decision tree model, the process of iteratively partitioning the samples in splits continues until a predefined tuning parameter stops the optimization or when the node is pure i.e. there is only a single target class for all samples at the node.

The hyper-parameters for this model includes the length of the tree, the splitting rule threshold and the node impurity measures. From the descriptions previously given, we can list these directly:

- Max depth of the tree, or the allowed levels of splits.
- The criteria or measure used to select the best split feature at each node.
- The leaf size or the total number of minimum samples allowed per leaf. Note that this is a related to limit on branch depth.

- Number of features selected to decide on the best split feature at each node.

Intuitively, it is natural to find that trees of longer depth will overfit the data since more complex interactions among variables will be captured by refining the input space partition. A trivial example is to allow a tree to grow fully in depth to later assign all training set's samples to their own self-contained regions. This will yield a model with virtually zero bias yet with a very high prediction error.

In Figure 4.2.1 we give an overfit decision tree example, using the Problem 2 problem of tagging outward migrants of the endemic region. It is clear that the overly-complex model produces poor generalization error. Here the training and CV scores were both compared on models with increasing tree depth. We see how at some point, the learner started to overfit the training set where this is reflected in the gap between the CV and training errors.

At the same time, having a tree which is too shallow in depth will result in a biased algorithm for most cases. This is because it results in an overly simple model incapable of correctly assigning labels. We must then consider that the depth of a tree is a measure of the model's complexity and as such, one of the most important hyperparameters of our model.

Another drawback of the decision tree model is the high variance instability. Authors point out that two very similar datasets can grow two very different resulting trees. This is due to the hierarchical nature of the splits, where errors randomly made in the first splits will be carried later on. Once a sample has been directed through a lower branch, it will continue down through this one without reconsideration of the past errors.

For the reasons described, in this model it is important to control the depth of the trees built. To do this, the most common method for doing this grows a very large tree T_0 that will continue until it reaches a depth limit threshold that is very nonrestrictive. Then the tree will be pruned by removing branches and nodes to lower the model's complexity whilst at the same time trying not to compromise much of its accuracy. More details on this can be seen in Section C.1. For a broad characterization of decision trees and their construction in classification or regression problems, please refer to [Breiman et al., 1984].

5.1.4 Experiment

We present here a run of a decision tree over $\mathcal{T}$. Again, the case the Problem 2 was used as an example, where we intend to correctly tag those TelCo users which were living in the endemic region in T_0 and then migrated to T_1.

The tree was built using a standard configuration with a low depth of 5 and a Gini splitting criteria. At each decision node, we configured the algorithm to be able to select any feature X^j to perform the split. Also, we configured the minimum split threshold to be of 20 samples. This meant that no further splits were created when less than this number of samples was left at that node. Under this setup, the algorithm ran in approximately 15 seconds. Figure 5.1.2 shows a partial

representation of the actual decision tree grown on Problem 2. This allows us to see how the algorithm selects better features at low-level nodes, versus the deeper splits.

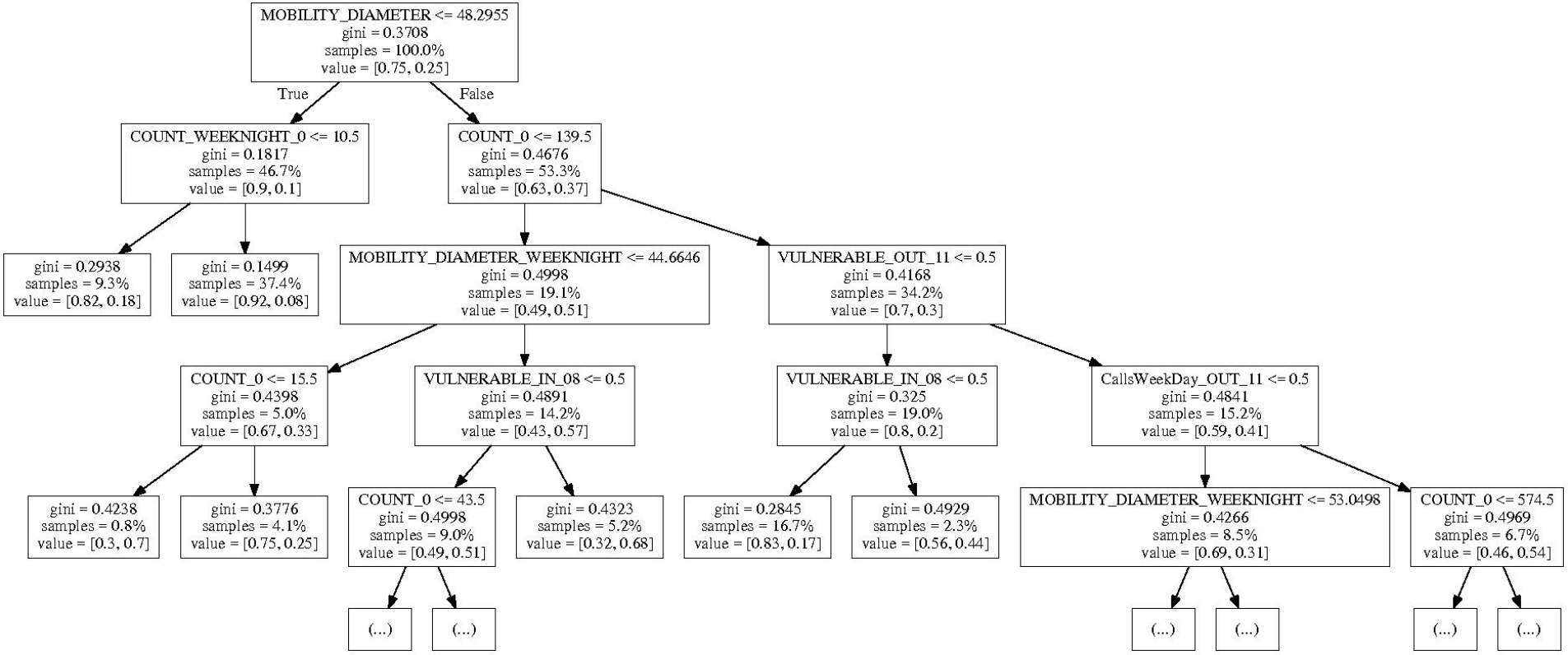


Figure 5.1.2 Example of a final decision tree. This model was made with a cross validated procedure on Problem 2 and the resulting tree has been pruned to fit the image. For every node, the samples are split into sets given by the implicit indicator function defined in the condition.

From Figure 5.1.2 we can approximate which features are most important to the algorithm when deciding a node split, given by the Gini splitting criteria. Given that in Problem 2 we are looking for people that migrated out of the endemic region, the choice of the tree's root feature seems appropriate. The mobility diameter gives an idea of a user's influence area when using their mobile phone and as such, this attribute might indicative of past migrations. Subsequently, the usage volume of a user's home antenna appears as the second most important features, both during weeknights and for the whole day.

The error score reached by this tree, however, yields a low performance over all of the samples because if we evaluate the learner with the *Accuracy* metric, we see a classification score of 43.2%. Under these circumstances, noting which features were selected near the root leaves of the tree can be misleading if we do not recheck these results. In this way we decide to iterate this model one more time, but now choosing to change some hyperparameter values. For this time, we set control the target class imbalance in the training set $\mathcal{T}$. As we have noted before, the positive and negative classes have a disproportionate balance of samples and this has direct impact in the overall model's poor performance. To help tackle this imbalance, we modify the tree's loss function's setup to assign higher weight to samples of the under-represented class and underweight the samples from the abundant group. We will have that for each sample, its loss weight was adjusted by its class representation ratio and, with this, the new run scored a 57.9% *Accuracy* which is more than a 15% increase over the previous score. The figure for this model is shown in Figure 5.1.3.

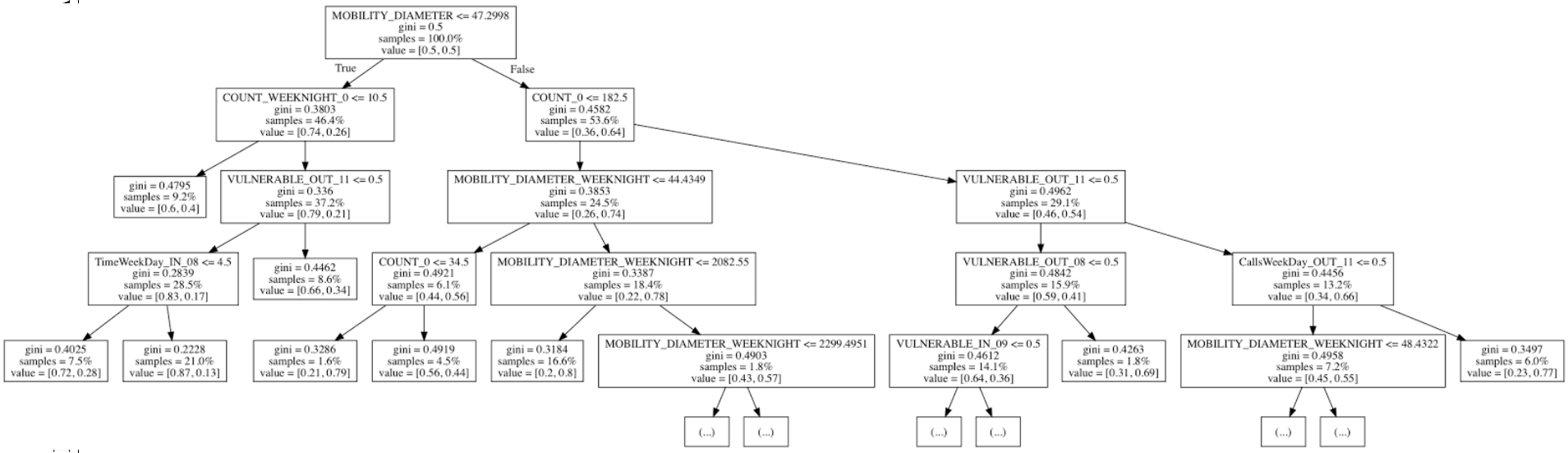


Figure 5.1.3 Second example of a decision tree. This model was made with a cross validated procedure on Problem 2 from an artificially balanced dataset. The resulting tree has been pruned to fit the image.

The importance of the mobility diameter becomes clearer in Figure 5.1.3. Once again the learner selects this attribute as the first split in the data, and this time with balanced samples. Recall that by the stochastic nature of the algorithm constructing the tree, two runs with the same hyperparameter settings and data need not output the same trees. In this case however, it is notable that the final tree still uses the mobility feature in the first split in the data, even with balanced samples. And the same can be said of the feature representing the count of calls made from the home antenna.

This tree has a better performance than before yet it has an average performance in its prediction of which users have migrated in the past. At some point, it is reasonable to assume that balancing classes leads to a better predictive outcome yet there are still various misclassifications in the predicted target classes. If we let the tree grow deeper though, we might increase the performance of the training error as shown in Figure 4.2.1. But again, this won't help our generalization error. So to tackle most of the disadvantages described by this technique, *Random Forests* are what follow as a natural extension to this classifier.

5.2 Random Forests

Random Forests are estimators that extend from constructing a group of single decision trees and then combining them to produce a single output decision. This grouping of classifiers is known as an *ensemble*.

The objective in this construction is to focus on decision tree's low bias whilst controlling their overfit as much as possible. For this, a forest of single trees will be constructed in such a way that correlations among each of the individual models is limited. The idea behind this is to have every individual model train on a random subset of features or samples and then, average the single outputs as the forest's target output. With this we preserve single estimator's interdependence as much as possible. There are other different techniques that intent to construct uncorrelated trees which are combined at the output, yet the main idea is common to all.

5.2.1 Random Forests Formulation

Let K be the number of trees in the ensemble and let Θ_k encode the parameters for the k-th tree. As we have mentioned before, there are various variants to the model and these variants will define the type of encoding for the $\Theta_k, k \in 1, \ldots, K$ parameters. For the following part, we will not specify the specific ensemble type constructed since the formulation is common to all of them.

To begin, we define:

$$h(\mathbf{x}, \Theta_k)$$

as the corresponding individual classifier and we let N be the number of samples in the training set $\mathcal{T}$. The creation of a random forest involves an iterative procedure where at the k-th step, the parameter Θ_k is fit from the same distribution as $\Theta_j,\ j < k$, yet

it is built in a way that is independent of the previous parameters $\{\Theta_1, \ldots, \Theta_{k-1}\}$.

Let $\{h_k(\mathbf{x})\}_{i=1}^{K}$[2] be a set of classifying trees and let $I(\cdot)$ denote the indicator function.

Define the margin function as

$$mg(\mathbf{x}, \mathbf{y}) = \frac{1}{K}\sum_{k=1}^{K} I(h_k(\mathbf{x}) = \mathbf{y}) - \max_{j \neq \mathbf{y}} \left(\frac{1}{K}\sum_{k=1}^{K} I(h_k(\mathbf{x}) = j) \right) \tag{5.2.1}$$

This function measures, in average, how much do the trees vote for the correct class in comparison to all other classes and it is the training error of the model when using the misclassification loss. Here the generalization error is denoted as $PE*$ and is equal to

$$P_{\mathbf{x},\mathbf{y}} \left(mg(\mathbf{x}, \mathbf{y}) < 0 \right) \tag{5.2.2}$$

It can be shown that, for K sufficiently large, the generalization error under the misclassification loss converges to

$$P_{\mathbf{x},\mathbf{y}} \left(P_{\Theta}(h(\mathbf{x}, \Theta) = \mathbf{y}) - \max_{j \neq \mathbf{y}} P_{\Theta}(h(\mathbf{x}, \Theta) = j) < 0 \right) \tag{5.2.3}$$

almost surely for all sequences of parameters $\Theta_1, \Theta_2, \ldots, \Theta_k, \ldots$

This proof can be found in Section C.3.

5.2.2 Experimental comparison to Decision Trees

From what is formulated in Section 5.2.1 and in order to compare how this model improves over the Decision Tree model, we ran two experiment setups on a Random Forest learner for the difficult Problem 2. We selected the *Accuracy* score to measure the learner's performance. The model was trained in a cross validation procedure of 10 folds for each configuration, and in each experiment we optimized the score for a different model hyperparameter.

For the first experiment, we evaluated the forest's scores on a maximum depth variation for the trees. On the second one, we tried to illustrate the point of Section 5.2.1 by considering forests of increasing size. As a default configuration, we set both the trees' max depth and the number of trees to 10.

Also, we pre-configure a balanced weighting of positive and negative samples in the loss function, where each sample is weighted by the reciprocal of the class's ratio to the whole dataset. Finally, the impurity measure we used was the Gini index for the splitting criteria at each tree's node.

---

2. There is an abuse of notation by noting trees as $h_k(\mathbf{x})$ and not $h(\mathbf{x}, \Theta_k)$

Cross validation procedures ran for 1412.8 and 2506.1 seconds respectively. In both of them we excluded any user attributes that informed their home state and current endemic condition, in the time period T_1.

The experiment's scores outcomes across their hyperparameter values were graphed in Figures 5.2.1 and 5.2.2. For both, the cross-validated and training set scores were compared. In this way, for each hyperparameter value, the $1 - Accuracy$ score is shown. In this way, having a lower score in the graphic means a better model was reached.

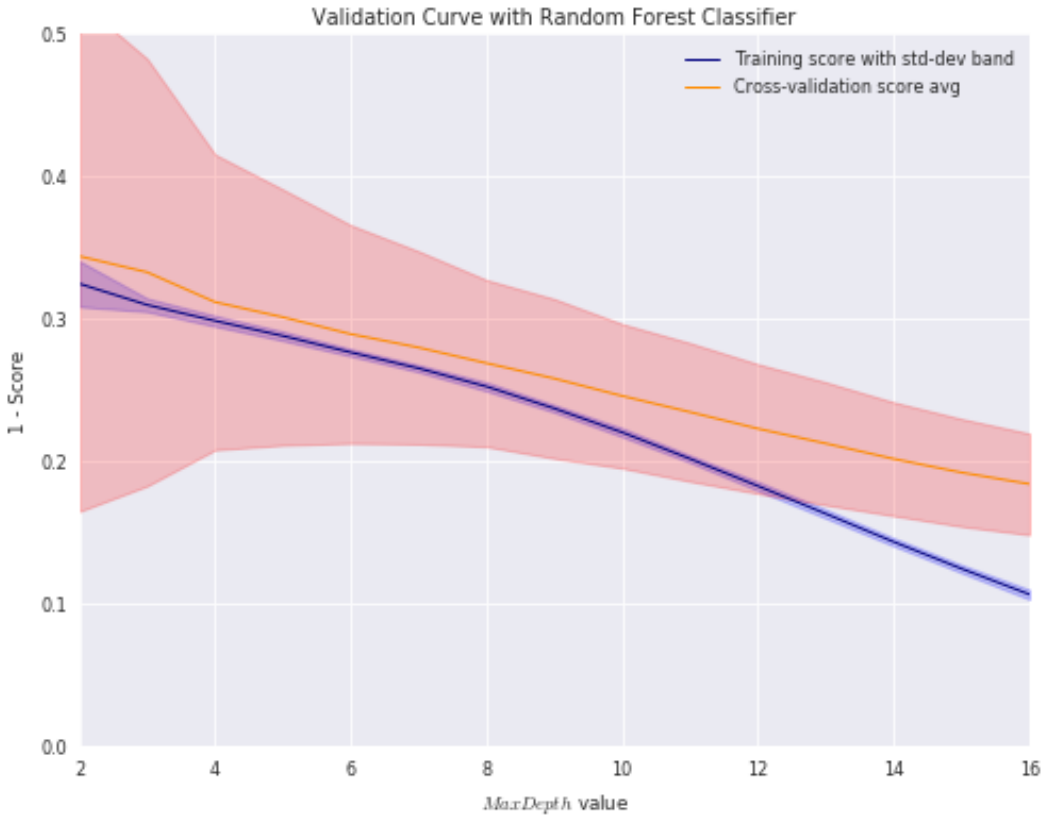


Figure 5.2.1 Validation curve on the tree-depth hyperparameter values for the Random Forest learner. The mean CV scores for the $1 - Accuracy$ score is shown for a CV run experiment on Problem 2

Recall how this relates to Figure 4.2.1, in which a tree's training error always improves as a result of an increase in the tree's depth. Yet we saw how the tree's generalization error starts to suffer and deteriorate at one point.

From what we can see in Figure 5.2.1, the cross validated error is always decreasing for increasing tree depth. A very slight deceleration in the decrease of the cross validated error can be seen for tree depth greater than 9. However, scores keeps improving and don't worsen. This is different to Figure 4.2.1 where if we let the tree

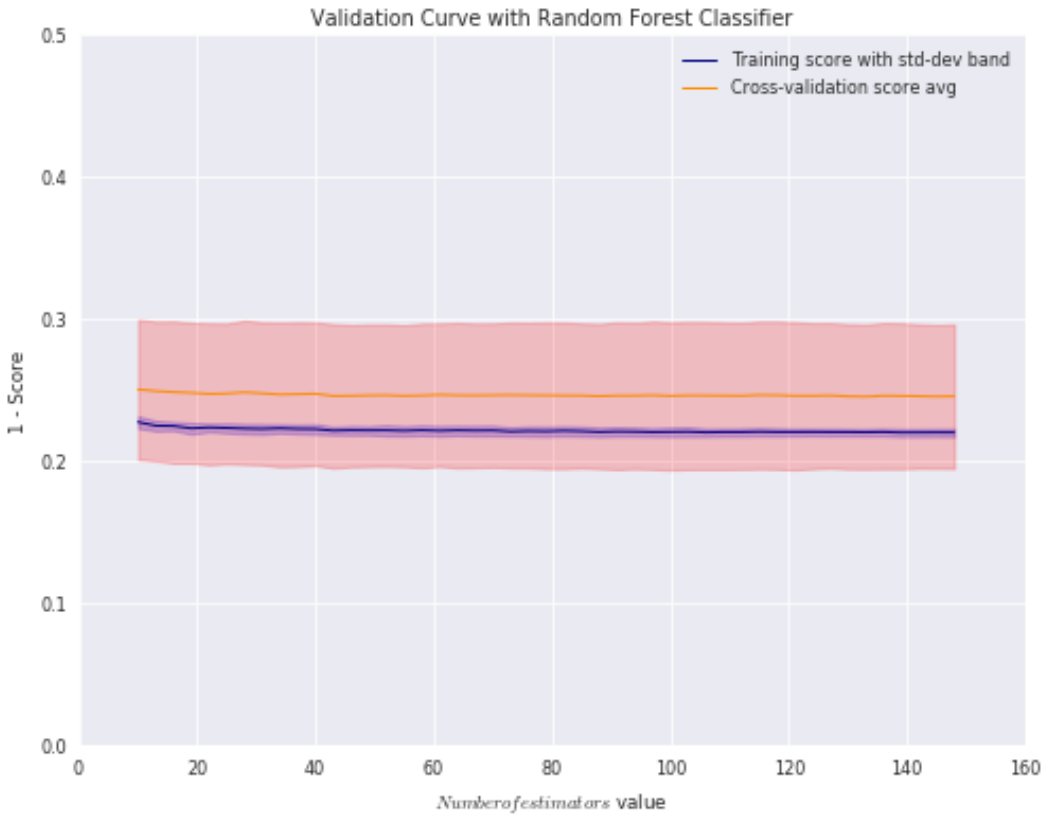


Figure 5.2.2 Validation curve on the number of trees hyperparameter values, for the Random Forest learner. The mean CV scores for the $1 - Accuracy$ score is shown for a CV run experiment on Problem 2

grow deeper, at one point we only worsen the performance of the test error. Here, we are possibly improving the generalization error, as indicated by the CV estimation.

On the other hand, it is interesting to note that there is an insignificant decrease in the training and CV scores when the number of trees is increased. For this case, there is virtually no change in model error by improving bias or variance.

An explanation for this might that we are setting a default tree depth of 10 levels already when running this experiment. And a tree configuration of this type might then be capturing all the possible model complexity to this dataset. Another plausible explanation for this is that, for this task, the forest's performance rapidly converges to the generalization error with a minimum of 10 trees and that adding more trees doesn't certainly improve on the predictions.

At their optimal configuration, both setups reached CV score averages of at least 75%, with a maximum of 81% for the forest with trees of 16 levels. The optimal number of estimators for the CV and the training sets were not the same though, where 139 was the best configuration for the training score whilst 133 was the best

average for the CV set. The relative difference in these best scores was of 3.3%.

At the same time, the relative difference of the best CV and training scores, for the maximum tree depth experiment was of 9.5%. This percentage difference signals that the max depth hyper-parameter is significantly more sensible to over-fitting than the number of trees used.

5.2.3 Predictive error bounds

Random Forests are built upon a bag of weaker classifiers, of which each individual estimator has a different prediction error. To build an estimate of the generalization error on the ensemble classifier, these individual scores and the relationship between them must be measured. In this sense, the *strength* and *correlation* of a Random Forest must be analyzed to arrive on an estimate of the generalization error.

Define

$$\hat{j}(\mathbf{x}, \mathbf{y}) = \underset{j \neq \mathbf{y}}{\operatorname{argmax}} P_{\Theta}(h(\mathbf{x}) = j)$$

and let the margin function for a random forest (not a group of classifiers) be defined as

$$mr(\mathbf{x}, \mathbf{y}) = P_{\Theta}(h(\mathbf{x}) = \mathbf{y}) - P_{\Theta}(h(\mathbf{x}) = \hat{j}) = \mathbb{E}_{\Theta}\left[I(h(\mathbf{x}, \Theta) = y) - I(h(\mathbf{x}, \Theta) = \hat{j})\right]$$

This characterizes the expectation taken over another function which is called the **raw margin function**. Intuitively, the raw margin function takes each sample to be 1 or -1 according to whether the ensemble classifier can correctly classify or not the sample's label, given the ensemble's structure specified by Θ.

With this, we can introduce the strength for this forest as

$$s = \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[mr(\mathbf{x}, \mathbf{y})\right] \tag{5.2.4}$$

Now define $\rho(\Theta, \Theta')$ as the correlation between $rmg(\Theta, \mathbf{x}, \mathbf{y})$ and $rmg(\Theta', \mathbf{x}, \mathbf{y})$ of two learners, then we can have the mean value correlation for the ensemble as

$$\overline{\rho} = \frac{\mathbb{E}_{\Theta,\Theta'}\left[\rho(\Theta, \Theta')\sigma(\Theta)\sigma(\Theta')\right]}{\mathbb{E}_{\Theta,\Theta'}\left[\sigma(\Theta)\sigma(\Theta')\right]} \tag{5.2.5}$$

With the above we can then see that,

Theorem 5.2.1 *There exists an upper bound for the generalization error of a Random Forest which is*

$$PE^{*} \leq \overline{\rho}\frac{(1 - s^2)}{s^2} \tag{5.2.6}$$

The proof of this can be found in Section C.2.

This bound on the generalization error shows the importance of each individual weak classifier's strength to the forest's generalization error, and the correlation interdependence among them.

This proof was first introduced in [Breiman, 2001a] and there, it is said that the bound may not be tight enough for practical significance. Special importance is also put on the ratio between base learner's correlation and strength ($\frac{\bar{\rho}}{s^2}$). It is know that, to build a strong classifier, this should be as small as possible. For this reason decision trees make a canonical choice for weak learners, being low biased models which are prone to overfitting. These ensembles are a first choice when building a first, cost-effective, model to deliver a prediction task with acceptable to excellent performance.

5.2.4 Other Notes on Random Forests

One benefit of building Random Forest classifiers is that the algorithm can easily increase a group of estimators' prediction error by randomly building every specific learner in a way that decreases the overall model's variance whilst trading a small loss in bias.

The model is also robust to the introduction of noisy features where if the ratio of informative to non-informative features is not extreme, selecting m features at random for each split will mean that, in most cases, splits will be made on those informative features. Note that in any given tree, the probability of drawing at least one informative feature in a split is still very high. This is because it follows a hyper geometric distribution $\mathcal{H}(P, j, l)$ with l draws from a total population of P features and only j informative ones.

The depth of growth for each tree is another important tuning parameter. We must choose it correctly by assessing the model's performance across different values for m. A deep tree will tend to overfit the data by partitioning input space to fit the training data. This effect will counter the overall reduction in variance of the forest and thus increase the generalization error of our algorithm.

In addition, the algorithm benefits from a heuristic to measure variable importance, where a special modification in the way forests are built allows this to happen. The idea for this is that at each split we can measure the gain of using a certain variable for the split versus not using it. Given a candidate feature X^j to be analyzed and for every node in a tree where a split is to be done, we compare the improvement in split performance, as measured by some loss function, with and without X^j. These results are recorded and averaged across all trees and all the split scenarios to have a score for the feature. With this, the features with highest scores can be thought to be the most informative variables of the model.

5.2.5 Experiments

In this subsection, we explore on the task described on Problem 4 by fitting random forest learners. The idea was to look for those same users that had migrated out of the endemic region from T_0 to T_1, but only focusing on those users that are not endemic in the present.

Here, our cross validation procedure explored multiple combinations of hyperparameters that we predefined. Let α_i be the value of a the hyperparameter i, where i is one of the J possible hyperparameters for this model. Denote all possible hyperparameters combinations by $A_1 \times A_2 \ldots \times A_J$, where each α_i will take a value from the predefined set A_i $\forall i \in 1, \ldots, J$.

In practice, we must be careful with the size of these combinations since the full cross validation procedure is costly for the Random Forest algorithm. The learner has $J \geq 10$ and for each component A_i it is normal to have more than 10 possible values. As such, the search space can grow exponentially big with the size of each hyperparamter's values. Added to this, each hyperparameter combination has to be cross validated on the number of K folds we set, creating a very large amount of iterations for the whole search. Training the forests can be computationally intensive in computer memory and CPU time, so we have to limit the search over a bounded amount of values.

If we were to use commodity hardware and standard programming tools for this procedure, the experiment would take weeks and, given our dataset size, it would not fit into a standard computer's memory of less than 12GB of memory. For these reasons, we preform the task with a specialized server with a 16 core CPU and 72 gigabytes of available RAM. The system runs a UNIX based OS and our algorithms were scripted in Python 3.5 [Foundation, 2015]. We used Graphlab, [Low et al., 2012], and Sckit-Learn, [Pedregosa et al., 2011], two specialized Machine Learning Python packages, to handle the parallelization and the distribution of system resources.[3]

This package has allowed us to conveniently run these long experiments without risk of failure due to lack of memory and without any prior expert domain knowledge in multi-threading or parallel computing.

The cross validation was done over the same hyperparameter configurations that we will later use for the general experiments procedure. The following list describes the hyperparameters used:

- Every tree's maximum depth — [*max_depth*]
- If the loss function will balance the weights of the positive and negative classes — [*balanced*]

3. We can't stress enough how convenient these packages are to our purposes. Graphlab comes specially relevant when handling most of the complex tasks of maxing out the system resources to perform the the cross validation procedures as fast as possible. It also helps boost productivity by improving the time it takes to construct the main X dataset. This package has an *out-of-the-box* memory management for large datasets. With this, we can natively handle data that is larger than the amount of memory used by the server, requiring minimal user knowledge to tune.

- The maximum number of trees to grow in each fit— [num_trees]
- The splitting criterion at each node which can be either by the Gini Index or the Entropy condition— [split_criteria]
- The sample percentage of features which are available to use at each split — [$max\_features$]
- The minimum number of samples required to split a node — [min_split]
- The scoring function used to evaluate the learner during cross validation — [$CV score$]
- The minimum reduction in the loss function required to split a node — [min_loss]

The full experiment ran two different tasks, where each was characterized by the attributes used for our X dataset. In the first one, the procedure ran with all available features and in the second one we separated the features that had a high correlation, bigger than 0.3, with the target variable. This features that met this criteria were those that counted the user's vulnerable neighbors for a given month, segmented by the direction of the call.[4] Recall that all of these features were central to the construction of the risk maps in Section 2.3. The color of each antenna's circle represented the amount of vulnerable users, for that given month, when looking at the incoming or outgoing calls.

For this experimentation, we recorded the cross validation procedure's outcome in terms of different scoring metrics. We also logged the run-time, the hyperparameters chosen and the top ten best features as given by the algorithm. A summary of these results is given in Table 5.2.1.

From the result's table, it is relevant to note that both optimal learners chose the Gini splitting criterion as best, with tree's of a relatively high depth with 12 levels. A similar situation arose with the number of trees grown; best configurations had more than 100 trees with the second experiment having an optimal number of 200.

Higher differences appear in the choice of the minimum loss reduction and the features sampled per split. There are no clear indications as to which configuration is better. And, whilst having a higher level of randomization during tree growth seems to produce learners which are less overfitting, this difference is not that notable in the CV scorings.

For most of the metrics used to evaluate the CV procedure, the scores look very similar. It is interesting that the second experiment is slightly better in the $Accuracy$, $Log - loss$ and $ROCAUC$ scores, with a maximum difference of 5% for all of them. Yet simultaneously the second experiment's performance drastically drops for the recall and precision. The same occurs accordingly for the $F1$ measure. We see the learner is losing performance in correctly tagging the user's who have migrated from the endemic region. This high relative difference hints to how much of an impact is

4. For a complete description on the definition of these features, please refer to Table 2.4.1.

Table 5.2.1
Table of best results comparing two 10-fold full grid cross validation procedures on a Random Forest Classifier fit for Problem 4. The scores, best hyperparameter values and run-time are shown for all experiments.

Result	Experiment 1	Experiment 2
Running time (s)	2030	1264
CV $F1$ score	0.373	0.296
CV *Accuracy* score	0.880	0.903
CV *Precision* score	0.267	0.213
CV *Recall* score	0.618	0.480
CV *ROCAUC* score	0.843	0.848
CV $Log - loss$ score	0.337	0.391
max_depth	12	12
num_trees	150	200
balanced	True	True
split_criteria	Gini	Gini
min_loss	10	1
max_features (%)	50	80

characterized by removing the features in the second experiment. We can suspect that there's enough missing information in those features to reduce the predictive power of the learner.

In both experiments, it is important to note though that the elected hyperparameters are not optimal in any strict sense. At most, they can hint to the best possible configuration. Given the large combinatorial nature of this process, there is no guarantee of optimality in this search. To a greater degree we are only covering as much space as our time and our computational systems allow.

We also recorded the model's selection of best features, following the heuristic described in Section 5.2.4. This outcome is presented in Table 5.2.2, where the top 10 features for each experiment are shown.

The top features list shows us the importance of mobility diameter for all cases, where we found that the user mobility is relevant to detect a past endemic condition, both for mobility specific to weeknights and, also, for mobility during the whole week.

As expected, in both cases we see that there are a number of features logging vulnerable interactions of users living in non-endemic regions. This is in line with what we hypothesized in the construction of the risk maps in Section 2.3. The direction, duration and number of call occurrences are all important predictors for

Table 5.2.2
A representation of the top features that resulted from the Random Forest experiment on Problem 4.

	Top Features		
First Experiment	Call Count Antenna_0	Mobility Diameter	Mobility Diameter Weeknight
	Call Count Weeknight Antenna_0	TimeWeekDay Out Month_09	CallsWeekDay In Vuln Month_08
	TimeWeekDay Out Month_08	State Hidalgo	CallsWeekDay In Month_08
		TimeWeekDay Out Month_12	
Second Experiment	Mobility Diameter	Call Count Antenna_0	Mobility Diameter Weeknight
	Call Count Antenna_1	Call Count Weeknight Antenna_0	TimeWeekDay Out Month_12
	TimeWeekDay In Vuln Month_09	TimeWeekDay Out Vuln Month_08	TimeWeekDay Out Month_09
		TimeWeekDay In Vuln Month_08	

this learner.

Given that this best feature methodology is an heuristic, there doesn't seem to be any clear indication over which type of CDR interaction has better predictive power. Yet there is a preference for interactions logged in the earlier months, August and September, This might be the case where the algorithm is picking up on the most recent migrations, given that our time period of analysis T_1 starts in August.

The analysis here presented is relevant to answer the question of long-term migrations. High scores for these experiments show that there is value in CDR data for predicting long-term migrations between two regions. We also see that the resulting top features from these models are in line with the assumptions on which the risk maps were constructed.

5.3 Boosting Models

Boosting models are similar to additive methods such as Random Forests, because they combine the predictions of weak learners to output their combined prediction. The full learner is grown sequentially from base estimators such as decision trees, but the difference is that each new iteration tries to reduce the overall bias of the combined estimator. This provides greater predictive power when the base model's accuracy is weak. However, care must be taken to control the increase in variance.

5.3.1 Ada Boost

In the Ada Boost variation of ensembles, each iteration builds a new weak learner which is set to improve on the samples misclassified by the previous ensemble of weak learners. The new learner will not be uncorrelated and this is an important distinction of this model with the previous one.

Weights are used by the algorithm to rank the samples by misclassification importance: a sample with higher misclassification rate will receive a stronger weight.

The algorithm's name is derived from the term *adaptive boosting*, where sample weights are updated at each iteration.

Tuning parameters in this algorithm are a superset of those used to build the base learners. As an addition this model adds other hyperparameters, such as the number of steps that the ensemble is *boosted*.

This chained construction of weak learners has implications on the computational complexity of the optimization. Base learners are not constructed independently and as such, the parallelization of this algorithm becomes limited. At the same time, the sequential optimization of learners improving on the one before marks a *greedy* minimization approach of the general loss function.

These properties underline a substantial difference to Random Forests where base learners are built as uncorrelated as possible and where optimization can be performed globally, which allowed for a significant runtime improvement by parallelizing the algorithm.

5.3.2 Formulation

Let

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq \hat{y}_i) \tag{5.3.1}$$

denote the training set's misclassification error. As usual, N is the amount of samples in our dataset, y is our target variable and $\hat{y}$ is our model's prediction for the target, given the samples. We also take

$$\mathbb{E}_{X\ Y}[I(Y \neq \hat{Y}(X))] \tag{5.3.2}$$

to be the expected error rate, or EPE, of the model on the true, unknown distribution of the data.

Let m index the iteration number in the Ada Boost algorithm. Set $w_i^{(m)}$ to be the i-th sample's weight at this iteration. We will initialize w to be equiprobable at $w_i^{(0)} = \frac{1}{N} \forall i$.

Let $h(x, \theta)$ denote a weak learner. With this notation, we assume the loss function to have a domain in the input feature space and in the parameters defining the learner. Naturally these will depend on the problem structure and on the base learner.

Then a fully grown Ada Boost's classifier takes the following form:

$$\hat{y}^{(M)}(x) = sgn(\sum_{m=1}^{M} \gamma_m h(x, \theta_m)) \tag{5.3.3}$$

where M is the model's hyperparameter indicating the amount of weak learners and thus the amount of iterations. Here, each θ_m will encode the base learner's parameters and γ_m will denote the weight of that weak learner in the overall model.

The algorithm's iteration will build $\hat{y}^{(M)}$ starting from $\hat{y}_i^{(0)} = 0 \ \forall i$ in such a way that $\hat{y}^{(m+1)}$ will improve the loss of the previous iteration $\hat{y}^{(m)}$. This is because at each stage, we will minimize a function that tries to correct the performance of the previous model. At step m we will search for (γ_m, θ_m) where

$$\begin{aligned}(\gamma_m, \theta_m) &= \underset{\gamma,\theta}{argmin} \sum_{i=1}^{N} L(y_i, \hat{y}^m(x_i) + \gamma h(x_i, \theta)) \\ &= \underset{\gamma,\theta}{argmin} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{m} \gamma_j h(x_i, \theta_j) + \gamma h(x_i, \theta))\end{aligned} \tag{5.3.4}$$

The greedy nature of the algorithm becomes explicit in the procedure above, where we have fixed all the previous optimized values for γ_j and θ_j.

Ada Boost was first derived in [Freund and Schapire, 1996] and it was introduced with a specific minimizing function. The general version here presented allows the use of a broad range of base learners which need not come from the same algorithmic family. In the first version introduced, the loss function used was the exponential loss which is $L(y, z) = e^{-yz}$ and the target variable took the values 1 or -1.

This particular case yields a similar equation as in Equation (5.3.4), but where

$$\begin{aligned}(\gamma_m, \theta_m) &= \underset{\gamma,\theta}{argmin} \sum_{i=1}^{N} \exp\big(-y_i(\hat{y}^m(x_i) + \gamma h(x_i, \theta))\big) \\ &= \underset{\gamma,\theta}{argmin} \sum_{i=1}^{N} \exp\big(-y_i \hat{y}^m(x_i)\big) \exp\big(-\gamma h(x_i, \theta) y_i\big)\end{aligned} \tag{5.3.5}$$

Given that we are only minimizing γ and θ, we can group $e^{-y_i \hat{y}^m(x_i)}$ into a single value $w_i^{(m)}$ which we will set to the weight of each sample. This weight strongly depends on the past steps of the algorithm. The equation now becomes

$$(\gamma_m, \theta_m) = \underset{\gamma,\theta}{argmin} \sum_{i=1}^{N} w_i^{(m)} \exp\big(-\gamma h(x_i, \theta) y_i\big) \tag{5.3.6}$$

We can then minimize for θ first, independently of the value of γ. The series in Equation (5.3.6) can be decomposed

$$\begin{gathered}e^{-\gamma} \sum_{i|y_i = h(x_i,\theta)} w_i^{(m)} + e^{\gamma} \sum_{i|y_i \neq h(x_i,\theta)} w_i^{(m)} = \\ (e^{\gamma} - e^{-\gamma}) \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq h(x_i, \theta)) + e^{-\gamma} \sum_{i=1}^{N} w_i^{(m)}\end{gathered} \tag{5.3.7}$$

and then the minimizing solution for $h(\cdot, \theta_{m+1})$ will be the one satisfying

$$\theta_m = \underset{\theta}{argmin} \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq h(x_i, \theta)) \tag{5.3.8}$$

Let $u = \sum_{i=1}^{N} w_i^{(m)}$ and $v = \sum_{i=1}^{N} w_i^{(m)} I(y_i \neq h(x_i, \theta))$, which are both constant in γ. Consider Equation (5.3.1) and note that $\frac{u}{v} = \frac{1}{\overline{err}}$. If we now solve for γ in Equation (5.3.7), we can take

$$f(\gamma) = (e^{\gamma} - e^{-\gamma})u + e^{-\gamma}v \tag{5.3.9}$$

which has a minimum at

$$\gamma_m = \frac{1}{2} \log \left(\frac{1 - \overline{err}}{\overline{err}}\right) \tag{5.3.10}$$

As seen from the equation above, the minimizing value for γ is directly related to the training error of the algorithm for the *whole* dataset. This weight will be reflected upon all samples in general and then we would expect this rate to decrease at every iteration. Taking advantage of this closed form, the value is plugged into the next step of the Ada Boost procedure to update sample weights as

$$w_i^{(m+1)} = w_i^{(m+1)} e^{\gamma_m(-y_i h_m(x_i))} \tag{5.3.11}$$

In this way, we have that the weights are updated for those samples which have a higher misclassification rate. This is a crucial aspect of the algorithm. At each step, more importance is given to misclassified samples over correctly classified ones.

The final form of the model is

$$\hat{y}(x) = sgn\left(\sum_{m=1}^{M} \gamma_m h_m(x)\right) \tag{5.3.12}$$

which outputs the most frequent prediction given by all of the weak learners. This is because all the correct predictions will be greater than zero and negative for the incorrect predictions[5].

At first the choice of the exponential loss function can seem arbitrary, but in the context of statistical learning this measure presents an important property. Here we have that, with this loss function, the optimal classifier from all learners with is the log-odds ratio of the two output classes for the trained dataset:

$$f^*(X) = \underset{f}{argmin}\ \mathbb{E}_{Y|f(X)}\left[\exp(-Yf(X))\right] = \frac{1}{2} \log \left(\frac{P(Y = 1 \mid X)}{P(Y = -1 \mid X)}\right) \tag{5.3.13}$$

5. This is when we consider the binary class case where Y can take only 1 or -1 values.

The use of an exponential loss function $\exp(-Yf(X))$ is also desirable in this context since significantly more weight is put on misclassifications rather than on correct classifications. This is because the function is not symmetric in the negative or positive predictions $\hat{Y}$ because we have that a correct classification will result in a weighting factor of only e^{-1}, whilst on the other hand, a misclassification will result in a factor of e.

A drawback of this loss though, is that it is not robust to outliers or to noisy data. During run-time weights are constantly shifting towards misclassified samples. Then if samples are mislabeled, the added noise this will make the algorithm repetitively focus on fitting to data which is incorrect.

As explained before, the boosting methods build an ensemble model learned from other *weaker* learners. If we consider decision trees as our base models, we are specifically looking at the case of *Gradient Tree Boosting*, which is a specific variant of the model above described. For the rest of this work we will focus on this specific variant.

This booster was first introduced by Friedman, J., and the full details of the work can be found in [Friedman, 2001].

5.3.3 Notes on Gradient Boosting Optimization

Given that each base learner has its own parameter θ and its weight γ in the booster's loss function, take

$$\Gamma = \{(\gamma_m \, \theta_m)\}_{m=1}^{M}$$

to be the general parameter of the whole learner. Note that if an optimization routine were to collectively fit all the parameters in Γ to fit this model, we would computationally have a highly difficult model to train. Instead, applications of boosters rely on optimization heuristics which use first and second order approximations of the loss function at step m to build on the next learner $m+1$.

To work with this in the AdaBoost formulation , smooth loss functions become very convenient for this procedure. As an example, we explain in detail a *Gradient Tree Boosting* optimization heuristic in Section C.4.

There are two additional heuristics commonly used to improve the booster's generalization performance. And for these, the arguments in favor of their use are rather experimental than theoretical. The authors in [Hastie et al., 2009] and [Bishop, 2006] mention them because of their overall contribution to the generalization error and because although they are intuitive to the idea of variance reduction.

The first idea to reduce the booster's variance is to subsample the data. This means that at each iteration, only a bootstrapped sample of the dataset will be selected to build a new weak learner. Samples from $\mathcal{T}$ which are not part of the bootstrapped sample are ignored when optimizing for the new learner at Equation (5.3.4). The motivation behind this is the same that as in Random Forests, where reducing the overall available data to fit the new weak learner will most likely reduce the variance

of the method. In practice, the rate of sampling will be supervised by a tuning parameter in the model.

The other heuristic, which was found to be more experimentally important by [Hastie et al., 2009], is to successively apply a *shrinkage* factor $v \in (0, 1)$ to the new model. At step m, instead of letting the overall model be

$$\hat{y_i}^{(m)} = \hat{y_i}^{(m-1)} + \gamma_m h_m(x_i) \tag{5.3.14}$$

we multiply the shrinkage factor v with the new learners before adding them to the overall model. In the literature this shrinkage factor is also called the *learning rate* of the algorithm.

Note that v is reducing the movement of the algorithm in the direction of optimization provided by γ_t and h_t. In practice, this results in longer iterations needed to reach the algorithm's *best* prediction rate. However, when this factor is combined with sub sampling, experiments have shown improvements in the overall generalization accuracy.

5.3.4 Experimental comparison to Random Forests

In order to compare this learner to Random Forests, we ran the same setup as in Section 5.2.2 with two experiments each varying a different hyperparameter. Again, models were built to classify on task Problem 2 and the *Accuracy* score was used to evaluate their performance. The cross validation procedure set 10 folds for each hyperparameter configuration and both experiments individually tuned on a) the maximum tree depth and b) the number of base learners

For each experiment, we chose one hyperparameter and varied its values over a predefined range. The results were then graphed the change in score across these values. As a default configuration, 50 was the number of trees, and 6 was the trees' maximum depth, whilst 0.1 was the fixed learning rate.

Additionally, we used the Gini index as the impurity measure in the splitting criteria of each tree's node and the exponential loss function to grow the next estimator in line with the formulation given in Equation (5.3.7).

Cross validation procedures ran for 3616.5 and 2776.3 seconds respectively and they both excluded attributes of the user's home state and their current endemic condition at T_1.

The experiment's score outcomes across the different hyperparameter values were graphed in Figures 5.3.1 and 5.3.2 where these compare the cross-validated and training set scores. Here we have that for each hyperparameter value, the $1 - Accuracy$ score is graphed as a function of it. Thus having a lower score in the graphic means a better model was reached.

Recall how these outcomes compare to Figure 4.2.1 and Figure 5.2.2, in which a tree's training error always improves as a result of an increase in the tree's depth.

Here, we see that the estimator is prone to overfitting when using a high *maxdepth*

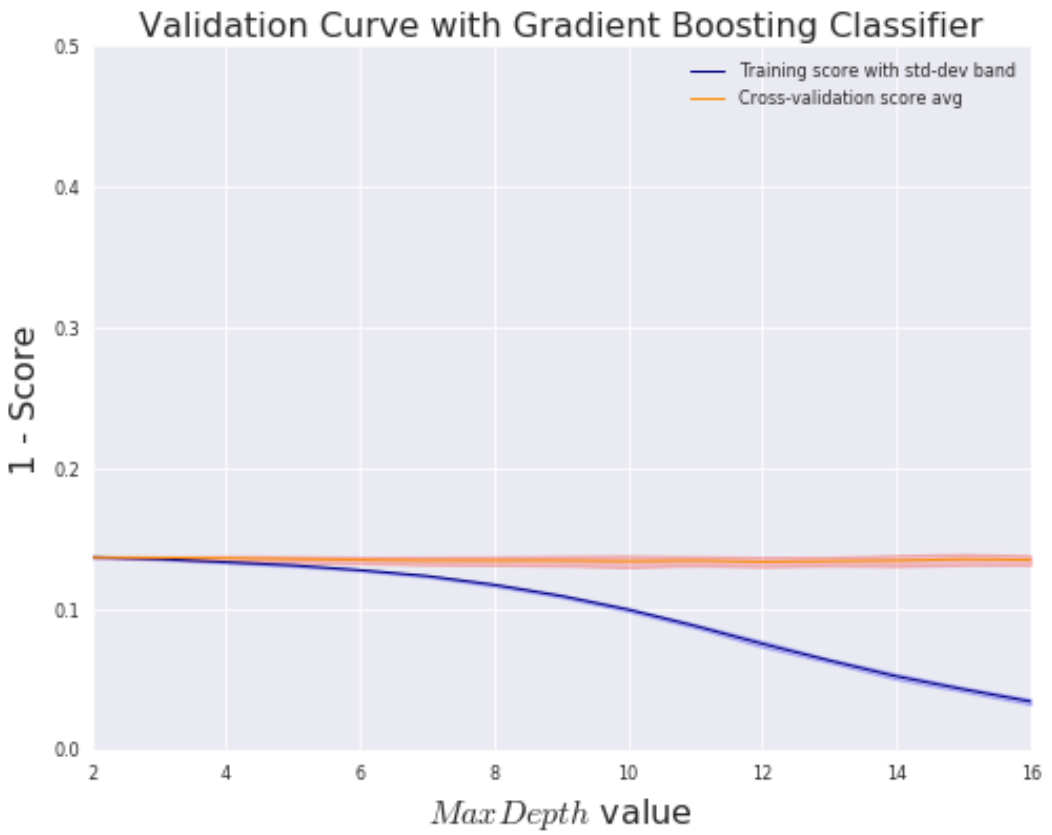


Figure 5.3.1 Validation curve for the tree-depth hyperparameter of the Gradient Boosting Tree learner. The mean CV scores for the $1 - Accuracy$ score is shown for a CV run experiment on Problem 2

value because the training deviates faster from the CV error when the depth is higher. This acceleration is clear from the figure when we see that the CV score is invariant while the training error decreases. At the same time the model differs from the Random Forest classifier in that the trees are not uncorrelated, because the t-th tree improves on the error of the model at t. This is an effective difference in the models.

For the second experiment, the overfit is smaller, given that the relative difference in the CV score and the train score is smaller.

It is interesting to note that, for both cases, there is a very small gain in the CV error when varying the hyperparameters, as evidenced by the range of the scores. Yet the decrease in CV score is slightly better for the second experiment and this situation is different to the case of the Random Forest learner, where the scores had a stronger improvement in the cross validated score. The model seems to better fit the data compared to the Random Forest, since lower error rates are achieved, and it converges faster to a stable CV error where adding more trees or depth doesn't improve on the score.

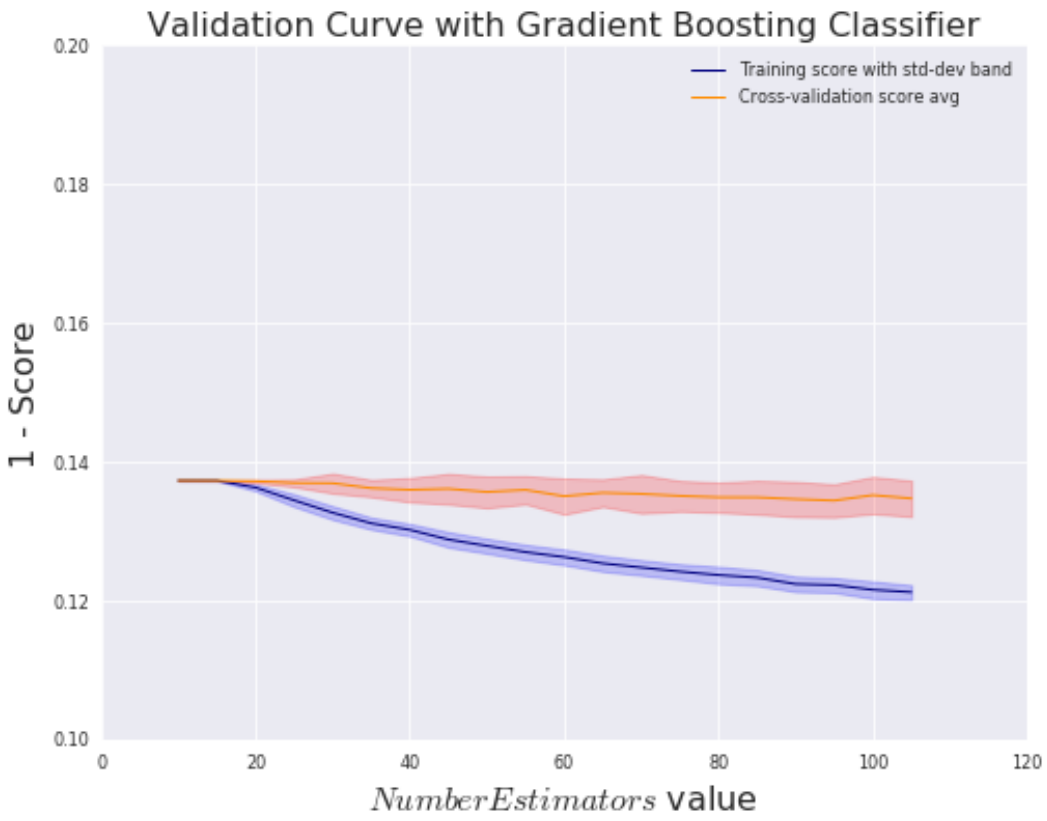


Figure 5.3.2 Validation curve for the number of trees hyperparameter of the Gradient Boosting Tree learner. The mean CV scores for the $1 - Accuracy$ score is shown for a CV run experiment on Problem 2

At their optimal configuration, both setups reached CV score averages of at least 86.5%, with a maximum of 86.9% for the learner with trees of 12 levels. However, the optimal number of estimators for the CV and training sets were different, where 105 was the best configuration for the training score and 95 was the best average for the CV set. Wit this, the relative difference in these best scores was of 1.5%. It is not surprising that the relative difference of the best CV and training scores, for the maximum tree depth experiment was much higher with 11.5%, given this hyperparameter's influence to the model's complexity.

The outcomes here expose the drawbacks of this model which is more prone to overfitting and takes longer to fit when compared to the Random Forest algorithm. Yet it gives better overall performance on the CV error, when the model is correctly calibrated.

5.3.5 Experiments

In this subsection, we explore the task described in Problem 4 using Gradient Boosting classifiers and we compare the outcomes with the results outlined in Section 5.2.5. This task is set to predict those users that migrated from the endemic region, but only allowing users which are currently not endemic as the base set which implies that we are excluding all users that are currently endemic.

We set two experiments similar to the ones before, where we run two procedures: one with all available features and another without the top target-correlated ones. For the third experiment, we trained **only** with the best features extracted in the Random Forest's second experiment. These were selected from the best-fit learner used from that case and, with this third variant, we expected to see how good of a performance can the gradient boosting classifier achieve, using only the top 30 features, as selected by best-fit random forest learner for the previous experiments. Doing this has a hypothetical trade-off between accuracy and procedure run-time, where we can gain speed in our fitting process by compromising some of the predictive power of the algorithm.

Once again, in these experiments we performed a cross validation procedure over the available hyperparameters of this classifier and in our CV procedure, we will be searching the best learner over a grid combination of all parameters. For the most part, these hyperparameters are the same as for the Random Forests learner, yet the "Boosting" model has some additional ones, which we outline in the following list:

- The gradient descent's defined step size — $[step\_size]$
- Minimum threshold for the sum of samples' Hessians at a given node. If a node's sum is smaller than this threshold then the fitting will stop — $[min\_leaf\_weight]$

Using the same setup as before, we recorded the outcome of the three experiments' cross validation procedures. The results were tabulated in terms of different metrics' scores, runtime, best resulting hyperparameters values and the top ten best features as selected by the best models. A summary of the metrics and best hyperparameter values is given in Table 5.3.1.

The first notable fact from the results is how fast the last experiment is, compared to the first two. Training the model was much faster, with almost 3 minutes to complete the full procedure whilst the others had at minimum 20 minutes to do so. We see that the information from the top 30 features of the RF models was efficiently used in this boosting experiment and that this feature reduction considerably sped up the learner's runtime. This improvement should not be surprising then, as we have limited the amount of available features to use, thus decreasing the necessary computer resources to compute the fit.

The second interesting fact from the outcome is that the model seems to be more overfit, when compared to Random Forest experiments. This is specially significant in the first two experiments, where very complex models were chosen from the procedure, with a depth of 15 levels and at least 250 trees for both. However, this is

Table 5.3.1
Table of best results comparing three 10-fold full grid cross validation procedures with a Gradient Boosting Classifier for Problem 4. All relevant scores, best hyperparameter values and run-time are shown for both experiments.

Result	Experiment 1	Experiment 2	Experiment 3
Running time (s)	2993	1972	204
CV $F1$ score	0.314	0.319	0.354
CV *Accuracy* score	0.889	0.843	0.8327
CV *Precision* score	0.209	0.217	0.249
CV *Recall* score	0.629	0.601	0.607
CV *ROCAUC* score	0.884	0.880	0.831
CV $Log - loss$ score	0.524	0.337	0.399
balanced	False	False	False
max_depth	15	15	12
num_trees	250	300	120
min_leaf_weight	2	5	10
min_loss	10	1	0.01
$step\_size$	0.01	0.1	1
$max\_features$ (%)	80	75	10

not the last model's case though, where a simpler model resulted in a depth of only two levels and 120 trees grown.

The same applies for the $max\_features$ and min_leaf_weight hyperparameters, for which lower complexity values resulted form the last CV procedure. There is difficulty though in establishing which of them had the most weight in an underweight model.

When analyzing the $F1$ score of the experiments, we find ourselves with a scenario which is not that different. We see that the last procedure improved on the first two experiments in more than 10%, using this score and in an almost 20% increase in the precision score. This suggests that the first two models were overfitting the training set and by using the top random forest features, we strongly improved on the $F1$ performance. Finally, when compared to the Random Forest's experiments, we see that the boosting models performed near the top $F1$ value of 37% achieved in the Random Forest experiments. From all model $F1$ scores, the lowest and highest output scores were 31.4% and 35.4% respectively. This means that, at most, the worst model had a performance for this score which was 17% lower.

Still, for that case, the top Random Forest's $F1$ score was attained in the first experiment, where we made use of the highly target-correlated features. This is

then a remarkable performance for boosting learners which were not trained on this attributes and still achieved similar or higher scores.

We also notice that the third model is slightly worse in the *ROCAUC* score, when compared to the first two. Here, the first model reached a score of almost 90%, whereas the third model had a poorer performance of 83%. Still, the lowest value for all experiments is more than satisfactory to predict the which of the currently non-endemic users have migrated out of the endemic region.

All in all, with these results and due to the difference in *ROCAUC* vs. $F1$ scores, there is no clear indication as to which configuration is better across all experiments.

Below we introduce the learner's selection of best features for each experiment. Note that the last selection will have a very similar configuration to the top Random Forest features, since it selects 10 features of the available 30 used to train the model. This outcome is presented in Table 5.3.2, where the top 10 features for each experiment are shown.

Table 5.3.2
A representation of the top features that resulted from the Gradient Boosting procedure on two experiments of Problem 4.

	Top Features		
First Experiment	Call Count Antenna_0	Mobility Diameter	Mobility Diameter Weeknight
	Call Count Weeknight Antenna_0	Call Count Antenna_1	Call Count Antenna_2
	TimeWeekDay In Month_08	Call Count Weeknight Antenna_2	TimeWeekDay Out Month_09
		TimeWeekDay Out Month_12	
Second Experiment	Mobility Diameter	Mobility Diameter Weeknight	Call Count Antenna_0
	Call Count Weeknight Antenna_0	Call Count Antenna_0	TimeWeekDay In Month_09
	TimeWeekDay Out Month_09	TimeWeekDay Out Month_12	TimeWeekDay Out Vul Month_08
		TimeWeekDay In Month_10	
Third Experiment	TimeWeekDay In Vul Month_09	CallsWeekDay In Month_09	TimeWeekDay Out Vul Month_12
	Mobility Diameter Weeknight	Call Count Weeknight Antenna_0	CallsWeekDay Out Month_09
	CallsWeekDay In Vul Month_08	TimeWeekDay In Vul Month_08	TimeWeekDay Out Month_08
		TimeWeekDay In Month_12	

The top features list shows us the importance of mobility diameter in both cases. We see that both weeknight and all-week user's mobility, are deemed relevant to detect their past endemic condition. This outcome confirms what was seen in Table 5.2.2 where these attributes were, in general, highlighted.

As expected, in all cases we see that there are a number of features logging vulnerable interactions of users living in non-endemic regions. The same can be said about the call volume features, which are selected as important, both in the number and times of calls. Unfortunately, we cannot determine their relationship to the target variable in the sense that it might equally affects in both a positive and negative way.

These results are in line with the assumptions we made when we constructed the

risk maps in Section 2.3. The direction and duration of calls, and the number of occurrences of these are all relevant predictors in the problem of tagging users that moved out of the endemic region.

Once again, we see there are more features of earlier month interactions, such as August and September. At the same time, the only other month appearing is December. This could be indicative of familiar relations as it is a month which coincides with festivities and family reunions.

The analysis here confirms the relevance of CDR information in tagging long-term migrations. Experimental results show satisfactory scores across different metrics and for a highly imbalanced class. Also, top features are in line with the assumptions on which the risk maps were constructed.

5.4 Benchmark Classifier: Naive Bayes

The Naive Bayes model encompasses a group of simple and computationally efficient algorithms which are built with a strong statistical assumption of independence among the features. Even though this belief is in practice wrong, the model still achieves acceptable classification rates for some problems. In addition, it does not suffer in problems of high-dimensionality, where $p >> n$ i.e. there are more attributes than samples in the data.

The method is presented in this work for the purpose of providing a benchmark algorithm. In practice it is expected that the classification rate achieved by this model serves as a baseline for other, more complex, learners.

The benefits of the algorithm include having linear complexity in the number of features and samples, $O(d + n)$, so it can be easily extended to *large* problem implementations. Furthermore, its maximum-likelihood estimation of the parameters has a closed form solution which is faster to compute over other iterative methods such as techniques using gradient descent or other similar iterative optimization routines.

Let $x = (x_1, \ldots, x_p)$ be any given data sample and C_k be one of K possible output classes of a classification problem. We take $p(C_k \mid x)$ to be the class posterior probability of this class given the sample.

In Chapter 3 we have used

$$p(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)} \tag{5.4.1}$$

and argued that if our data is given, then our model can only improve the posterior probability by optimizing $P(x|C_k)P(C_k)$ which is just the joint probability of the sample and the class.

Here we can approximate the posterior as

$$P(C_k \mid x) \propto p(C_k) * \prod_{j=1}^{p} P(x^j \mid \bigcap_{l=j+1}^{p} x^k \cap C_k) \tag{5.4.2}$$

We now impose a strong independence assumption among features, given the target class, to let the conditional probability factors become the probability of each feature.

This yields a posterior probability which depends only on the prior probability and on the individual likelihood of each feature.

$$P(C_k \mid x) \approx p(C_k) * \prod_{j=1}^{p} P(px^j \cap C_k) \tag{5.4.3}$$

As we have said before, the parameters of the model can only reweigh the likelihood factors, so if we look to maximize the posterior probability, our final estimate of the posterior will take the following form.

$$P(C_k \mid x) \approx \frac{1}{Z} p(C_k) * \prod_{j=1}^{p} P(x^j | C_k) \tag{5.4.4}$$

where in the equation $Z = p(x)$ is a scaling factor and is calculated from the dataset.

In practice, the model will stem into different algorithms where each variant will have a different probabilistic assumption on the likelihoods $p(x_j \mid C_k)$ of the model and on the priors $p(C_k)$. It is common to choose among these by using non-parametric density estimations from the training data or by setting parametric assumptions on the data distribution, such as being distributed from an exponential distribution family such as a Gaussian, Bernoulli or Multinomial distribution.

Different choices will certainly lead to different cross validation scores among problems. Altogether, these choices can be treated as part our model's hyperparameters and the best one can be selected with our CV procedure.

Finally, the output class for a given sample will be given by taking the class k' which maximizes the probability $P(C_{k'} \mid x)$.

5.4.1 Experiments

Here we present a very brief outline of this algorithm's performance on our general problem. Without going in to depth on the specifics of the experiments, we ran the classifier in a cross validation procedure against all of the four tasks outlined in Section 3.2, and using the same *Accuracy*, $F1$ and $ROCAUC$ classifier scores as in other method's experiment runs. We took advantage of this method's fast training runtime to rapidly evaluate all tasks on this algorithm. The results were composed

into Table 5.4.1 and summarize two aspects of the method: its lower performance when compared to other previous methods, and the algorithm's fast run-time. Given that this algorithm was introduced for bench marking purposes, we will not expand in this section any further results for this learner as these will be made in Chapter 6, along with specific comparisons to other learner's results.

Table 5.4.1
Table comparing cross validated results of best fit learners, comparing all 8-fold cross validation procedures on a Naive Bayes classifier. Here a Multinomial prior distribution was used and the relevant classification scores are shown

Measure	Problem 1	Problem 2	Problem 3	Problem 4
Running time (s)	125	121	105	85
CV *Accuracy* score	0.842	0.649	0.658	0.852
CV $F1$ score	0.753	0.313	0.457	0.626
CV $ROCAUC$ score	0.827	0.617	0.635	0.768

The idea of this chapter is to present a combined overview of the work done and the results encountered in Chapters 2 to 5. All in all, we intend to showcase what relevant information were found in the preceding chapters. Most importantly we found in this work that there is evidence to confirm the hypothesis which stipulates that cell phone usage logs are rich in social interactions and, in turn, we show that these interactions are informative enough to predict long-term migrations and human mobility.

Chapter 6

Summary of Results

In Chapter 2 we presented the CDR dataset and how it contained information relevant to the problem of long-term human migrations. With a minuscule sample of the CDRs we showed that the dataset is rich in social interactions between users and that the data contained dynamic georeferenced information at the user level. This gave us an idea of how this type of data could be leveraged to the general problem. We then framed the problem of long-term human migrations in this context and gave an idea of their importance, in relation to the epidemic nature of the disease.

We noted that the dataset, as it was presented, posed an unbalanced classification problem as most TelCo users did not show any migration pattern to or from the endemic region, in the time frame of analysis. This also implied that we had a very correlated problem, where most users did not change their past and present endemic conditions. This information, combined with the local calling nature, meant that we had correlations with the target feature. This was noted by seeing that users living in the endemic region will most probably have lived there in the past as well.

In later sections we also noted that this strong past-to-present correlation in user's homes was relevant to the prediction problems. This is because for tree based learners, knowing where the user lived in the present, was a strong indicator of where the user's past residence. This information provided the algorithms with features that were highly informative. Yet we decided to exclude this attribute for the problem of predicting a user's past home endemic condition because we wanted to evaluate the information present in other, less trivial, features.

More so, we found other variables having mild interactions with the condition of being a past endemic user. As expected, people with high mobile interactions with the endemic region, in the form of call duration and call counts, also resulted in attributes which were related to users having past endemic homes. This is most probably due to the locality effect in calling patterns, where most interactions between TelCo users occur in local regions. Thus users living in endemic regions would call other users nearby.

In this stage we also set to explore the data with feature transformations and visualizations to further discover insights relevant tot he problem of long-term human migrations. As a starting point we transformed the user-level data into a dataset at the antenna level. This step aggregated information of call patterns to and from the endemic region. It allowed us to present geolocalized visualizations of these social interactions to the vulnerable areas. As an intermediate step in this process we had to introduce the definitions that we later used for the rest of this work. We defined what a vulnerable interaction meant and what is a user's home antenna. To finish with the data exploration and produce the heatmaps, we aggregated the data at the antenna level. In these visualizations, call interactions were georeferenced and antennas were colored according to their level of vulnerable interactions..

The heatmap visualizations exposed a 'temperature' descent from the core regions outwards. As expected, the heat was noted to be concentrated in the ecoregion. We also found out that the level of vulnerable interactions per antenna gradually descends as we move further away. We say this is expected behavior because it is consistent with findings in the literature in which most calls are done to other local antennas. Given this period and TelCo of analysis, we saw that ninety percent of the users limited usage to at most four TelCo antennas.

We also discovered some unexpected findings that were highlighted by the risk maps. Interactions from non-endemic antennas with those in the endemic region were seen to be non-homogeneous in some areas. As an example, Figure 2.3.3 outlines various antennas with higher vulnerability. We suspect this non-uniformity in the vulnerable interactions can help detect communities with higher probability of disease prevalence.

Health experts agree that these anomalies can be a great starting point to start as communities atypical in their neighboring region carry potential of being a latent endemic foci. These antennas stood out for their strong communication ties with the regions studied, showed significantly higher links of vulnerable communication.

In the images presented, the differences in the vulnerable interactions were clear. When talking to the "Mundo Sano Foundation" researchers participating in this project, they pointed to the fact that the detection of these antennas through the visualizations was of great value to their goals.

At the national level, the results evidenced by the maps were coherent with the expert's knowledge of the endemic zone's migration patterns.

In Chapters 3 and 4 we gave a practical introduction to Machine Learning in general, and of Supervised Classification problems in particular. This helped us set our Chagas problem inside a systematic process to analyze long-term migrations with the CDR dataset.

To do this, we broke the problem of long-term human migrations down in four tasks which were later analyzed through different classifiers. In these two chapters, we introduced the Machine Learning theory necessary to structure our problem, along with its methods to solve them. These models were some of the most common techniques found in the literature for the task we were trying to solve.

To begin we considered the Logistic Regression Classifier along with the Log-

loss metric to evaluate model performance. Taking to our advantage the model's more tractable formulation, we showed how model regularization fits inside the Machine Learning frame. This concept was introduced along with the notion of model hyperparameters. Both were relevant to subsequent sections and in particular to Figure 3.5.1. There we showed that there is importance in model regularization by fitting a Logistic Regression classifier and comparing how its Log-loss varies across different regularization values. This affected both the training and test set performance in a similar way.

At the same time we introduced other relevant concepts in the Machine Learning and we illustrated them with examples from the long-term human migration problems. As is common in the literature, in Figure 4.2.1 we saw that increasing a decision tree's complexity leads to a clear case of overfitting the training data when trying to approximate the generalization error.

The same procedure was used to illustrate the concept of "Cross Validation" in Figure 4.3.1. We found that when making predictions on which users were endemic in the past, the test score was inside a one standard deviation band from the CV score, across a varying hyperparameter value for the Logistic Regression classifier. These findings are consistent with lots of other experiments found in the literature where by cross validating the search of optimal hyperparameters, we can have a fair estimation of the test error.

Towards the end of Chapter 4 we performed a deeper experimentation with this same problem and also on unidirectional users: those that moved out of the endemic region from the past to the present time frame. Here we used the same classifier as before but on the whole dataset in what defined our systematic approach of analyzing classifiers performance.

The runtime for this experiment had a lower bound of one and a half hours. This was for all of the cross-validated fits and for the whole set. Also since we limited the number of iterations to at most one hundred steps, this runtime is actually smaller than the real optimization time for this algorithm, Another issue found with this algorithm was its extremely poor $F1$ performance on the prediction of these users that migrated out of the endemic region. We found that it had a test score lower than 17%. As we will see later in other statistical learners, we had that the overall precision of the classifier was very poor due to not being able to accurately predict positive samples. The algorithm was very inefficient in this aspect of overestimating users that migrated out of the endemic region, and this misclassification had a direct impact in the observed $F1$. However, this low score was not repeated in the rest of the metrics we evaluated. For this same problem, both the best *Recall* and *Accuracy* test and CV rates were over 61%.

Another relevant observation was that best-fit C values were not consistent along all fits, because these resulting values were seen to differ by orders of magnitude. Solutions for C ranged from to 10^{-3} to 10^{-1} on different fits. With this we can still positively affirm that the best-fit models for this experiment were those which penalized the loss function with strong regularization terms.

In contrast, when using this classifier to predict past endemic users residents, we

found that we could get a $ROCAUC$ score of 76%. Moreover, this score is achieved when cross validating solely on the $l1$ regularization parameter. A similar test score of 74.4% was reached when optimizing for the $l2$ hyperparameter of regularization. Each were optimized separately in order to exemplify how the overall $ROCAUC$ varied along different regularization thresholds.

This outcome of better $ROCAUC$ scores, when compared with the experiments on migrations out of the endemic region, underscore the difference among the problems' difficulty. If we recall the highly unbalanced class relations for Problem 2, we confirm our hypothesis that these predictions characterize a problem which is very difficult to precisely attack. This problem sought to predicting those users that had migrated in one direction, out of the endemic region from the past to the present time.

Overall, both procedures had best fits on the more regularized C values. A similar result can be found for regularization models and on procedures where hyperparameters were cross validated. Both optimizations improved the model's predictive power in varying degrees.

The previous results outline a complete systematical Machine Learning approach and tool set used to evaluate the long-term migrations prediction problem. In this way, we provide a methodical way to analyze classifiers performing on this problem. With this we tackled the task with the introduction of new algorithms in the following chapters.

In Chapter 5 we introduced four new classifiers along with their properties and characteristics. Three of these, Sections 5.1 to 5.3, were tree based methods whilst the last, Section 5.4, was a Naive Bayes Classifier used for benchmarking purposes. For each, we presented an introductory review and pointed the reader to further literature references where applicable.

All but the Decision Tree learners were put through a number of experiments to evaluate how they performed across all prediction problems. For each model, we tried to draw the greatest prediction performance from the features extracted from the dataset. However, Random Forest and Gradient Boosting models were evaluated in further detail whilst Decision Trees were not systematically analyzed. This base learner still served as common ground from which the following, more complex, learners were built upon.

As a start, we showed through a validation curve that, for the problem of users moving out of the endemic regions, the learners would become overfit if we let the trees grow in depth. We had that the cross-validated scores and the test scores diverged rapidly after a tree-depth of 7 and this is plotted in Figure 4.2.1. The wide gap in the scores resulted in a difference larger than 30%.

In conclusion, the overfitting effect in this learner was visible in the experiments where deep trees and complex configurations underpowered the resulting model's generalization error. This is not surprising as Decision Trees are known to become highly complex after a certain level of tree growth.

We also fit two Decision Tree instances for the same problem to evaluate the model with an *Accuracy* score. The first of two fits run over a smaller hyperparameter space and, in particular, this configuration did not consider a re-balancing of the

samples when computing the value of the loss function over the whole set. Then in the second fit we performed sample balancing in the loss function to quantify the algorithm's outcome with and without this configuration.

To aid our understanding of this model's outcome we also decided to create visualizations for both fitted trees. Each tree is plotted in a graph which shows up to five levels of splits. With this we showed, for each node, the attribute used in that splitting decision and the actual split value.

These plots provided some insights into what features were initially being used by the algorithm to separate samples, according to their split index performance. For instance, in both fits the root split was based on the user's mobility diameter, with split values near to 47.5 kilometers in both cases. This means that a mobility diameter of 47.5 was the best split decision the algorithm could take in order to improve the Gini index of the sub groups. Recall here that the mobility diameter quantifies the user's area of influence given by their mobile connections.

The other significant features, selected at splits near to the tree's root level, was the user's home antenna call count. Both the volume of calls during the whole week and usage specifically during weeknights time was important to the node-splitting algorithm.

For both trees, the split values for these two variables were relatively similar, where in the volume of weeknight calls using the home antenna was of 10.5, whilst for total volume of home antenna calls, at any given time, the values were 139 and 186 calls for each case. These split values were characterizing segments of heavy users, that is users that are higher than the median values for the home antenna usage variables.

Here we must note however, that both best-fit trees result in a low *Accuracy* scores of 43.2%,and 57.9% respectively. Under these circumstances we find that these top features selected when building the trees could not be very informative of the overall problem in users that migrated out of the endemic region. This means that, due to the low performance of these learners, the interpretations we build from this output can be based on the wrong assumptions.

Interestingly enough, in these experiments we found that for this learner, re-balancing the set helped us increase the learner's score by a great amount. We hypothesize that the hyperparameter is a valuable in helping face problems where target classes are heavily unbalanced.

The rest of the Chapter 5 continues with the introduction of other learners. All of them are presented with a brief explanation of their formulation and their distinctive characteristics.

For each of these, we successively experiment the same difficult predictive task. Where we look to tag users that migrated out of the endemic region, from time periods T_0 to T_1. In this way, we set a common ground to compare and evaluate the algorithms. As we will see later in Table 6.1.1, this task, Problem 2, happened to be the hardest one, as it resulted in the lowest metric scores for all classifiers.

When tested over with the Random Forest classifier, we saw better *Accuracy* scores than with the Decision Learner over a 10-fold cross-validation procedure. This

model improved the mean cross validated score to reach a high score of 81%, for trees of depth bigger than 15. Note that in this experiment we reweighed samples in the loss function by default, following the improved results we obtained by doing this for the Decision Learner classifier.

We also saw that there was barely no change in the average CV score when we increased the number of trees used in the ensemble. Having an ensemble of 20 was enough to capture most of the predictive information we needed to correctly tag the unidirectional migrants of the endemic region. From this it seems that adding more trees would not improve the generalization error in great values. Still, the optimal number of trees was found to be around 130 for the CV procedure, and this gives scores which are only 3% better than the ones for an ensemble of 20 trees.

In comparison to the previous model, the Forest also overfit the data when we deepened the tree depth. However it did so at higher depths, where the first deviation across the mean error rates for both the CV and training sets was at a depth of 9, compared to the previous level of 7.

At the same time we had that for this model, the biggest differences between the average training and CV errors, across hyperparameter values, was at most of 10%. This result is expected in this learner which is designed to lower the base models' variance. The downside in this learner was that the optimizations run for longer, taking at least 20 minutes for both cases.

These results for the Random Forest model were also compared to the Gradient Boosting model. The same two hyperparameters were validated across a range of values, and for each we looked at how the Training and CV scores varied. Similarly, we found that this model's score won't be greatly affected by the number of trees used in the ensemble, and that most of its predictive performance can be captured by using only 20 trees. Still, we note that the best-fit CV value was of 95.

For this learner, we also found that it will rapidly overfit the data once the base learners get a maximum depth of 6. What was interesting here was that the mean CV *Accuracy* constantly hovered around 86%, whilst the score for the training gradually improved as the tree-depth grew, up to a score of 96%. This algorithm rapidly overfits users that migrated out of the endemic region, which is similar to what we saw with the Decision Tree Learner. Still, and compared to the Forest learner, we get a better CV error, at the expense of a runtime which at least 50% slower.

In a second stage of experiments, we performed extensive testing both on the Random Forest and Gradient Tree Boosting learners for the problem of tagging the same users that migrated out of the endemic region, but only considering as user-base, those users which are currently living outside of the endemic region.

For both, we ran a full CV procedure in two feature versions, one with all the features and one in which target-correlated features were removed.

As a general statement we can say that the Random Forests experiments have lower runtimes because take better advantage over the algorithm's parallelizability. Also, we can not affirm that one algorithm is strictly better than the other. Both of them exceed each other across different metrics. It seems overall that the Random Forest is better at correctly classifying positive instances, while the Booster is slightly

better over the *Recall*. These asymmetrical performances across *Precision* and *Recall* scores resulted in that no algorithm outperformed the other when evaluated using the $F1$ score. Due to the low *Precision* rates, both achieved low $F1$ scores with similar outcomes, where the maximum of these values was 37%.

On other scores such as *Accuracy* and *ROCAUC* the experiments showed high classifier rates, where all of these ranged between percentages of 83% and 90%. These high scores show there is enough predictive capacity in using these algorithms for this problem of tagging migrants from the endemic region.

On the best-fit hyperparameter values, we see that both algorithm's values agree in that a large number of base learners are needed to perform better. This is evidenced by the fact that, for all experiments, the optimal value was higher than 100 trees, up to a maximum of 300 base learners. A similar result is seen with the tree's depth, where all best-fit values were, at least, over a depth of 12.

Both the Random Forest and Gradient Boosting models played an important role in all of the prediction tasks we made. The high scores that result in some of the metrics indicate that there is predictive information for the problems of long-term human migrations. Another advantage of these two algorithms, is that they provide heuristics on the best-features that we can use from the dataset.

From the process of calculating those best-features, we can say that for this prediction task of finding, of those currently non-endemic users, which are the migrants that moved out of the endemic region, the feature selection heuristics of ensemble algorithms agreed upon a group of features. These were highlighted, for a number of times, over the rest of the features and they provide insights into what CDR variables provided most predictive value. Note though that this heuristic does not provide any information into how the feature favors the prediction, be it in a positive or negative way. Below we can find a descriptive collection of those features for these learners, where we have broken down all of this information into feature categories:

- Calls made in older months and in December: Interactions of different type, that occurred closer to the split month between both periods T_0 and T_1 (July), were also distinguished. The model frequently used interactions from the months of August and September, as well as December. We conjecture that this last result can be due to the fact that December is a month where user's increase their mobile activity with their families.

- Vulnerable calling patterns: As we first suspected in Chapter 2, the vulnerability in the mobile interaction between users was relevant. The duration and volume of and volume of calls, at different time periods, were highlighted for their importance and in a number of occassions the interactions to and from vulnerable users was relevant to the algorithm. We make the reminder here that these measurements were the basis of our construction of the heatmaps in Chapter 2, where for a given visualization we were showing how a month worth of communications was aggregated at the antenna level. These plots aggregated calling patterns for all users with vulnerable calls.

- Mobility size: Finally, different measures of the distance in human mobility were determined to be important by the model. Both mobility diameter attributes were highlighted, where in one considered the mobility of users during weekdays and for the other, only specific mobility during weeknights. General mobility diameters and weekday or weeknight specific mobility were relevant. The importance of these features were highlighted on a number of occasions by the ensemble learners. We can also note that in Section 5.1.4 the Decision Tree Learners selected this features for the higher nodes.

These factors identify strong predictors in long-term human mobility and provide additional insights into our original hypothesis of which predictors were most valuable for the task. Understanding these factors can further provide information into the problem of Chagasic disease spread in the long run.

As a final remark that we must add here, the previous best-features list carries some bias introduced by the model's feature-target correlations. A similar argument can be made for the best-features calculated from the Gradient Boosting experiment that used only a filtered set of features, previously selected by the Random Forest experiment's best-features result. There is a strong feature-target correlation and there is a leakage of target information within this experiment.

To look at a more thorough examination of these results and how they were selected, the reader can refer to Tables 5.2.2 and 5.3.2.

Given all of the previous findings we collected, we can say that there is no absolute single method that can be single-handedly applied to all problems and evaluation performances. However, our results outline that the Gradient Boosting methods were, in general, top performing for all tasks and classification metrics, except for the $F1$ score. Surprisingly enough, for some problems the Naive Bayes was outperforming all learners in the $F1$ metric. The algorithm was more conservative in its positive predictions and this difference saw a trade off in its performance across other metrics.

However we can say that results show that the performance of Gradient Boosting models is best than other models such as Logistic Regression or Naive Bayes algorithms for most tasks, and slightly better than the Random Forests. These models can find complex interactions in the data whilst handling enough care not to lose generalization power.

6.1 Master Table of Results

Table 6.1.1 shows a compendium of the results obtained from the CDR dataset, using all classification models. For each problem and model, we found the best performing learner with a cross validation procedure which searched over an extensive grid of hyperparameters. In all cases, the hyperparameter configuration with highest cross-validated $ROCAUC$ was selected as the *best-fit*. The table shows the best-fit classifier's performance across three scores on the test set $\mathcal{T}_f$: *Accuracy*, $F1$ and $ROCAUC$.

Table 6.1.1
Master results table comparing results for all of the classifiers run in this work. For each task and classifier, we show the its *Accuracy*, *ROCAUC* and $F1$ test-set scores, along with the runtimes of a full cross validation procedures on the learner.

	Problem 1			
	Accuracy	*ROCAUC*	$F1$	*Runtime*(m)
Logistic Regression	0.893	0.857	0.9	96
Random Forest	0.878	0.857	0.9	33
Gradient Tree Boosting	0.974	0.978	0.952	41
Naive Bayes	0.84	0.82	0.75	2

	Problem 2			
	Accuracy	*ROCAUC*	$F1$	*Runtime*(m)
Logistic Regression	0.714	0.726	0.248	119
Random Forest	0.79	0.776	0.278	45
Gradient Tree Boosting	0.838	0.819	0.291	54
Naive Bayes	0.64	0.61	0.31	2

	Problem 3			
	Accuracy	*ROCAUC*	$F1$	*Runtime*(m)
Logistic Regression	0.705	0.754	0.307	115
Random Forest	0.792	0.845	0.346	21
Gradient Tree Boosting	0.811	0.855	0.359	33
Naive Bayes	0.65	0.63	0.45	1

	Problem 4			
	Accuracy	*ROCAUC*	$F1$	*Runtime*(m)
Logistic Regression	0.883	0.85	0.331	106
Random Forest	0.898	0.853	0.393	19
Gradient Tree Boosting	0.885	0.873	0.384	47
Naive Bayes	0.85	0.76	0.62	1

Table 6.1.1 exposes all combinations in model performance and runtime across the problems studied for this work. At first glance there are some notable results exposed in this table.

6.1.1 Runtime Performance

It is clear that the Naive Bayes algorithms outperforms all other models in runtime and, in some cases, this improvement is in orders of magnitude. The algorithm's worst performance for all problems is of two minutes whilst the best runtime for any other classifier is for the Random Forest, with 19 minutes of runtime for the problem of labeling users that traveled out of the endemic region, yet only for a subset of the users base: namely those users which are currently settled outside of the endemic region.

The Naive Bayes algorithm's runtime shows a strong difference with the Logistic Regression which was the slowest method overall, with the fastest runtime for all problems being greater than an hour and a half. Having the Logistic Regression as the slowest performing algorithm may be surprising to the reader. Consider that in the runtimes measure timings on the cross validation procedures and that the logistic classifier has a short list of hyperparameters combinations to optimize, when compared to tree based methods. Still, we suspect that the time bottleneck in this algorithm comes in the slow optimization procedure where, for every iteration, the whole dataset needs to be reused to build the direction of steepest descent.

Another result here is that the Gradient Boosting algorithm had a higher runtime than the Random Forest and, in certain cases, the results showed almost the double time taken to fit the model. Even though the boosting algorithms implement several heuristics to speed up the algorithms with local gradient approximations, we also have that when building the Random Forest ensemble, the weak learners need not be dependent, so the optimization routine can create them in parallel computations. Thus this algorithm benefits strongly from parallelization procedures. On the contrary, the boosting procedure builds successive trees that improve on the most current model. This dependency limits the optimization routine parallelization opportunities.

At this point it is worth mentioning that the runtimes resulted from experiments that were not performed under controlled computing environments. For example, we cannot assure that all cross-validation procedures were done in settings with equal available computational resources. Even though results seemed consistent across all problems, we know that these results shouldn't be considered as benchmarks.

6.1.2 Low F1 Scores

Overall, we can say that, excepting the problem of predicting users that had lived in the endemic region in the past, no model had satisfactory $F1$ scores for any of the problems. The top two $F1$ scores for Problems 2 to 4 and for any best-fit learners, were 0.62 and 0.31. Both these scores were attained by the Naive Bayes algorithm for the problems of currently non-endemic users that were living in the endemic region

in the past, and for the problem of tagging users that had migrated in any direction.

Interestingly enough, these poor scores for the Naive Bayes algorithm were, in comparison, much higher than the ones reached by any other classifier, when compared in relative terms. All of the other classifiers had extremely poor $F1$ performances in all of Problems 2 to 4. The best $F1$ score for the rest of the classifiers, across these problems, was 0.393. This was attained by the Random Forest learner when tagging people that had migrated out of the endemic region, from a user-base of those who are currently non-endemic.

We mentioned this situation earlier for the experiments specifically performed on this same Problem 4. We saw that all of the learner's output a very low *Precision* score and this impacted the overall $F1$ score.

The overconfidence shown by the models derived a high misclassification rate for the samples in the positive class. This error was so strong, that it drove the $F1$ scores to low values, even when they output acceptable *Recall* scores. Still, the trade-off in the precision of their positive predictions was very strong.

We mentioned that this situation might appear in highly unbalanced classification problems, such as the ones we are working with here. The overconfidence effect in learners is affected by the unbalanced target class. With this we can say that precisely tagging migrant users is a very difficult prediction problem, with the data available.

It was surprising to discover that a very modest and fast model could, under this metric, outperform all of the other more complex ones. We suspect that the Naive Bayes's formulation helps limit the overconfidence of the final model, allowing it to still reach acceptable *Recall* scores. In all, we saw that this was the best model for all problems of migrant users, and when evaluated on the $F1$ score.

In contrast, the problem of predicting users who, in the past, had homes in the endemic region, by only taking into account their information from the present, is a task that does not present this ill-conditioned class imbalance. Results show that the Gradient Tree Boosting learner reaches very high $F1$ scores of 95% whilst both the Logistic Regression and Random Forests reach values of 90%.

A similar situation was observed in other scores and across all problems.

6.1.3 Other Scores

For the *ROCAUC* and *Accuracy* test scores, we can say that, in general, the best performing learner for all problems were the Gradient Boosting Trees models.[1] It scored high across all problems in the *ROCAUC* metric, where values ranged from 81.9% for the problem of tagging unidirectional migrants, up to 97% for the task of predicting which users had lived in the endemic region in the past.

We confirm there that this learner, being more complex than the others presented, effectively outputs higher scores in these metrics and that the model can better

1. There is only one specific case where the Boosted learner does not achieve the highest score, and that is for the problem of tagging users that moved out of the endemic region, but only for those users that are currently not endemic.

predict those different user migrations with lower bias. We still had to be careful not to overfit the model on the training set by configuring shallow base learners or by using stronger regularization of the parameters.

A similar statement can be said for the Random Forest model. For these metrics only, the results show that this model was in a 5% relative score distance to the Gradient Boosting model, except for the first problem of predicting which users were from the endemic region in the past, where the model was outperformed for more than 10%. We still see that this model had extracted enough predictive information from the dataset to correctly classify users according to these scores.

Under this view we would say that with this dataset provides enough information to effectively predicted long-term migrations with the Random Forest and Gradient Tree Boosting learners. The performance metrics were more than satisfactory, building best-fit models which could accurately tag users under the $ROCAUC$ and *Accuracy* metrics. Also, when compared to the Logistic Classifier, in some cases we see that they both have notable gain in the metrics and for others, such as for the first problem, the Logistic model is on par.

On the other end of the spectrum, we had that the Naive Bayes model had the worst $ROCAUC$ and *Accuracy* scores for all problems where it reached a maximum *Accuracy* of 84% for the problem of predicting users that were from the endemic region in T_0.

And the next best metric was a maximum $ROCAUC$ of 76% for the case of tagging migrants that moved out of the endemic region, but only making predictions over users that are non-endemic in T_1. These results show that we also had acceptable rates for a model that is not expected to perform.

Overall we found that there are discrepancy in the evaluation scores across the Machine Learning models here analyzed. Some are better at capturing the relevant information to predict human movements in the long-term. This is because not all social information is presented directly in the dataset, and feature interactions are helpful.

With this we showed that it is possible to use the mobile phone records of users during a bounded period (of 5 months) in order to predict whether they have lived in the endemic zone E_Z in a previous time frame (of 19 months). This task provided to be the easiest classification one, since it resulted in the highest scores across all learners.

The other effective tasks were the prediction of users that migrated in any direction to and from the endemic region, and the tagging of users migrating out of the endemic region, but only for those base of users which are currently non endemic.

Chapter 7

Conclusions

The purpose of this work was to analyze and explore how CDR data could be used as a surrogate to other, more traditional, methods of disease control and surveillance. We showed that it was possible to build strong classifiers to detect, from those users currently not endemic, which users had migrated in the past from the endemic region.

This was done starting from the hypothesis that long-term migrations are relevant to the Chagas disease spread. We analyzed people that moved from and to the regions of high vector-borne infections because we understood from other works in the literature that CDRs could be used for non business purposes. In this way we took advantage of the CDRs by leveraging the georeferenced information contained in them. In all, this work shows how the use of this data can help with the Chagas disease problem.

This analysis allowed us to expose that the Risk maps we constructed are of interest to national health authorities. These highlighted spots are likely of high disease prevalence and they are associated with communities which had strong calling interactions with the endemic region. These anomalous communities were highlighted by means of data features which were later confirmed to be relevant in helping detect long-term movement of users.

In this line, the risk maps produced interesting insights as to how migrations might be occurring at a national level at a low production cost. They also helped reinforce the hypothesis that these movements spread the disease out of the endemic region.

Added to the previous points, this work served to provide an example of how Machine Learning tools on CDR data can help authorities with disease-control strategies in a way which is not intrusive to cellphone users. We showed how the data can be effectively reused for reasons other than logging a user's calling expenses in a way which is relevant to public health issues.

Health-related measures can then be applied outside of the endemic region, and directed towards specific neighborhoods and communities. Next to traditional health

control actions, this research may provide a very insightful piece of information for decision-makers. The insights derived from the probabilistic models provide insights into the disease spread and it might also provide a research line for other diseases of similar characteristics.

We carried out experiments which show there is evidence that Machine Learning can provide a characterization of fluxes of human migrations and disease spread at the user level. These experiments with supervised classification models showed a systematic approach to analyze the problem with the information contained in the data. They effectively helped understand how relevant the dataset is to the question, and we explored how the algorithms used could assimilate the complexity and scale of the data. We saw that large, at-scale, and temporal user mobility was captured by the learners when using data from the CDRs, allowing us to understand a variety of human movements throughout the country.

From the prediction experiments, we had results that highlighted the benefit of using CDRs to characterize long-term human dynamics in the form of migration patterns. Most of the tasks showed high prediction scores across most of the metrics.

On the other hand, we also outlined the difficulty in the specific problem of tagging migrants that moved out of the endemic region at a national level. The same happened when problems were measured with an $F1$ evaluation metric: except for the problem of locating past endemic users, the algorithms achieved low prediction scores across all tasks.

In this work we did a thorough comparison in the strengths and weaknesses of the learners that were used to estimate the probability of each user's migrations. According to our results, there are algorithms which are reliable enough to detect users which have migrated from old regions.

In some instances, we have achieved high values across all scoring measures and we saw that, in general, the best learners for the tasks were the Random Forests and Gradient Boosting models. Because of this, we focused our work on tuning the ensemble learners to achieve higher predictions. This allowed us to find slight performance differences in these ensemble methods, where the Gradient Tree Boosting learner was found to be better than the Random Forest across most of the scores tested.

We also explored which feature categories were most relevant to the predictions in the ensemble algorithms, as selected by the "best-features" heuristic. Our findings selected the following feature categories as relevant: a) the area of influence of the users, where large mobility was indicative of past migrants, b) interactions and calling patterns with vulnerable users, and c) calls placed in the months nearest to T_0 and also, calls placed in December.

Since we performed our analysis on two Latin-American countries and in one single endemic region, we expect that this CDR usage can be extended to other similar countries and diseases, with demographic, cultural and geographical similarities beyond Chagasic spread in Mexico or Argentina.

The Argentinean and Mexican case studies allow us to find that it is possible to characterize human movements of long duration. In all, cellphone datasets are

rich enough in information to detect detailed patterns of users moving to and from a particular area, at a national level.

At the moment, health experts from the Mundo Sano Foundation state that, in Argentina, epidemic countermeasures include coordinating national surveillance systems with institutions and primary health care organizations, vector-centered policy interventions through fumigation of vector-infested regions and individual in-field screening of people. These measures require costly infrastructures to set up and be run.

On the other hand, this work provides a different point of view to the disease spread problem and it is built only on top of existing mobile networks. We know then that this analysis demands lower costs and takes advantage of the already available infrastructure.

The potential value these results could add to health research is hereby exposed. Finally, the results stand as a proof of concept which can be extended to other countries or to diseases with similar characteristics.

7.1 Discussion on the Methodology

There are several points that can be stated to point to weak spots on the methodology used.

An important observation is that we cannot directly jump from the human mobility insights found in this study, to conclusions from a model of disease spread. Since we were looking solely at human mobility to and from endemic regions, we can not imply disease spread or prevalence in a direct way. However, under the advice of this topic's researchers, this work does bring valuable insights to the problem, reinforcing previous hypothesis that vulnerable users are not only to be found inside vector-infested regions.

Added to the previous argument, we have stated numerous times before that the methods shown suffer on datasets with high target class imbalance. We saw, across all of the learners, low $F1$ scores on the problem of tagging user that migrated out of the endemic region. This is a very significant negative outcome using this methodology and these attributes.

Another issue with this work is the inherent biases in our dataset. As shown in Table 2.4.2, there are significant differences in the percentage of population represented by our dataset vs. current state distribution estimates. Due to this, we have to interpret results with caution, since we do not hold any real-data comparisons. We did not find any available georeferenced disease data outside of the endemic region, at municipal or regional levels.[1]

Other interesting biases stemming from the data are related to international pattern migrations. Here we measured migrations only from phone owners within a

1. As a matter of fact, we tried to contact health institutions and research organizations working on the matter with the purpose of enriching the original dataset. This attempt did not add any disease-related data of the kind.

single country, yet we have seen in Chapter 2, that epidemic regions traverse political borders. The current dataset is limited to only capturing national migrations, and with this our analysis is limited to movements of TelCo users interacting with national antennas only. A similar argument applies to the detection of mobility patterns for people with no cell-phone usage.

Our dataset also presents problems when processing users' home antennas. To do this we had to define what we thought were the expected "working hours" for all users and, in doing so, we assumed that this could be generalized to all samples. Even though this decision was supported from past research we referenced, the definitions might not be easily translated for datasets across other geographical and cultural regions.

Finally, we know that we can have significant time seasonality and stationarity in the data, yet we did not consider statistical methods to reduce this effect in a thorough fashion. When processing the data, we specifically tagged weekend and working hour attributes separately from the rest of the features. The same was done for month specific attributes on user calling patterns. However we understand that there could be other longer term time effect which was not considered under our current codebase. There are possibilities of migration patterns being strongly related to seasonal factors such as, for example, in agricultural workers having seasonal migrations around different regions.

The importance of other user specific biases, such as demographics unbalance, was not thoroughly examined. Yet these were out of the scope of this work, due to the lack of ground truth data.

7.2 Lines of Future Work

The mobility and social information extracted from CDRs analysis has been shown to be of practical use for long-term human migrations and for Chagas disease research. it adds value and information to help make data driven decisions which in turn is key to support epidemiological policy interventions in the region. For the purpose of continuing this research line, the following is a list of possible extensions:

Results validation. Compare observed experiment results of risk maps and best features against actual serology or disease prevalence surveys. Data collected from fieldwork could be fed to the algorithm in order to supervise the learning towards a target variable that is defined by the disease prevalence at a certain level.

Differentiating rural antennas from urban ones. This is important as studies show that rural areas have epidemiological conditions which are more favorable to the expansion of the disease expansion. The vector-borne transmission through *Trypanosoma cruzi* is helped with poor housing materials and domestic animals. All contribute to complete the parasite's life-cycle. Using the CDR data, antennas could be automatically tagged as rural by analyzing the

differences between the spatial distribution of the antennas in each area. A similar goal could be to identify precarious settlements within urban areas, with the help of census data sources.

International and seasonal migration analysis. Experts from the *Mundo Sano* Foundation underlined that many seasonal and international migrations occur in the *Gran Chaco* region. Workers are known to leave the endemic area for several months possibly introducing the parasite to foreign populations. The same happens with foreign workers, which are not part of our dataset. The mobility analysis on particular time periods or events, for example on holidays, or specific migrations from bordering countries, can give information on which communities have a higher influx of people from the endemic zone during a certain period. This additional analysis would also lower the bias of the current dataset.

Search for epidemiological data at a detailed level. For instance, specific historical infection cases. If we can split the endemic region according to the infection rate in different areas, or considering particular infections we can then attempt to build a model complex enough to detect infected users nationwide. For this to happen, it is of primary importance to have a reliable dataset available, otherwise it would not be possible to reach a classifier with good generalization performance.

Feature importance. From the best-features results shown in Chapter 6, we can think of further exploring Machine Learning methods that focus solely on exploring features' predictive power to determine how attributes are associated with a user's long-term migration. Also, from these same results, we find that it will be relevant to explore specific calling patterns on annual holidays during the last week of the year, from the 24 to the 31 of December.

Test other classifiers. The probabilistic models introduced in this work are a brief list of the ones available in the literature and of these, we know that only Naive Bayes had an acceptable classification performance in Problem 2. It would be interesting to see how other well-established and known methods perform for this case and see if any of them can correct this. Some examples of other algorithms not shown in this work are K-Nearest Neighbors, Support Vector Machines and Classification Neural Networks.

Test other pre-processing techniques. In line with the problem addressed by the previous point, there are multiple other data-based techniques to address the problem of learning with highly unbalanced classes. These tend to focus on under-sampling the majority class, over-sampling the minority class, or a combination of both.

Domain Evaluation. The evaluation of Machine Learning algorithms needs to incorporate domain specific knowledge to reassess the errors accordingly. In

this sense, we can say that Type I or Type II errors may not be the same at the eyes of a health researcher. In a future iteration, it would be interesting to evaluate again the results with the opinion of other domain experts where the algorithm's performance could be seen differently.

Bibliography

Y. Bengio and Y. Grandvalet. No Unbiased Estimator of the Variance of K-Fold Cross-Validation. *Journal of Machine Learning Research*, 5:1089–1105, 2004.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001a. ISSN 1573-0565.

L. Breiman. Statistical modeling: The two cultures. *Statistical Science*, 2001b.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

R. Briceño-Leon. Chagas disease in the Americas: an ecohealth perspective. *Cadernos de Saude Publica*, 25:S71–S82, 2009.

A. Carabarin-Lima, M. C. Gonzalez-Vazquez, O. Rodriguez-Morales, L. Baylon-Pacheco, J. L. Rosales-Encina, P. A. Reyes-Lopez, and M. Arce-Fonseca. Chagas disease (american trypanosomiasis) in Mexico: an update. *Acta tropica*, 127(2): 126–135, 2013.

V. S. Cherkassky and F. Mulier. *Learning from data : concepts, theory, and methods*. Hoboken, N.J. John Wiley, 2007. ISBN 978-0-471-68182-3.

E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *ACM SIGKDD*, pages 1082–1090. ACM, 2011.

R. Chunara and E. Nsoesie. Large-scale Measurements of Network Topology and Disease Spread: A Pilot Evaluation Using Mobile Phone Data in Cote d'Ivoire. *NetMob D4D Challenge*, pages 1–18, 2013.

A. Cruz-Reyes and J. M. Pickering-Lopez. Chagas disease in Mexico: an analysis of geographical distribution during the past 76 years-A review. *Memorias do Instituto Oswaldo Cruz*, 101(4):345–354, 2006.

B. C. Csáji, A. Browet, V. Traag, J.-C. Delvenne, E. Huens, P. Van Dooren, Z. Smoreda, and V. D. Blondel. Exploring the mobility of mobile phone users. *Physica A: Statistical Mechanics and its Applications*, 2012.

J. de Monasterio, A. Salles, C. Lang, D. Weinberg, M. Minnoni, M. Travizano, and C. Sarraute. Analyzing the spread of Chagas disease with mobile phone data. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, aug 2016. doi: 10.1109/asonam.2016.7752298.

E. Dumonteil. Update on Chagas' disease in Mexico. *Salud publica de Mexico*, 41(4): 322–327, 1999.

E. Enns and J. Amuasi. Human Mobility and Communication Patterns in Cote d'Ivoire: A Network Perspective for Malaria Control. *NetMob D4D Challenge*, pages 1–14, 2013.

Python. S. Foundation. Python language reference, version 3.5, June 2015. URL http://www.python.org.

Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning (ICML 1996)*, pages 148–156. Morgan Kaufmann, 1996. ISBN 1-55860-419-7.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.

M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

C. Guzman-Bracho. Epidemiology of Chagas disease in Mexico: an update. *TRENDS in Parasitology*, 17(8):372–376, 2001.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2009.

P. J. Hotez, M. E. Bottazzi, E. Dumonteil, J. G. Valenzuela, S. Kamhawi, J. Ortega, S. P. de Leon Rosales, M. B. Cravioto, and R. Tapia-Conyer. Texas and Mexico: sharing a legacy of poverty and neglected tropical diseases. *PLoS Negl Trop Dis*, 6 (3), 2012.

P. J. Hotez, E. Dumonteil, M. B. Cravioto, M. E. Bottazzi, R. Tapia-Conyer, S. Meymandi, U. Karunakara, I. Ribeiro, R. M. Cohen, and B. Pecoul. An

unfolding tragedy of chagas disease in north america. *PLoS Negl Trop Dis*, 7(10), 2013a.

P. J. Hotez, E. Dumonteil, M. J. Heffernan, and M. E. Bottazzi. Innovation for the 'bottom 100 million': eliminating neglected tropical diseases in the Americas. In *Hot Topics in Infection and Immunity in Children IX*, pages 1–12. Springer, 2013b.

L. Hyafil and R. L. Rivest. Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

G. M. James. Variance and bias for general loss functions. *Machine Learning*, 51(2): 115–135, 2003. ISSN 1573-0565.

Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, Apr. 2012. ISSN 2150-8097.

X. Lu, L. Bengtsson, and P. Holme. Predictability of population displacement after the 2010 haiti earthquake. *Proceedings of the National Academy of Sciences*, 109 (29):11576–11581, 2012.

J. M. Manne, C. S. Snively, J. M. Ramsey, M. O. Salgado, T. Barnighausen, and M. R. Reich. Barriers to treatment access for Chagas disease in Mexico. *PLoS Negl Trop Dis*, 7(10), 2013.

S. J. Mason and N. E. Graham. Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation. *Q.J.R. Meteorol. Soc.*, 128(584):2145–2166, 2002.

Ministerio de Salud Argentina. Plan Nacional de Chagas, 2016. URL http://www.msal.gob.ar/chagas/. [Online, accessed 9-may-2016].

T. Mitchell. *Machine Learning*. McGraw Hill, 1997. ISBN 0-07-042807-7.

D. Naboulsi, M. Fiore, S. Ribot, and R. Stanica. *Mobile traffic analysis: a survey*. PhD thesis, Universite de Lyon; INRIA Grenoble-Rhône-Alpes; INSA Lyon; CNR-IEIIT, 2015.

M. Navarro, B. Navaza, A. Guionnet, and R. Lopez-Velez. Chagas disease in spain: need for further public health measures. *PLoS Negl Trop Dis*, 6(12), 2012.

NetMob. Main conference on the scientific analysis of mobile phone datasets, 2016. URL http://netmob.org/. [Online, accessed 12-may-2016].

OPS. Estimacion Cuantitiva de la Enfermedad del Chagas en las Americas. *Organizacion Panamericana de la Salud/HDM/CD*, 425-06:1–6, 2006.

OPS. Mapa de Transmision vectorial del Mal de Chagas. *Organizacion Panamericana de la Salud*, 2014.

Orange D4D. Data for Development (D4D) Challenge, 2016. URL http://www.d4d.orange.com/. [Online, accessed 12-may-2016].

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

N. B. Ponieman, C. Sarraute, M. Minnoni, M. Travizano, P. Rodriguez Zivic, and A. Salles. Mobility and sociocultural events in mobile phone data records. *AI Communications*, 29(1):1–10, 2016.

C. Rasmussen and Z. Ghahramani. Occam's razor. In V. T. Leen, T.G. Dietterich, editor, *Advances in Neural Information Processing Systems 13*, pages 294–300, Cambridge, MA, USA, 4 2001. Max-Planck-Gesellschaft, MIT Press. ISBN 0-262-12241-3.

A. Rassi and J. M. de Rezende. American trypanosomiasis (Chagas disease). *Infectious disease clinics of North America*, 26(2):275–291, 2012.

C. Sarraute, J. Brea, J. Burroni, K. Wehmut, A. Ziviani, and I. Alvarez-Hamelin. Social Events in a Time-Varying Mobile Phone Graph. In *Fourth International Conference on the Analysis of Mobile Phone Datasets (NetMob)*, 2015a.

C. Sarraute, C. Lang, J. de Monasterio, and D. Weinberg. Descubriendo Chagas con big data. In *XVII Simposio Internacional sobre Enfermedades Desatendidas*, Aug 2015b.

G. A. Schmunis and Z. E. Yadon. Chagas disease: a Latin American health problem becoming a world health problem. *Acta tropica*, 115(1):14–21, 2010.

C. Song, Z. Qu, N. Blumm, and A.-L. Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

M. Tizzoni, P. Bajardi, A. Decuyper, G. K. K. King, C. M. Schneider, V. Blondel, Z. Smoreda, M. C. Gonzalez, and V. Colizza. On the use of human mobility proxies for modeling epidemics. *PLoS Comput Biol*, 10(7), 2014.

V. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer New York, 2000. ISBN 9781475732641.

P. Wang, T. Hunter, A. M. Bayen, K. Schechtner, and M. C. González. Understanding road usage patterns in urban areas. *Scientific reports*, 2, 2012.

A. Wesolowski, N. Eagle, A. J. Tatem, D. L. Smith, A. M. Noor, R. W. Snow, and C. O. Buckee. Quantifying the impact of human mobility on malaria. *Science*, 338 (6104):267–270, 2012.

A. Wesolowski, T. Qureshi, M. F. Boni, P. R. Sundsoy, M. A. Johansson, S. B. Rasheed, K. Engo-Monsen, and C. O. Buckee. Impact of human mobility on the emergence of dengue epidemics in Pakistan. *Proceedings of the National Academy of Sciences*, 112(38):11887–11892, 2015.

World Health Organization. Chagas disease (American trypanosomiasis). *World Health Organization Fact sheets*, 2016. URL http://www.who.int/mediacentre/factsheets/fs340/en/.

Appendices

Appendix A

Introduction to Machine Learning

A.1 Heaviside Function

The Heaviside step function is defined as

$$h(z) = \begin{cases} 0 \text{ if } z < 0 \\ \frac{1}{2} \text{ if } z = 0 \\ 1 \text{ if } z > 0. \end{cases} \tag{A.1.1}$$

and the relation between the logistic function and the heaviside function is

$$H(z) = \lim_{k\to\infty} \left(\frac{1}{2} + \frac{1}{2} \tanh(kz) \right) = \lim_{k\to\infty} \left(\frac{1}{1+e^{-kz}} \right) \tag{A.1.2}$$

Note in this sense that for a binary classification problem, the heaviside function Equation (A.1.1) is more appealing to the task, although its singularity at $z = 0$ is problematic for optimization routines.

A.2 Details of the log loss functions

In Figure A.2.1 we display the values of the log loss function for different input probabilities:

We have that it is a negative, concave, monotone increasing function with the estimated probability output by the classifier. Also, the function is heavier for small probability values near zero than it is positive for large values near one. This means that it penalizes strongly on wrong predictions; samples where the algorithm has a strong confidence in the prediction yet has misclassified. This properties can be taken into advantage during the optimization procedure, which will always converge to a global optima.

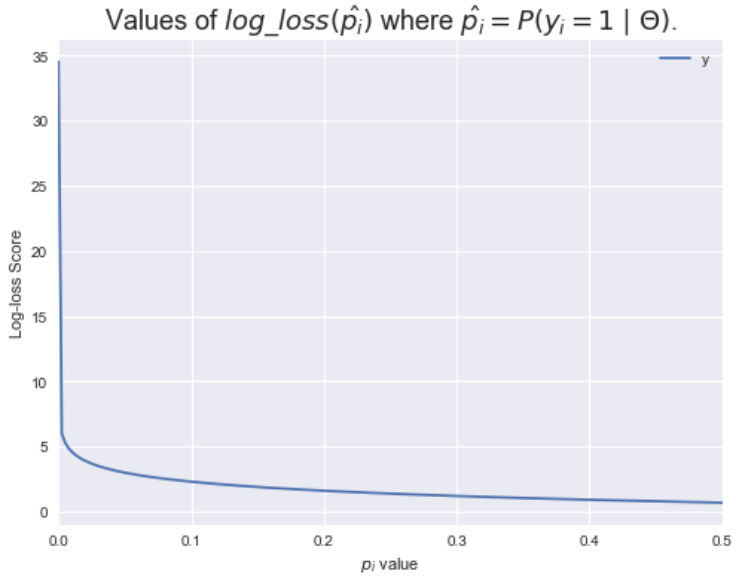


Figure A.2.1 Error score series plot for the log loss function. Different p_i input values are displayed with their output log loss score.

Also, explicit forms can be given for the gradient and the Hessian of the loss function with regard to the whole dataset X as a sum involving individual samples x.

The negative gradient can be analytically expressed in the following way:

$$-\nabla l(\theta) = \sum_{i=1}^{N}(y_i - P(y_i \mid x_i, \theta)) \cdot x_i = \mathbf{X}^\intercal(\mathbf{y} - \mathbf{p}) \tag{A.2.1}$$

whilst the Hessian takes the following form:

$$\frac{\partial^2 l(\theta)}{\partial\theta\partial\theta^\intercal} = \sum_{i=1}^{N} x_i \cdot x_i^\intercal P(y_i \mid x_i, \theta)(1 - P(y_i \mid x_i, \theta)) \tag{A.2.2}$$

With this formulation, we can take any optimization procedure that benefits from the closed-form representations of the first and second derivatives of the loss function.

Appendix B

Model Selection

B.1 Bias and variance errors in other loss functions

The work in [James, 2003] defines what properties should be displayed by the bias and variance of an estimator in the general case.

For this, they define the systematic value of a random variable with respect to a loss function.

Definition B.1.1 Systematic Value: Given a training set $\mathcal{T}$, a loss function $L(\cdot)$ and a random variable Z, the systematic value or systematic component of this variable is

$$SZ = \underset{u}{\operatorname{argmin}}\, \mathbb{E}_Z \left[L(u, Z) \right]$$

The systematic value is the nearest constant value to the random variable, with the measure as given by the loss function. In this definition we are implicitly assuming that the necessary finiteness conditions of the expectation of the loss function with the random variable exist.

The systematic value for the input and target varialbe then the same way as with the squared loss function in Equation (4.1.1), where the expectation is over the target's distribution.

For the general setting, the author argues that the bias should be the measure of the systematic difference in which a random variable differs from a particular target value and it should also be the measure of how much this systematic difference contributes to the error.

On the other hand the variance of an estimator should measure the spread of the estimator around its systematic component and it should also capture be the effect this variability has on the prediction. In this sense, the author defines the variance and bias of the learner to be:

$$
\begin{aligned}
Bias(\hat{f})^2 &= L(S\hat{f}, SY) \\
Var(\hat{f}) &= \mathbb{E}_{\hat{f}}\left[L(\hat{f}, S\hat{f})\right]
\end{aligned}
\tag{B.1.1}
$$

With these definitions we have that the variance is an operator defined only for the estimator and which is null only when the predictor is a constant value for any training set. As such, it is unchanged by new data. The bias on the other hand is an operator built from the systemic values of Y and $\hat{f}$ and is null only if both of these are equal.

The generalized bias-variance decomposition now takes the following form, where the bias and variance of the squared loss are replaced with more general properties such as the variance effect and the systematic effect:

$$
\begin{aligned}
\mathbb{E}_{X,Y}\left[L(\hat{f}(X), Y)\right] = &\mathbb{E}_Y\left[L(SY, Y)\right] + \\
&\mathbb{E}_Y\left[L(S\hat{f}(X), Y) - L(SY, Y)\right] + \\
&\mathbb{E}_{Y,X}\left[L(\hat{f}(X), Y) - L(S\hat{f}(X), Y)\right]
\end{aligned}
\tag{B.1.2}
$$

Note that the first term is equal to the variability of the target variable with respect to its systematic value, the second term is the systematic effect. That is, the expected bias between the systematic values of Y and $\hat{f}(X)$. The last term is the variance effect which is the expected variability between Y and $\hat{f}(X)$.

For the case of supervised classifiers, loss functions are also called *metrics* and they originate from *Type I & Type II* errors common in statistics but have different meanings in this context. There are a number of variations for classification metrics. The suitability of each will always depend on the problem, since they differ on what type of prediction errors are more valuable to minimize.

With the definition above, the systematic and variance values then become

$$
\begin{aligned}
SY &= \operatorname*{argmax}_{1 \le i \le K} P(Y = i) \\
S\hat{f}(X) &= \operatorname*{argmax}_{1 \le i \le K} P(\hat{f}(X) = i) \\
Var(Y) &= 1 - P(Y = SY) \\
Var(\hat{f}(X)) &= 1 - P(Y = S\hat{f}(X))
\end{aligned}
\tag{B.1.3}
$$

With the above, we have that the decomposition of the prediction error among variance, systematic and variance effect becomes

$$
Var(Y) + P(Y = SY) - \sum_{i=1}^{K} P(\hat{f}(X) = i) P(Y = i)
\tag{B.1.4}
$$

which again relies heavily on the variability of the target and on how well the classifier approximates the labels.

Summarizing all of the above we have that the classification prediction error will be as

(B.1.5)

$$EPE = \mathbb{E}_X \left[\sum_{k=1}^{K} L(Y_K, \hat{f}(X)) P(Y_k|X) \right] = \mathbb{E}_X \left[\sum_{k=1}^{K} I(Y_K \neq \hat{f}(X)) P(Y_k|X) \right]$$

B.2 The Vapnik-Chervonenkis (VC) Dimension

In Vapnik and Chervonenkis' work, the focus is put on the estimation of the prediction error through the convergence and consistency of the training error. To do this, the authors build a measure, the Vapnik-Chervonenkis (VC) dimension, as an important value in determining convergence speeds for different Machine Learning algorithms. All of this theory is given in the context of functional spaces.

The VC Dimension is a number based on what is called *Statistical Learning Theory* (SLT). The theory is used to bound estimates of the true expected prediction error from finite i.i.d. samples. Its advantage is that it requires weak assumptions on the data and the structure of the model, and the results are distribution independent.

Here we only mention the VC dimension for the purpose of showing a theoretical approach to error estimation in Machine Learning methods. Most of the results and reasoning follows the works in [Cherkassky and Mulier, 2007]. For a more broad development of this work, [Vapnik, 2000] is recommended too. Here we present a brief comment on the consistency of the training error with respect to the prediction error and give example bounds for classification learners.

Vapnik-Chervonenkis discuss a theoretical framework in which the dimension of a class of approximating functions is a quantity that can be constructed from the data. This allows us to have analytical estimates on the distance between the empirical and generalization errors. This is done for a number of known Machine Learning algorithms and the theory also provides methods to calculate these dimensions in a constructive way and for a broad class of functions. For cases in VC dimensions can not be calculated, authors propose sampling or experimental methods to bound it.

Let $\mathcal{F} = \{f(x, \theta) \mid x \in X, \theta \in \Theta\}$ be a class of classifiers where $f_\theta : X \to Y \, \forall f \in \mathcal{F}$. This is a class of functions with domain over the input dataset X and which are indexed by a parameter. As an example, in logistic regression, the paramaters θ can be indexed by the values of each feature's weights x^p. The class F will then represent the set of all possible functions from which to select the optimal model.

As it was seen in Chapter 3, a common problem in supervised learning happens when a particular learner overfits the data. In practice, we find that this is common when n is *small* or when the number of features is *large*.

Incorporating the loss functions directly into the model, SLT looks at functions

of the form

$$Q(t,\theta) \;=\; L(f_\theta(x,y))$$

The function $Q(\cdot,\cdot)$ is bivariate, and we note $t=(x,y)$ to represent a sample of input and output data expressed as a random variable.

In this way, the $EPE(\theta)$ for any model takes the functional form

$$\begin{aligned} EPE(\theta) &= \mathbb{E}_{\mathbf{t}}\left[Q(t,\theta)\right] \\ &= \int Q(t,\theta)p(t)dt \\ &= \int L(f_\theta(x),y)p(x,y)dxdy \end{aligned} \tag{B.2.1}$$

where we let $p(t)$ be the **true** underlying probability function of the data. For simplicity, in this analysis we will assume the functions Q in the class to be bounded.

Knowing that we only have access to restricted data, we would like to estimate the risk functional for that functional space, only with the available information given by training and test errors. In this framework, this is called the empirical risk:

$$Err_{train}(\theta) = \sum_{i=1}^{n} Q(t_i,\theta) \tag{B.2.2}$$

Note that the learner is estimated directly through the risk functional, which in turn is shaped by the loss function. On the other hand, the unknown true underlying distribution $p(t)$ will not be estimated to minimize the functional and SLT puts focus in using the training error as a *good* substitute for the prediction error.

Definition B.2.1 Consistency of the Training error

As a principal characteristic of SLT, we have that if we increase the sample size, we would still want to have the training error converge and to be consistent with the prediction error of the models.

More specifically, let $\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_n,\ldots\}$ be a set of training samples such that $\forall n\ |T_n|= n$. Given a training sample of size n and a class of learners Θ, let θ_n^* be the argument minimizing the training error and let θ_0 be the minimizing argument for the prediction error over the class of functions F i.e. over all the models of this class.

Then, the training error is said to be consistent if

$$\lim_{n\to\infty} Err_{train}(\theta_n^*) \;=\; \lim_{n\to\infty} EPE(\theta_n^*) \;= EPE(\theta_0) \tag{B.2.3}$$

The property of asymptotic convergence in the training and prediction errors is what is expected for any learning algorithm. In SLT, the consistency requires that the training and prediction errors' sequences not only converge to the same values,

but that the sequence of minimal training errors also converge, to the minimizing value of the prediction error.

In reality, it is reasonable to think that the prediction error approximation using the training error introduces a strong overestimation. As we know, the training error will be biased by the sample used whilst the prediction error, which is given for the whole class, is not dependent on the sample used.

This is a very important condition, because by it the training error minimization when using bounded loss functions will be consistent if and only if:

$$\forall \epsilon > 0 \ , \ \lim_{n\to\infty} P\left[\sup_{\theta\in\Theta} \mid Err^n_{train}(\theta) - EPE(\theta) \mid > \epsilon\right] = 0 \tag{B.2.4}$$

Here the training error $Err^n_{train}(\theta)$ is the error's value when using a sample of size n. In this sense, the training error is said to be consistent if it converges uniformly in probability over the whole class of functions[1]. This implies that the capacity of the learner will be captured in the set of functions $Q(t, \theta)$ used to approximate the data.

Four our work in binary classification, the main results of the VC theory establish that, for any given $\epsilon > 0$

$$P\left(|Err^n_{train}(\theta^*) - EPE(\theta_0)| > \epsilon \right) \leq 8 poly(\Theta, n) \exp\left(\frac{-n\epsilon^2}{32}\right) \tag{B.2.5}$$

where $poly(\Theta, n)$ is a polynomial on n whose coefficients depend only on VC dimension of the class of functions F. For practical applications though, use of these bound estimates is limited by the calculation of the VC value. This becomes increasingly difficult in algorithms which are more complex and expressive, such as in neural networks. Due to this limitation, we show in Section 4.3 an approach which is, in comparison, widely practiced. It has the advantage of being easier and a simple heuristic that estimates the EPE.

B.2.1 VC for Prediction Error Bounds

The next results will be focused on a binary supervised machine learning classifieres. Yet if the reader would like to explore a more general list of results on prediction error bounds for binary classifiers, these can be found in [Cherkassky and Mulier, 2007].

To begin, we need to define what we mean by the shattering coefficient of a functional space.

Definition B.2.2 Shattering

Let $\mathcal{A} = \{A_1, A_2, \dots\}$ be a set family and T a finite set. Let $t \subseteq T$, it is said that $\mathcal{A}$ picks out t if there exists $A' \subseteq \mathcal{A}$ such that $T \cap A' = t$. T is said to be shattered by $\mathcal{A}$ if it picks out all its subsets.

1. Remember that in SLT the approximating functions $f \in F$ are indexed by the θ parameter.

The n-th shattering coefficient Δ_n of a class $\mathcal{A}$ is defined to be the maximum number of subsets of n elements picked out by the class.

If, for each training set $\mathcal{T}$ of size n, we consider a learned function from our set of classifiers: We can think of each classifier acting as an indicator function on the inputs $\{x_1, x_2, \ldots, x_n\}$. The *diversity* of this set of classifiers intuitively represents all the different ways in which the input sample can be partitioned by the classifiers.

We would say that t is picked out by Θ if there exists a classifier $f_\theta \in \Theta$ such that $T = f_\theta^{-1}(\{1\})$. In this way, the classifiers in Θ define a unique mapping to the class of sets where each classifier is positive. Taking this into account, it is said that $\mathcal{F}$ shatters a set A if all its subsets are picked out by the class of functions.

We will now reproduce the main necessary and sufficient conditions to have uniform consistency in predictive error approximation of our defined class of loss functions. These are needed to provide the explicit bounds for the exponential convergence of this error. The benefit of these bounds are that they are built and can be calculated for a number of known classifiers such as linear regressors and support vector machines.

Definition B.2.3 Vapnik-Chervonenkis (VC) Dimension

The Vapnik-Chervonenkis Dimension (VC) of a class of binary functions is the cardinality of the largest set which is shattered by $\mathcal{F}$.

Note that by definition this means that there needs only to exist one set shattered by $\mathcal{A}$, to have the VC dimension at least as big as that set's cardinality.

With this dimension, we can give a certain criteria for measuring the complexity of a class of binary functions by evaluating its expressiveness. Note however that it need not be finite.

As a simple example, one could use a linear regression of d features

$$g_\theta = \sum_{i=1}^{d} x_i \theta_i + \theta_0 \tag{B.2.6}$$

as a classifier if we consider the indicator function of the positive half-plane induced by the regression:

$$f_\theta = I(\sum_{i=1}^{d} x_i \theta_i + \theta_0 > 0) \tag{B.2.7}$$

This class of approximating functions can shatter up to $d + 1$ samples, but no bigger sample. Thus the VC dimension is exactly $d + 1$. The proof relies can be given on induction on the number of dimensions. Still we omit it and refer the reader to the bibliography [Cherkassky and Mulier, 2007] Pg. 113 for the details. Other examples are given, both for finite and infinite VC classes.

SLT proves that for classes which have a finite VC dimension h, the n-th shattering coefficient is bounded by a polynomial of order equal to the dimension i.e. $\Delta_n(\mathcal{F}) \leq$

$O(n^h)$,[2].

Also, if we have a uniformly bounded class of risk functions such as

$$Q(t,\theta) \leq A,\ \forall t \in \mathcal{T}, \theta \in \Theta \tag{B.2.8}$$

we can then define $\eta \in (0,1)$ to have that with probability at least $1-\eta$ and $\forall \theta \in \Theta$

$$EPE(\theta) \leq Err^n_{train}(\theta) + \frac{A\epsilon}{2}\left(1+\sqrt{1+\frac{4Err^n_{train}(\theta)}{A\epsilon}}\right) \tag{B.2.9}$$

$$\epsilon = a_1 \frac{h\left(\ln(\frac{a_2 n}{h}) - \ln(\frac{\eta}{4})\right)}{n} \tag{B.2.10}$$

, and when the approximating class F is finite of size d:

$$\epsilon = 2\frac{\ln(d) - \ln(\eta)}{n} \tag{B.2.11}$$

which is a simpler relation for ϵ.

In the preceding equations, the values of constants a_1 and a_2 are related to the nature of the density function $p(t)$ of the data. However, its values are proven to be uniformly bounded for all distributions, with $a_1 \in (0,4]$ and $a2 \in (0,2]$.

As a last result, the authors show that a more precise bound can be given for the function that minimizes the empirical risk $Err^n_{train}(\theta^*)$. They show that with probability $1-2\eta$

$$Err^n_{train}(\theta^*) - EPE(\theta_0) \leq A\sqrt{\frac{-\ln(\eta)}{2n}} + \frac{A\epsilon}{2}\left(1+\sqrt{1+\frac{4}{\epsilon}}\right) \tag{B.2.12}$$

These results prove that effective approximations of the prediction error can be given for most algorithms of finite VC dimensions. They also give a distinct characterization of how model complexity is related to the prediction error estimation. All of them are explained in detail in [Vapnik, 2000], Ch. 3.

2. $O(\cdot)$ corresponds to Big-O notation.

B.3 Choice of K parameter

In synthetic and actual dataset [Hastie et al., 2009] P. 243, have found that using a *higher* value of K, relative to the size of the dataset, means having a bigger training fold since each left-out partition is only $\frac{N}{K}$ in size. The results show models with good bias but high variance, which is in conformity with the small size of the validation set. Note that having a higher value for K will effect to a higher computational burden since K estimators need to be fitted.

Having a lower K value means using a smaller training set. Then it is common to find models with lower variance and higher bias. Empirical results suggest that to show that the EPE is overestimated. This is because having less available data to fit the model, implies having worse estimates from asymptotic results.

One last common choice for K is $N - 1$. This scenario is known as *leave-one-out CV* and is a very used format for problems where data arrives sequentially over time. Here, samples of the training set $t_i = (\boldsymbol{x_i}, \boldsymbol{y_i})$ are accessible only in an orderly fashion. For these cases model evaluations are made against the new sample and the training set is updated with every arrival.[3]

A valuable survey and explanation of the drawbacks of the K-Fold CV estimator can be found in [Bengio and Grandvalet, 2004]. The authors prove that there is no distribution-free estimator unbiased estimator of the K-Fold cross validation estimator's variance. The theoretical arguments focus on the idea that error correlations between the training and validation sets are not taken into account by the CV procedure. These correlations are known and understood for the general case, but their estimation is not possible. As a consequence of this, comparison between different possible models are hindered.

Empirical examples are built from synthetic datasets to show the shortcomings of the cross validation's algorithm which might show high deviations from its central value. Some exceptions appear though, specifically where distribution-free bounds can be found for a class of approximating functions. But these cases are specific only for this class. In general, these bounds are not known, so using an unknown deviation of the CV estimator will affect the evaluation of different models.

Consensus[4] is that in general CV is a good procedure to estimate the expected prediction error with the training set fixed, but not good for the prediction error, conditional on the training set $\mathcal{T}$ fixed.

3. Note that here we must assume samples to be *exchangeable*. This means that the training set distribution is not altered by a permutation of samples $F(x_1, \ldots, x_n) = F(x_\sigma 1, \ldots, x_\sigma n)$ for any random permutation σ.

4. [Hastie et al., 2009] P. 260

Appendix C

Ensemble Methods & Naive Bayes Classifier

C.1 Decision Tree Pruning

Let $T \subset T_0$ be a subtree of the first tree, where T is obtained by pruning T_0. Here the partition regions R_j will be associated to T's terminal nodes or leafs, indexed by j, which $j \in \{1, \ldots, |T|\}$.

Given a loss function, and a problem with K possible target classes, we can define the following values:

$$\begin{aligned} N_j &= \mid \{x \in R_j\} \mid \\ \hat{p}_{jk} &= \frac{1}{N_j} \sum_{x \in R_j} I(y = k) \\ c_j &= argmax_k \ \hat{p}_{jk} \end{aligned} \tag{C.1.1}$$

As we have mentioned before, at each split we use the node impurity measure to quantify this action. Here, we will denote $Q_j(T)$ to be the impurity measure for region j. With this we will have an impurity measure for each region R_j and the algorithm will then aggregate all of them into a single measure called the *cost complexity criterion*. The idea is that this loss function, as a function of a tree, will control the whole optimization procedure through the tree's parameters. The criterion will be managed to control the final model's bias and variance. We define it in the following way

$$C_\alpha(T) = \sum_{j=1}^{|T|} N_j Q_j(T) + \alpha |T| \tag{C.1.2}$$

Here $\alpha \in \mathbb{R}_{\geq 0}$ is a tuning parameter that values the trade-off between the tree complexity, as given by its depth, and the accuracy of the model as given by the measure we proposed in Section 5.1.2. The idea is that given an α we find the subtree $T_{alpha} \subset T_0$ that minimizes Section C.1.

There are various methods we could use to find the optimal subtree. As an example, here we give an example of the *weakest link pruning* algorithm which goes as follows:

Let $B(T) = \sum_j N_j Q_j(T)$ be our pure loss function, without any complexity cost added. [Breiman et al., 1984] shows that we can find T_{alpha} included in a sequence of trees built for this. This sequence is constructed by iteratively pruning the node j that, when removed from the tree, creates the smallest increase in $B(T)$.

In this way, we'll have a sequence of trees $T_0, T_1, \ldots, T_l$ and a sequence of nodes $j_0, j_1, \ldots, j_l$ respectively the ones minimizing the increase in $B(T_0), B(T_1), \ldots, B(T_l)$ at each step. The algorithm will stop when have reached the root node and we will find our tree T_{alpha} by comparing all the $C_\alpha(T)$ for all of the trees built in the sequence. In practice, it is common to have this procedure done within a K-fold cross validation routine to reach to an estimated $\hat{\alpha}$.

C.2 Random Forests' Predictive Error Bounds

Define

$$\hat{j}(\mathbf{x}, \mathbf{y}) = \underset{j \neq \mathbf{y}}{\operatorname{argmax}}\, P_\Theta(h(\mathbf{x}) = j)$$

and let the margin function for a random forest (not a group of classifiers) be defined as

$$mr(\mathbf{x}, \mathbf{y}) = P_\Theta(h(\mathbf{x}) = \mathbf{y}) - P_\Theta(h(\mathbf{x}) = \hat{j}) = \mathbb{E}_\Theta\left[I(h(\mathbf{x}, \Theta) = y) - I(h(\mathbf{x}, \Theta) = \hat{j})\right]$$

This characterizes the expectation taken over another function which is called the **raw margin function**. Intuitively, the raw margin function takes each sample to be 1 or -1 according to whether the ensemble classifier can correctly classify or not the sample's label, given Θ.

Let the strength of the set of weak classifiers in the forest be defined as

$$s = \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[mr(\mathbf{x}, \mathbf{y})\right] \tag{C.2.1}$$

Define $\rho(\Theta, \Theta')$ as the correlation between $rmg(\Theta, \mathbf{x}, \mathbf{y})$ and $rmg(\Theta', \mathbf{x}, \mathbf{y})$ of two weak learners, then we can have the mean value correlation for the ensemble as

$$\overline{\rho} = \frac{\mathbb{E}_{\Theta,\Theta'}\left[\rho(\Theta, \Theta')\sigma(\Theta)\sigma(\Theta')\right]}{\mathbb{E}_{\Theta,\Theta'}\left[\sigma(\Theta)\sigma(\Theta')\right]} \tag{C.2.2}$$

With these definitions we can then see that,

Theorem C.2.1 *There exists an upper bound for the generalization error of a Random Forest which is*

$$PE^* \leq \overline{\rho}\frac{(1-s^2)}{s^2} \tag{C.2.3}$$

Proof. It is straight to see that

$$mr(\mathbf{x},\mathbf{y})^2 = \mathbb{E}_{\Theta,\Theta'}\left[rmg(\Theta,\mathbf{x},\mathbf{y})\ rmg(\Theta',\mathbf{x},\mathbf{y})\right] \tag{C.2.4}$$

This in turn implies that

$$\begin{aligned} var(mr) &= \mathbb{E}_{\Theta,\Theta'}\left[cov_{\mathbf{x},\mathbf{y}}(rmg(\Theta,\mathbf{x},\mathbf{y})rmg(\Theta',\mathbf{x},\mathbf{y}))\right] \\ &= \mathbb{E}_{\Theta,\Theta'}\left[\rho(\Theta,\Theta')\sigma(\Theta)\sigma(\Theta')\right] \end{aligned} \tag{C.2.5}$$

where $\sigma(\Theta)$ is the standard deviation of $rmg(\Theta,\mathbf{x},\mathbf{y})$. In both cases, Θ and Θ' are given.

Equation (C.2.5) in turn implies that

$$\begin{aligned} var(mr) &= \overline{\rho}(\mathbb{E}_{\Theta}\left[\sigma(\Theta)\right])^2 \\ &\leq \overline{\rho}\,\mathbb{E}_{\Theta}\left[var(\Theta)\right] \end{aligned} \tag{C.2.6}$$

Assuming that $s \geq 0$ we have that the prediction error is bounded by

$$PE^* \leq var(mr)/s^2 \tag{C.2.7}$$

by Chebyshev's inequality. On the other we also have that

$$\begin{aligned} \mathbb{E}_{\Theta}\left[var(\Theta)\right] &\leq \mathbb{E}_{\Theta}\left[\mathbb{E}_{\mathbf{x},\mathbf{y}}\left[rmg(\Theta,\mathbf{x},\mathbf{y})\right]\right]^2 - s^2 \\ &\leq 1 - s^2 \end{aligned} \tag{C.2.8}$$

We can use Equation (C.2.6), Equation (C.2.7) and Equation (C.2.8) to establish the upper bound for the prediction error we are looking for

$$PE^* \leq \overline{\rho}\frac{(1-s^2)}{s^2} \tag{C.2.9}$$

□

C.2.1 Binary Class Bounds

In the context of a binary class problem, where the target variable can only take two values, there are simplifications to the formula Equation (C.2.3). In this case, the

margin function takes the form of $2P_\Theta(h(\mathbf{x}) = \mathbf{y}) - 1$ and similarly the raw margin function results in $2I(h(\mathbf{x}, \Theta) = \mathbf{y}) - 1$.

The bounds prediction error bounds derived in Equation (C.2.7) assume that $s > 0$ which in this case results in

$$\mathbb{E}_{\mathbf{x},\mathbf{y}} \left[P_\Theta(h(\mathbf{x}) = \mathbf{y}) \right] > \frac{1}{2} \tag{C.2.10}$$

Also, the correlation between $I(h(\mathbf{x}, \Theta) = \mathbf{y})$ and $I(h(\mathbf{x}, \Theta') = \mathbf{y})$, denoted $\overline{\rho}$ will take the form

$$\overline{\rho} = \mathbb{E}_{\Theta,\Theta'} \left[\rho \left(h(\cdot, \Theta), h(\cdot, \Theta') \right) \right] \tag{C.2.11}$$

C.3 Random Forests' Margin Function Convergence

C.3.1 Proof

The proof follows from seeing that given a training set, a tree Θ and a class j then

$$\forall \mathbf{x} \; P_\Theta(h(\theta, \mathbf{x}) = j) \; = \; \lim_{L \to \infty} \frac{1}{L} \sum_{l=1}^{K} I(h_l(\mathbf{x}) = j) \tag{C.3.1}$$

almost surely.

This is because if we consider the nature of the tree, we see that the set $\{\mathbf{x}/h_l(\mathbf{x}, \Theta) = j\}$ is built as a union of hyper-rectangles partitioning feature space. And given the finite size of the training set, there can be but a finite set of these unions of hyper-rectangles for all the input data. Let $S_1, \ldots, S_M$ be an indexation of these unions and define $\phi(\Theta) = m$ if $\{\mathbf{x}/h(\mathbf{x}, \Theta) = j\} = S_m$.

We denote by L_m the number of times that $\phi(\Theta_l) = m$, where $l \in 1, \ldots, L$ and L is the total number of trees in this forest.

It is immediate that

$$\frac{1}{L} \sum_{l=1}^{L} I(h_l(\mathbf{x}, \Theta) = j) \; = \; \frac{1}{L} \sum_{m=1}^{M} L_m I(\mathbf{x} \in S_m) \tag{C.3.2}$$

and that following the law of large numbers, there is a convergence almost everywhere of

$$\frac{L_m}{L} = \frac{1}{L} \sum_{l=1}^{L} I(\phi(\Theta_l) = m) \xrightarrow[L \to \infty]{} P_\Theta(\phi(\Theta) = m). \tag{C.3.3}$$

If we let $C = \bigcup_{m=1}^{M} C_m$ where each C_m are zero-measured sets representing the points where the sequence is not converging. If we combine Section C.3.1 and Equation (C.3.3), we will finally have that outside of C,

$$\frac{1}{L}\sum_{l=1}^{L} I(h_l(\mathbf{x}) = j) \xrightarrow[L\to\infty]{} \sum_{m}^{M} P_\Theta(\phi(\Theta) = m) I(\mathbf{x} = j) \;=\; P_\Theta(h(\mathbf{x}, \Theta) = j) \tag{C.3.4}$$

C.4 Gradient Boosting heuristic optimization

The first take on this optimization problem goes using a greedy optimization routine. One tree is fit at a time and new trees are then successively added in later steps to improve on previous trees' errors.

Let t be the step indexer of the algorithm, where $t \in 0, .., K$, $Obj_t(\Theta)$ be the objective function and $\hat{y}^t$ be the target variable respectively. Then the i-eth target's value at each step would iterate in the following way:

$$\begin{aligned} \hat{y}_i^0 &= 0 \\ &\cdots \\ \hat{y}_i^t &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i) \end{aligned} \tag{C.4.1}$$

where each tree is added in such a way that we are minimizing

$$Obj^t(\theta) = \sum_{i}^{n} L(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + c(t) + R(f) \tag{C.4.2}$$

Note that we have included here a regularization term (see section Section 3.5) R on all of the weak learners. For most cases, this term will be in the form of a Tikhonov regularization. This will add another complexity tuning parameter to control the length of the overall procedure $c(t)$ which is variable only in t.

If we assume we have sufficient conditions to approximate the objective function with second order Taylor approximation around $f_t(x_i)$,we would have

$$Obj^t(\theta) \approx \sum_{i}^{n} L(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i, \theta_t) + \frac{1}{2} h_i f_t(x_i, \theta_t)^2 + R(f(\Theta)) + c(t) \tag{C.4.3}$$

Here g_i and h_i are first and second order approximations of the loss function with,

$$
\begin{aligned}
g_i &= \frac{\partial L(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}}, \\
h_i &= \frac{\partial^2 L(y_i, \hat{y}_i^{t-1})}{\partial (\hat{y}_i^{t-1})^2}
\end{aligned}
\tag{C.4.4}
$$

Still, the Equation (C.4.3) can be simplified by taking only the terms that are dependent on θ. This also means replacing the actual tree's predictions for each sample as $\theta_{q(x_i)}$, where $q(\cdot) : X \to leaf$ is the function that maps samples to the tree's leaves. Then,

$$
Obj^t(\theta) \approx \sum_i^n g_i \theta_{q(x_i)} + \frac{1}{2} h_i \theta_{q(x_i)}^2 + \gamma(t-1) + \frac{1}{2}\lambda \sum_{j=1}^{t-1} \theta_j^2 \tag{C.4.5}
$$

As an example, we have already replaced the regularization terms $c(t)$ and $R(f)$ with penalties on the size of the ensemble and with an $l2$ penalty on the weight of each individual leaf.

If we rearrange the equation above we get

$$
\begin{aligned}
Obj^t(\theta) &\approx \sum_{j=1}^{t-1} \left(\sum_{i \in \{q(x_i)=j\}} (g_i)\theta_j + \frac{1}{2} \sum_{i \in \{q(x_i)=j\}} (h_i + \lambda)\theta_j^2 \right) + \gamma(t-1) \\
&\approx \sum_{j=1}^{t-1} \left(\theta_j \sum_{i \in \{q(x_i)=j\}} (g_i) + \frac{\theta_j^2}{2} \sum_{i \in \{q(x_i)=j\}} (h_i + \lambda) \right) + \gamma(t-1)
\end{aligned}
\tag{C.4.6}
$$

which, as a function of θ is a quadratic equation if we assume γ to be fixed. This results in a convenient and closed-form analytical formulation to select the value at step t. In this sense, a greedy direct optimization approach, such as gradient descent, can be used to find the tree $f_t(\theta)$ minimizing the previous expression.

As we have stated before the approach assumes that we have met enough smoothness conditions on the loss function with respect to the prediction variable and that these values are actually computable. This is why smooth loss functions play an important part here in providing a feasible method.