2016-06-01

Machine Learning for Disease Prediction

Abraham Jacob Frandsen
Brigham Young University

Machine Learning for Disease Prediction

Abraham Jacob Frandsen

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Jeffrey Humpherys, Chair
Tyler Jarvis
C. Shane Reese

Department of Mathematics

Brigham Young University

June 2016

ABSTRACT

Machine Learning for Disease Prediction

Abraham Jacob Frandsen
Department of Mathematics, BYU
Master of Science

Millions of people in the United States alone suffer from undiagnosed or late-diagnosed chronic diseases such as Chronic Kidney Disease and Type II Diabetes. Catching these diseases earlier facilitates preventive healthcare interventions, which in turn can lead to tremendous cost savings and improved health outcomes. We develop algorithms for predicting disease occurrence by drawing from ideas and techniques in the field of machine learning. We explore standard classification methods such as logistic regression and random forest, as well as more sophisticated sequence models, including recurrent neural networks. We focus especially on the use of medical code data for disease prediction, and explore different ways for representing such data in our prediction algorithms.

Keywords: preventive healthcare, disease prediction, chronic diseases, machine learning, sequence classification, recurrent neural networks, ICD-9 codes, survival analysis

Acknowledgments

I thank Dr. Jeffrey Humpherys for his tireless efforts to procure data as well as the many ideas and nuggets of wisdom imparted to me. I thank the BYU Department of Mathematics for its financial support, which has allowed me to enjoy the student lifestyle far longer than most are able. I thank my parents and siblings for their encouragement and support.

Contents

List of Tables

List of Figures

Chapter 1. Healthcare, Data, Analysis

1.1 Introduction

Healthcare is big business. In 2013, healthcare spending in the United States accounted for 17.1 percent of the gross domestic product, or $9,086 per capita [1]. Healthcare consumption in that year included an average of four physician visits and 2.2 regularly taken prescription drugs per US resident, as well as notably high use of diagnostic imaging technologies (such as MRI machines and CT and PET scanners). On the other hand, satisfaction with the healthcare system has been consistently low in the US as compared to other wealthy countries, and this is due more to perceived availability of top-quality care than to the most recent healthcare experience [2]. With so much money on the line, so many dissatisfied consumers, and human life and health hanging in the balance, it seems there is a great need for problem solvers in healthcare.

One idea for improving healthcare is to place more emphasis on *prevention* and less on *treatment*; the former is proactive while the latter is reactive. Not every health issue is preventable, but in many cases, early intervention can lead to better health outcomes and lower costs. One of the key elements of preventive healthcare is the disease screen. Through such screening, diseases can be diagnosed while still in the early, treatable stage. However, it is not feasible for everyone to get screened for every possible disease. A better solution is to have a cheap, scalable, and reliable means of measuring disease risk and predicting disease occurrence. Healthcare providers or insurers could then use this predictor to identify high-risk individuals and recommend tests or other interventions.

Machine learning algorithms built on routinely collected medical data have the potential to build just such a system. In this work, we discuss and test several computational methods to this end. As a case study, we focus on chronic diseases, especially Chronic Kidney Disease and Diabetes Mellitus, which we now review.

1.1.1 Chronic Diseases.

Chronic diseases are diseases that persist over a long period of time, e.g. above three months. Examples include heart disease, cancer, and respiratory conditions. Chronic diseases are a leading cause of death in high-income countries, and their treatment is connected to three fourths of US healthcare spending [3]. According to the Center for Disease Control and Prevention, however, these costly and prevalent diseases are also among the most preventable [4]. It is plausible, then, that early detection or prediction of chronic diseases can impact a great number of people and reduce the incidence and cost of these diesases.

Chronic Kidney Disease (CKD) is a chronic disease in which kidney function deteriorates, allowing blood waste to accumulate and damage the body. The disease progresses gradually, and people with the early stages often don't experience noticeable symptoms. However, if left untreated, the disease progresses to kidney failure, at which point the only treatment options are regular and costly dialysis, or kidney transplant [5]. CKD can be detected through blood and urine tests, but these are not routine for everyone. Given that more than 20 million US adults are estimated to have some stage of CKD, a great many people could potentially benefit from these diagnostic tests. A disease prediction algorithm could be of great help in identifying high-risk individuals.

Diabetes Mellitus (DM), or Type II Diabetes, is a chronic disease in which blood glucose levels are excessively high due to defective insulin production or action. DM can lead to a host of other health problems, including heart disease, hypertension, blindness, lower limb amputation, and kidney disease; it is in its own right a leading cause of death in the US. An estimated 25 million people in the US had diabetes in 2010, including 26.9% of people 65 years old and up, and a full seven million were estimated to be undiagnosed [6]. A disease prediction system for DM could help identify many of these undiagnosed individuals, allowing them to begin preventive measures such as blood pressure and glucose level control.

1.2 Medical Data

Healthcare has long been a data-rich field. With so many moving parts, healthcare providers and insurers have no shortage of variables to measure. The captured data have many important uses. They keep tabs on costs and billing. They track the activity of hospitals and outpatient facilities. Crucially, the data record the health states of people at a microscopic and macroscopic level. It is hard to overstate the importance of data in healthcare, especially when it comes to improving healthcare systems. Although in this work we focus on using medical data in disease prediction, there are many other facets of healthcare that can be enhanced and even revolutionized through intelligent use of data.

Obtaining access access to healthcare data is often a fraught endeavor. Privacy laws and business concerns set a host of hurdles that must be cleared before data can be shared. Unfortunately, this can halt the progress of researchers unaffiliated with insurance companies or hospital systems. Forging academic relationships with healthcare and insurance institutions is a vital step toward data access. The author of the present work learned this first hand, and his adviser all the more. Despite these difficulties, the potential rewards of better understanding and utilizing medical data to improve healthcare far outweigh the frustrations of data access.

1.2.1 Data Capture. There are various systems in place for capturing medical data. Modern healthcare systems use Electronic Health Record (EHR) tools for systematically and digitally storing a wide range of data, including patient demographics and medical history, lab results, physicals, radiology images, and more. EHR systems also facilitate data access and visualization, allowing doctors and patients to better inform themselves. Although these records only capture the activities that occur within a particular facility or set of facilities, they provide a vivid account of an individual's health state during his visits to the hospital and other care facilities.

Medical insurance claims form another rich repository of healthcare data. Claims data center on individuals enrolled in the insurance policy and their interactions with the healthcare system. These records typically include basic demographic information about patients, along with diagnoses, procedures, medication, inpatient and outpatient visits, and associated costs. Because insurance claims data are so patient-centric and capture patient healthcare activity across a variety of hospitals, facilities, and pharmacies, they paint a rather comprehensive picture of an individual's medical history and current health state. On the other hand, claims data often lack the fine-grained detail and rich clinical information found in EHRs.

We review a few different types of medical data found in EHRs and insurance claims records.

1.2.2 Simple Variables. Much of healthcare data consists of simple numerical and categorical variables. These include demographic variables such as age, sex, and ethnicity. Health variables such as height, weight, blood pressure, pulse, and many others are also straightforward. These types of simple data are suited for standard analytical and statistical methods (such as linear or logistic regression). To stop with just the simple variables, however, would be to miss out on potentially valuable insight provided by more complex sources of data.

1.2.3 Image Data. Image data in healthcare come primarily from radiology. Imaging technologies include X-ray, ultrasound, CT, PET, and MRI. These techniques produce images of internal tissues. The images are then typically examined by a human with expertise in visual diagnostics, and the results are passed on to the doctor and patient. Using image data for computational analysis is a nontrivial task. Developing algorithms that can extract meaningful semantics from digital images is an active and ongoing field of research within computer vision and machine learning. Techniques from these fields are required to fully tap the utility of medical imaging data.

1.2.4 Freetext Data. A large component of medical data is found in free text fields. Freetext data can include nurse or doctor notes regarding any number of areas surrounding a patient visit. They may consist of incomplete sentences, or detailed paragraphs; they may contain shorthand specific to a particular nurse, incoherent grammar, typos, and illegible scrawl. Despite their messy nature, freetext medical data can include vital information regarding patient health and experience with the healthcare system, and are therefore an important data source.

Although humans are adept at making sense of freetext, from a computational perspective, freetext data can be very difficult to use. Software built to utilize freetext may require sophisticated computer vision for recognizing the handwritten text and extensive natural language processing for extracting the meaning of the text. Although we do not deal with it in this work, the problem of utilizing freetext medical data in machine learning methods is an important one.

1.2.5 Coded Data. Many aspects of an individual's interactions with the healthcare system are captured in various types of coded data, which are ubiquitous in healthcare data records. Such coded data are naturally fit for computational processing, and are thus widely used, for example, by insurance software. For the same reason, these data are also well-suited for use in machine learning algorithms. A large focus of the present work is to understand how to best harness these data and unlock any predictive insight they may contain.

For all the convenience and abundance of coded data, it is important to consider their shortcomings. Coded data often give only a coarse approximation of reality, since they are based on human-defined categories. For example, an eye operation can be a very complex undertaking, with many potential pathways and outcomes. It is not a stretch to imagine that two different eye operations, even under very similar circumstances and with identical objectives, might differ substantially in efficacy and side-effects. However, a coding system for medical procedures might record these two operations with the same code, ignoring a wealth of nuance and detail.

In addition to the coarsening nature of coded data, another potential source of error resides in the coding process itself, which is performed by humans and is thus error-prone. Coders are faced with the task of converting the activities and outcomes of a healthcare event into a sequence of various types of codes. Each coder has her own level of competence and set of habits, and these can affect the encoding process in potentially important ways, given that coding systems often have thousands of possible codes and it is not always clear which code is best. Another pitfall is the phenomenon of *upcoding*, in which coders are incentivized to assign more – and more costly – codes, so as to drive up charges. This corrupts the data and could lead one to think someone is more sick than he actually is.

We now look more closely at three important strains of coded data.

1.2.6 Diagnosis Codes. Diagnosis codes record the diagnoses that a doctor makes when treating patients. For any given visit to the doctor, a patient may end up with multiple diagnosis codes if the doctor determines there are multiple health issues at play. Often, insurance records will indicate a primary diagnosis code and then possibly several secondary diagnosis codes. The primary diagnosis code corresponds to the principal reason for the doctor visit, and secondary codes record any other maladies that the doctor notes. For example, a patient may see a doctor about back pain and receive a primary diagnosis code indicating as much, but then may subsequently be diagnosed with high blood pressure. The high blood pressure diagnosis code will be among the secondary codes. When using diagnosis codes for disease prediction, it is therefore wise to utilize all available data, rather than simply the primary diagnosis codes.

The most common flavor of diagnosis codes is the International Classification of Diseases (ICD) [7]. This coding system has been around for decades. Up until recently, the 9th version of ICD, or ICD-9, has been the standard in the US healthcare system. (Technically, the system is known as ICD-9-CM, with “CM” indicating clinical modification, but for the sake of brevity, we stick with ICD-9.)

ICD-9 consists of around 13,000 diagnosis codes. The majority of these codes are numer-

ICD-9 Code	Description
491.x	Chronic bronchitis
491.0	Simple chronic bronchitis
491.20	Obstructive chronic bronchitis without exacerbation
491.22	Obstructive chronic bronchitis with acute bronchitis
492.x	Emphysema
E8261	Pedal cycle accident injuring pedal cyclist
V10.x	Personal history of malignant neoplasm

Table 1.1: Selection of ICD-9 diagnosis codes. The ‘x’ character indicates a wildcard that may be replaced with additional digits. Note the hierarchical nature of the numerical codes.

ical strings of four or five digits, arranged in a semi-heirarchical manner. For these codes, the first three digits determine a general diagnosis, and subsequent digits indicate more detailed information. Two codes sharing the first three digits, thus, are closely related. Additionally, the first three digits are arranged such that two codes whose first three digits are distinct but numerically close are often related.

A smaller portion of the ICD-9 diagnosis codes are alpha-numeric strings, beginning either with ‘E’ or ‘V’, and followed by numerical digits. The ‘E’-codes generally correspond to injuries caused by external factors, such as automobile accidents or explosions. The ’V’-codes are a bit more eclectic, and allow for coding of factors that influence health but aren’t related to the numerical codes or the ‘E’-codes. See Table 1.1 for a selection of ICD-9 diagnosis codes codes.

While ICD-9 diagnosis codes have long been the norm, there are updated versions in the works. Recently, a new iteration of ICD has become the standard, namely ICD-10. The ICD-10 diagnosis codes come to around 68,000 in number, thus allowing for very specific diagnoses. The degree of specificity sometimes borders on the comical. Consider the following codes:

- W61.33XA: Pecked by chicken, initial encounter.

- V91.07XA: Burn due to water-skis on fire, initial encounter.

- Z63.1: Problems in relationship with in-laws.

While there is sometimes a straightforward mapping from ICD-9 to ICD-10 codes, it is often quite unclear how to switch between systems. ICD-11 is currently under development, with a 2018 release scheduled. In this work, we operate exclusively in the ICD-9 system, since our datasets predate the widespread adoption of the ICD-10 standard.

In some situations, it is desirable to have a smaller diagnosis code set. While truncation to the first three digits of the ICD-9 diagnosis codes gives an immediate mapping to a smaller set, there are also expert-derived groupings of ICD-9 codes that can be used for this purpose. The Healthcare Cost and Utilization Project (HCUP), a collection of US healthcare data and software tools sponsored by the Agency for Healthcare Research and Quality, provides just such a partition. In particular, the HCUP Clinical Classifications Software (CCS) maps each diagnosis code to exactly one of 283 groups [8]. Each group contains diagnosis codes that fall into a clinically meaningful category. Examples of these categories include tuberculosis, cancer of the stomach, and acute bronchitis.

1.2.7 Procedure Codes.

Procedure codes indicate what medical procedures were performed during a healthcare interaction. These codes can include surgical procedures, physical therapy, diagnostic interventions, and other measures. While diagnosis codes generally record what ails the patient, procedure codes capture what is done to remedy the ailments. In insurance records, procedure codes are often linked directly to diagnosis codes, thus providing potentially useful context for the procedures.

There are various coding systems for medical procedures. The ICD-9 code set has a section devoted to procedure codes, known as Volume 3. ICD-9 procedure codes are strings of two to four numerical digits. As with the diagnosis codes, these procedure codes are organized in a semi-hierarchical manner, with the first two digits specifying a class of procedures, and subsequent digits giving further detail.

Another common coding system for medical procedures is the Healthcare Common Procedure Coding System, or HCPCS. This coding system is built on the Current Procedural Terminology (CPT) coding system of the American Medical Association, but includes additional levels of codes to capture a wide variety of medical procedures. HCPCS codes are generally alphanumeric strings of length five. Mapping between HCPCS and ICD-9 procedure codes can be a tricky task, as there does not exist a clear correspondence in many cases.

1.2.8 Drug Codes. Prescription drugs are another important source of coded data. When an individual goes to the pharmacy to fill a prescription, the transaction is recorded by the insurance provider, and the purchased drug must be coded. There are a few options for drug coding.

The most straightforward approach is to simply record the brand name of the drug. However, this method lacks any way to link very similar drugs from different brands. A remedy for this is given by the United States Adopted Name system, or USAN. USAN assigns a unique name to pharmaceuticals in the United States, based on the chemical contents of the drug. This removes the artificial distinction between differently-branded ibuprofen, for example. Unlike ICD-9 codes, there is no hierarchical structure in USAN names. Of course, these generic names must be constantly supplemented and updated by a governing committee as new drugs enter the market. Another coding system for drugs is the National Drug Code set, or NDC. Each NDC drug code has ten numeric digits organized into three segments. These segments encode information on the drug distributor, drug product, strength and form of dosage, and package size.

It is important to consider that coded drug data doesn't indicate whether an individual has actually taken the prescribed medication, only that a prescription was filled. If the data indicate that an individual refills a prescription consistently at regular intervals, it is reasonable to assume that he is, in fact, consuming the drugs. On the other hand, if the data show an isolated drug code, it is hard to know if the prescription was only meant to be filled once, of if the person simply failed to refill. Additionally, absence of coded drug data doesn't guarantee that no prescription was given. Given this uncertainty, using drug codes for disease prediction and other analytics may require careful thought.

1.3 Machine Learning in Healthcare

Machine learning is all about developing mathematical, computational, and statistical methodologies for finding patterns in and extracting insight from data. Data, in turn, are the concrete manifestations of structures and processes that shape the world. Machine learning research aims to unlock technologies that can solve hitherto intractable problems and transform human life in many different areas. Such has already been realized to spectacular effect many times over.

Healthcare is rife with rich data and difficult problems; it is therefore fertile ground for machine learning. Indeed, machine learning occupies an ever-growing role within healthcare. The 2015 NIPS conference in Montreal, Canada featured a popular and bustling workshop on machine learning in healthcare [9]. Intermountain Healthcare expanded its Population Health division in 2015 by hiring data scientists with statistical and mathematical skills. Computer science and healthcare journals alike are publishing papers on a range of machine learning techniques applied to healthcare problems, from topic modeling [10] [11] [12] and natural language processing [13], to Markov processes [14] and hidden Markov models [15] [16], and so on [17].

The aim of machine learning research in healthcare is not, of course, to replace human doctors or nurses, but rather to supplement and provide support where humans struggle. By doing precisely what human can't, namely processing huge amounts of data quickly, machine learning methods can both improve the quality and consistency of care on a large scale. Additionally, machine learning has promise in aiding more basic research in healthcare-related fields, such as automated drug discovery, genomics, and computational biology.

Despite the optimism of the above paragraphs, the success of machine learning in healthcare is not inevitable. Researchers must be careful not to get caught up in the buzzwords and hype, but stay grounded in the field, ask the right questions, and always critically examine what they do. We examine two important issues to keep in mind when practicing machine learning in healthcare.

1.3.1 Interpretability.

An important attribute of machine learning algorithms in the context of healthcare is interpretability. Many techniques in machine learning, although capable of accomplishing sophisticated tasks admirably, often operate as black boxes. They take inputs and compute outputs, but the mechanisms of this computation can be difficult to understand on anything beyond a technical level, and we don't always gain any real insight into the problem. With the stakes so high in healthcare, doctors and patients are understandably uncomfortable with basing decisions on predictions produced by inscrutable algorithms. Indeed, even if we are confident in the accuracy and robustness of a machine learning algorithm, blind reliance on its output can lead to unintended and unfavorable consequences. Furthermore, it can be difficult to debug and validate an algorithm based purely on its output, without understanding its inner workings and comparing them to proven methods.

So what constitutes an interpretable machine learning algorithm? It is difficult to precisely quantify the notion of interpretability. It certainly depends both on the algorithm at hand and the particular problem we are trying to solve. Generally speaking, however, we say that an algorithm or model is *interpretable* if there is a known way to draw conclusions from the model about the underlying problem that are deemed useful by some qualified expert.

Example 1.1 (Interpretable Probabilistic Models)**.** Probabilistic models are frequently interpretable, since their parameters can usually be tied to useful probability statements about the data. For example, logistic regression is a probabilistic classifier whose parameters indicate how each element of the input vector marginally affects the odds of the output. We will explore this topic further at a later point.

Example 1.2 (Proprietary Software)**.** Healthcare systems encounter uninterpretable black-box models when they license proprietary software that doesn't expose its source code. Even though the quality of a model can in some ways be assessed purely based on the accuracy of its output, the healthcare system using the software will have to ultimately rely on the word of the third party owner, and the chance of gaining useful insight is low.

1.3.2 Datasets. At the heart of machine learning techniques is the use of observed data to train a model. If the dataset used in the training process is flawed in some way, the resulting model will likely be just as flawed. A high quality training set is of utmost importance. We list a few items of particular note when it comes to healthcare datasets.

- *Imbalance.* When a certain important property (as determined by the problem at hand) is shared by only a very small proportion of the dataset, we say that the dataset is *imbalanced.* Healthcare datasets dealing with, for example, rare diseases, are often imbalanced. Many standard machine learning methods deteriorate when the data are too imbalanced; they tend to ignore the important, yet rare, property. Fortunately, there are often ways to work around these pitfalls (see, e.g., [18] [19]), but this is only possible when the researcher is aware of the imbalance problem in the first place.

- *Sample Size.* Despite the enormity of recorded healthcare data, access issues or other factors may require one to work with small datasets. In this case, one should take care that the machine learning method used doesn't overfit the data and pick up on spurious patterns or features. The interpretation and generalization of models trained on small amounts of data should be regarded with a healthy does of skepticism.

- *Realism.* In the typical machine learning scenario in healthcare, the researcher must prepare a dataset for herself using data that were not collected with her particular research objectives in mind. Although the original data reflect reality insofar as they were accurately recorded in real-world healthcare systems, the machine learning researcher should ensure that the data preprocessing steps don't introduce unrealistic artifacts or attributes into the dataset. Mistakes here include filling in missing values incorrectly and attempting to boost the size of the dataset by admitting data that don't quite fit in the with the task.

- *Generality.* Healthcare datasets tend to consist of geographically coherent individuals. Rural populations will likely differ in important ways from urban populations, and the health concerns of one city may well differ from those of another. It is therefore important to consider just how general the dataset is, as that will bound the generality of the machine learning model itself. In many cases, it is probably impossible to have a single model that performs well on a variety of different datasets from different areas. However, it is important to test the *methodology* for creating the model on several different healthcare datasets, as this can suggest whether the machine learning technique is widely useful.

Chapter 2. Classification

2.1 Introduction

The aim of *classification* is to assign labels to objects. More formally, assume there is a set $\mathcal{X}$ of objects and a finite set $\mathcal{Y}$ of labels. The game is to find a function that maps from $\mathcal{X}$ to $\mathcal{Y}$. *Binary classification* is the simplest form of classification, where $|\mathcal{Y}| = 2$. In this situation, we conventionally set $\mathcal{Y} = \{0, 1\}$. Binary classifiers are often the building blocks for classification problems in which there are many possible labels.

Example 2.1 (Facial Recognition)**.** A common – and often difficult – example of classification is the task of recognizing the identity of individuals in digital images. In this case, the set $\mathcal{X}$ consists of images, and $\mathcal{Y}$ is a set of names of people we wish to recognize. While it may not be too difficult for humans to accomplish this task, it is not immediately obvious how to computationally automate the process. Much research has gone into developing effective image classification algorithms, and nowadays the state-of-the-art is quite effective indeed.

Example 2.2 (Author Identification)**.** Identifying the authorship of text is another instance of classification. It may be much more difficult and expensive for humans to accomplish this task at large scale, but with a bit of clever text processing, algorithmic solutions are quite feasible. In this case, $\mathcal{X}$ is a set of texts (such as forum posts, audio speech transcriptions, or letters), and $\mathcal{Y}$ is a set of potential authors.

Example 2.3 (Disease Prediction)**.** Of great interest to healthcare providers, insurers, and this thesis, is the task of predicting which individuals will contract a particular disease in the future. An effective disease prediction algorithm can lead to early intervention and preventive care, which tend to improve patient outcomes and lower medical costs. For this binary classification task, $\mathcal{X}$ is a set of attributes of potentially at-risk individuals (such as age, sex, and health history), and $\mathcal{Y} = \{0, 1\}$, where 0 signifies no disease incidence, and 1 signifies disease.

It is useful to make a distinction between hard and soft classification.

Definition 2.4 (Hard and Soft Classification)**.** Let $\mathcal{X}$ be a set and let $\mathcal{Y} = \{0, 1\}$. A *hard classifier* is a function from $\mathcal{X}$ to $\mathcal{Y}$. A *soft classifier* is a function from $\mathcal{X}$ to $\mathbb{R}$. (Generalizing these concepts to non-binary classification is straightforward, but of limited interest to this thesis.)

Whereas a hard classifier fully commits to a label, a soft classifier can indicate degree of certainty. In particular, when a soft classifier takes values in the unit interval, we can often interpret its output as the probability that the label is 1.

We can obtain a hard classifier from a soft classifier by a simple thresholding operation. Let $f : \mathcal{X} \to \mathbb{R}$ be a soft classifier, and let $\tau \in \mathbb{R}$. We define the hard classifier $f_\tau : \mathcal{X} \to \mathcal{Y}$ by the formula

$$f_\tau(x) = \begin{cases} 0 & f(x) < \tau \\ 1 & f(x) \geq \tau \end{cases}$$

Using a soft classifier and this thresholding technique, we can vary the level of certainty required to label an object with 1. This type of flexibility is valuable when there are costs associated with the different labels.

Example 2.5 (Disease Prediction)**.** Suppose we have a soft classifier that takes values in the unit interval. This classifier has been created to predict if someone is at risk of developing late stage CKD; the label 0 indicates no risk, and the label 1 indicates high risk. Should an individual be flagged as high risk, the doctor may order potentially costly tests and treatments. Thus, it is important to assign the label 1 only to those individuals who we are fairly certain are high risk. In this case, we might set the threshold at 0.90 when assigning hard labels.

In the sequel, we assume that $\mathcal{X} = \mathbb{R}^m$ for some positive integer m.

2.2 Linear Classifiers

Linear classifiers are a widely used family of classifiers. They have the advantage of being easy to use, interpretable, and fairly effective for many problems. Before we can define what these are, we first need to understand the decision boundary of a classifier.

Definition 2.6 (Decision Boundary)**.** Let $f : \mathcal{X} \to \mathcal{Y}$ be a classifier. The *decision boundary* of f is the set

$$\bigcup_{y \in \mathcal{Y}} \operatorname{bd}(f^{-1}(y)).$$

We remark that in the case of binary classification, the decision boundary can simply be expressed as

$$\operatorname{bd}(f^{-1}(1)).$$

The decision boundary gives us a way to understand how the classifier carves up the set of objects $\mathcal{X}$ into classes. Depending on the classification problem at hand, we may require very complex or very simple decision boundaries. Generally, different types of classification algorithms will enable different kinds of decision boundaries. See Figure 2.1 for the level sets and decision boundaries of two classifiers.

For some classifiers, distance from the decision boundary corresponds to the degree of certainty of the classification. For example, given objects $x^{(1)}, x^{(2)} \in \mathcal{X}$, if $x^{(1)}$ is much closer than $x^{(2)}$ to the decision boundary, then we might be more confident in the label assigned to $x^{(2)}$ than in the label assigned to $x^{(1)}$.

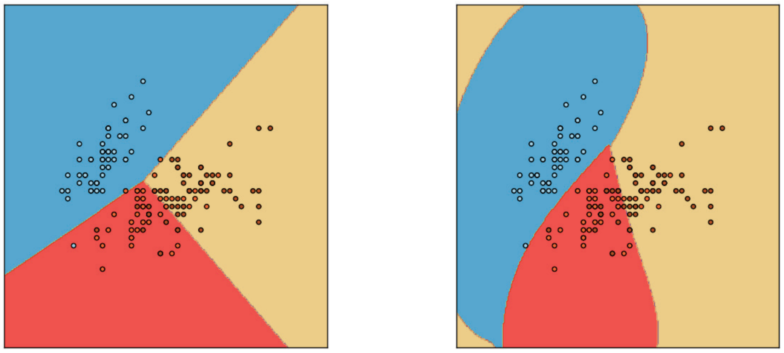


Figure 2.1: Decision boundaries and level sets for a classification problem with three labels.

We are now ready for the following definition.

Definition 2.7 (Linear Classifier)**.** Let $f : \mathcal{X} \to \mathcal{Y}$. We say that f is a *linear classifier* if the decision boundary of f is an affine hyperplane of $\mathcal{X}$.

A linear classifier is one of the simplest ways to classify objects. Essentially, it cuts $\mathcal{X}$ into two half-spaces, and assigns one label to each half-space. Observe that any hyperplane may be written as the level set of a linear function, i.e. has the form

$$\{x \in \mathcal{X} : \langle x, w \rangle = b\}$$

for some vector $w \in \mathcal{X}$ and scalar $b \in \mathbb{R}$ (here, $\langle \cdot, \cdot \rangle$ denotes the Euclidean inner product). We call this set the (w, b)-hyperplane.

If a linear classifier f has a decision boundary defined by the (w, b)-hyperplane, there is still the question of which label to assign to which half-space. Without loss of generality, we adopt the convention that $f(x) = 1$ if $\langle x, w \rangle \geq b$ and $f(x) = 0$ otherwise. Thus, there is a correspondence between linear classifiers and points of the form $(w, b) \in \mathcal{X} \times \mathbb{R}$. Indeed, it is apparent that each linear classifier can be obtained by zero-thresholding an affine function of the form

$$x \mapsto \langle x, w \rangle - b.$$

In this way, we see that affine functions from $\mathcal{X}$ to $\mathbb{R}$ are just soft classifiers which yield (hard) linear classifiers under thresholding. We generalize this notion slightly in the following definition.

Definition 2.8 (Soft Linear Classifier)**.** Let $f : \mathcal{X} \to \mathbb{R}$. We say that f is a *soft linear classifier* if there exist $w \in \mathcal{X}$, $b \in \mathbb{R}$, and some strictly monotonic function $\sigma : \mathbb{R} \to \mathbb{R}$ such that

$$f(x) = \sigma(\langle x, w \rangle - b).$$

We note that the level sets of a soft linear classifier are affine hyperplanes.

Example 2.9 (Logistic Regression)**.** One of the workhorses of binary classification is logistic regression. There are many different ways to motivate and formulate this particular classifier, but it is perhaps most simply understood as a soft linear classifier of the form

$$x \mapsto \sigma(\langle x, w \rangle - b),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the *logistic function* defined by the formula

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

Logistic regression also gives a method for selecting the parameters w and b given a set of labeled training data. More detail on this is given later.

2.3 Training

One of the central tenets of machine learning is to use data to make decisions. Concretely, machine learning techniques construct models or functions using observed data. This process is called *training*, and the dataset utilized is the *training set*. Training consists of selecting a model or function that best “fits” the training set.

When it comes to classification, the training set is a collection of labeled examples. That is, the training set is a subset of $\mathcal{X} \times \mathcal{Y}$. This situation is known as *supervised learning*, since the training data act as an oracle and "supervise" the training process. To formalize this, let $\mathcal{F}$ be a family of (possibly soft) classifiers, indexed by a parameter z that takes values in a *parameter set* $\mathcal{Z}$. For example, $\mathcal{F}$ may the the family of linear classifiers, and $z = (w, b) \in \mathcal{X} \times \mathbb{R}$. Let the training set be given by

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y} \,:\, 1 \le i \le n\}$$

for some $n \in \mathbb{N}$. Let $\mathrm{fit} : \mathcal{Z} \to \mathbb{R}$ measure the fit to the data; that is, $\mathrm{fit}(z)$ gives the fit between the training set $\mathcal{D}$ and the classifier defined by the parameter z. Then the training task boils down to the optimization problem

$$\underset{z \in \mathcal{Z}}{\mathrm{optimize}}\ \mathrm{fit}(z).$$

Whether the optimization is maximization or minimization depends on the interpretation of the fit function.

Coming up with a good measure of fit between a classifier and training set is of central importance. Below we review two common approaches, namely likelihood maximization and loss minimization.

2.3.1 Likelihood Maximization. Let $f : \mathcal{X} \to [0, 1]$ be a soft classifier. We say that f is a probabilistic classifier if there exists a $\mathcal{X}$-valued random vector X, a $\mathcal{Y}$-valued random variable Y, and a conditional probability distribution p such that

$$f(x) = p(Y = 1 \,|\, X = x).$$

Notationally, we use a lowercase p to denote either a probability mass function or density function, depending on whether the random variables in question are discrete or continuous. Since X takes values in $\mathcal{X} = \mathbb{R}^m$, we sometimes write $X = (X_1, \ldots, X_m)$.

Training a probabilistic classifier, then, amounts to choosing the conditional probability distribution p from some family indexed by a parameter z. We use the notation $p(\cdot\,;\,z)$ to denote dependence on this parameter. For a training set $\mathcal{D}$, the fit function is the conditional likelihood $L : \mathcal{Z} \to [0,1]$, given by

$$L(z) = \prod_{i=1}^{n} p(Y = y^{(i)} \mid X = x^{(i)}\,;\,z).$$

It is often convenient, both mathematically and numerically, to instead use the log conditional likelihood

$$l(z) = \log(L(z)) = \sum_{i=1}^{n} \log p(Y = y^{(i)} \mid X = x^{(i)}\,;\,z).$$

Probabilistic classifiers that are trained using the conditional (log) likelihood are sometimes called *discriminative models*.

Some probabilistic classifiers come with additional structure, namely a joint distribution for (Y, X). This is sometimes called a *generative model*. Since Y is binary, it is easy to obtain the marginal distribution of X, namely

$$p(X = x) = p(Y = 0, X = x) + p(Y = 1, X = x).$$

In this case, our classifier function can be computed as

$$f(x) = \frac{p(Y = 1, X = x)}{p(X = x)},$$

and we use the joint likelihood

$$L(z) = \prod_{i=1}^{n} p(Y = y^{(i)}, X = x^{(i)}\,;\,z).$$

Again, it is often more convenient to use the log likelihood.

Regardless of whether the joint or conditional likelihood is used, training involves maximizing the (log) likelihood function.

Example 2.10 (Naive Bayes Classifier)**.** The naive Bayes classifier is a generative model that defines a marginal distribution for Y and a conditional distribution for X given Y. The joint distribution is then recovered via the formula

$$p(Y, X) = p(Y)p(X \,|\, Y).$$

The "naive" part of this model comes from the assumption that the components of X are independent of each other when conditioned on Y, i.e.

$$p(X \,|\, Y) = \prod_{i=1}^{m} p(X_i \,|\, Y).$$

The "Bayes" part of the model comes from the use of Bayes theorem when computing the soft classifier function f:

$$f(x) = p(Y = 1 \,|\, X = x) = \frac{p(Y = 1)p(X = x \,|\, Y = 1)}{p(X = x)}.$$

The naive Bayes classifier is regarded as a very simple, yet often fairly effective solution to classification problems. However, it falls short on more difficult tasks.

Example 2.11 (Logistic Regression)**.** We return to the linear classifier known as logistic regression, and show how it can be viewed as a discriminative model. Define a conditional probability distribution with parameters $z = (w, b) \in \mathcal{X} \times \mathbb{R}$ by

$$p(Y = 1 \,|\, X = x\,;\, w, b) = \frac{1}{1 + e^{-\langle x, w\rangle + b}}.$$

Then the logistic regression classifier with parameters (w, b) is precisely

$$f(x\,;\, w, b) = p(Y = 1 \,|\, X = x\,;\, w, b).$$

Given a training set $\mathcal{D}$, we train logistic regression by maximizing the conditional log likelihood,

$$l(w, b) = \sum_{i=1}^{n} \log p(Y = y^{(i)} \,|\, X = x^{(i)} \,;\, w, b).$$

2.3.2 Loss Minimization. Loss minimization is another prominent paradigm for training classifiers. In this approach, we define a *loss function* (sometimes called cost function) $C : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$. Given a true label $y \in \mathcal{Y}$ and a predicted label $y' \in \mathbb{R}$, the quantity $C(y, y')$ measures the loss (or cost) incurred by the predicted y' in light of the true label y. Let $f(\cdot\,;\, z)$ be a hard or soft classifier with parameter z. The fit function for training is just the average loss over the training set $\mathcal{D}$, i.e.

$$\text{fit}(z) = \frac{1}{n} \sum_{i=1}^{n} C(y^{(i)}, f(x^{(i)} \,;\, z)),$$

and we seek to minimize this quantity. A more thorough discussion of loss minimization, grounded in statistical learning theory, is given in [20].

There are several common loss functions. We review some of them in the following set of examples. For notational convenience, we set $\mathcal{Y} = \{\pm 1\}$ for these examples.

Example 2.12 (0-1 Loss)**.** The 0-1 loss function is the most obvious choice for hard classifiers. It is simply defined by

$$C(y, y') = \begin{cases} 0 & y = y' \\ 1 & y \neq y' \end{cases}.$$

Observe that under this loss function, the average loss over the training set is just the proportion of the training set that the classifier incorrectly labels. Hence, by minimizing the average loss, we are simply maximizing the accuracy of the classifier on the training set.

Example 2.13 (Squared Loss)**.** Squared loss is appropriate for soft classifiers. It is defined by

$$C(y, y') = (y - y')^2.$$

Observe that this function coincides with 0-1 loss in the case of hard classifiers. Using a little algebra and the fact that if $y \in \mathcal{Y}$, then $y^2 = 1$, we have

$$\begin{aligned} C(y, y') &= (y - y')^2 \\ &= y^2(1 - yy')^2 \\ &= (1 - yy')^2 \\ &= \phi_s(yy'), \end{aligned}$$

where $\phi_s : \mathbb{R} \to \mathbb{R}$ is given by

$$\phi_s(t) = (1 - t)^2.$$

Example 2.14 (Hinge Loss)**.** Let $\phi_h : \mathbb{R} \to \mathbb{R}$ be given by

$$\phi_h(t) = \max(0, 1 - t).$$

Then hinge loss is defined as

$$C(y, y') = \phi_h(yy').$$

The name of this loss function comes from the shape of the graph of ϕ_h. Training a linear classifier using hinge loss is known as the linear support vector machine. Concretely, given the family of soft linear classifiers with parameters (w, b) defined by

$$f(x\,;\, w, b) = \langle x, w \rangle - b,$$

the linear support vector machine chooses the particular parameters (w^*, b^*) that minimize average hinge loss over the training set.

Example 2.15 (Logistic Loss)**.** Let $\phi_l : \mathbb{R} \to \mathbb{R}$ be given by

$$\phi_l(t) = \frac{1}{\log 2} \log(1 + e^{-t}).$$

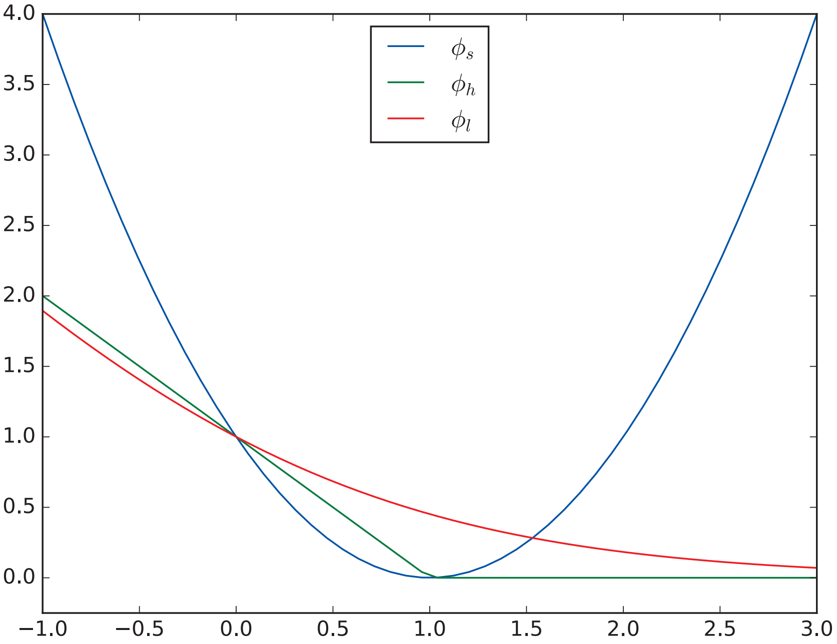


Figure 2.2: Plots of ϕ_s, ϕ_h, and ϕ_l as defined by squared, hinge, and logistic loss, respectively.

Then logistic loss is defined as

$$C(y, y') = \phi_l(yy').$$

Inspecting the graphs of ϕ_l and ϕ_h, we observe that ϕ_l is a smooth approximation of the piecewise-linear ϕ_h. See Figure 2.2 for a comparison of ϕ_s, ϕ_h, and ϕ_l. Training a linear classifier using logistic loss is known as logistic regression. Concretely, given the family of soft linear classifiers with parameters (w, b) defined by

$$f(x\,;\,w, b) = \langle x, w\rangle - b,$$

logistic regression chooses the particular parameters (w^*, b^*) that minimize average logistic loss over the training set. We will show later how this formulation is equivalent to our previous probabilistic formulation.

Example 2.16 (Cross Entropy)**.** For this example, we revert back to the convention that $\mathcal{Y} = \{0, 1\}$. Cross entropy is appropriate for probabilistic classifiers, i.e. classifiers that output values in the unit interval and are interpreted as giving the probability of the label 1. The loss function corresponding to cross entropy is

$$C(y, y') = -y \log y' - (1 - y) \log(1 - y').$$

This loss function is commonly used when training neural networks.

2.3.3 Solving the Optimization Problem. We have discussed how to formulate the training task as an optimization problem, but that is just the first step. Actually solving the optimization problem may range from very straightforward to wildly intractable, depending on the loss function or likelihood we choose. Finding better and faster ways of solving (or approximately solving) these optimization problems is an active area of research. In this work, we largely take it for granted that we can numerically perform the required optimization, but the reader should not infer that this is a solved problem in all cases.

2.4 Testing

In training, we do our best to construct a classifier that fits a training set of labeled data as well as possible. However, our ultimate goal is to have a classifier that performs well on *all* appropriate data, not just the particular examples in the training set. We use the term *generalization* to denote the performance of a classifier on new, unseen data. Since the effectiveness of a classifier resides in its ability to make useful predictions even on unseen examples, it is important to investigate its generalization.

We measure the generalization by gathering a new collection of labeled data, which we call the *test set*, and reporting possibly multiple numerical quantities that evaluate how well the classifier was able to predict the labels in the test set. What are effective performance metrics? There are numerous possibilities (see, for example, [21]). We touch on a couple examples below. Throughout these examples, we assume a test set $\{(x^{(i)}, y^{(i)} \in \mathcal{X} \times \mathcal{Y} : 1 \leq i \leq k\}$.

2.4.1 Accuracy. For a hard classifier f, the most obvious measure of performance is *accuracy*, which is given by the expression

$$ACC = \frac{\#\{i : f(x^{(i)}) = y^{(i)}\}}{k}.$$

Accuracy, of course, is just the percentage of correct predictions in the test set. It can range in value from zero to one, and higher values indicate better performance.

While easily understood, this measure suffers from a couple of drawbacks. First, it treats the two labels symmetrically, when in real life, predicting one class correctly (or avoiding false predictions of that class) may be much more important than with the other class. Accuracy also comes up short when dealing with imbalanced data, in which one class is much more prevalent than the other. When data are highly imbalanced, a "dumb" classifier that just labels everything with the most prevalent label will have high accuracy, but nevertheless will fail completely in classifying the rare class.

2.4.2 ROC and AUC. For soft classifiers, a popular performance metric is given by the *receiver operating characteristic* (ROC) curve and the *area under the curve* (AUC). The ROC curve gives a visual depiction of performance, while the AUC gives a succinct numerical measure.

Before getting into these, we first need to define the *true positive rate* (TPR) of a hard classifier f as

$$TPR = \frac{\#\{i : f(x^{(i)}) = 1 = y^{(i)}\}}{\#\{i : y^{(i)} = 1\}}$$

and the *false positive rate* (FPR) as

$$FPR = \frac{\#\{i : f(x^{(i)}) = 1 \neq y^{(i)}\}}{\#\{i : y^{(i)} \neq 1\}}.$$

Now let f be a soft classifier, and let τ be a threshold value. For each $\tau \in \mathbb{R}$, the thresholded hard classifier f_τ has true positive rate $TPR(\tau)$ and false positive rate $FPR(\tau)$.

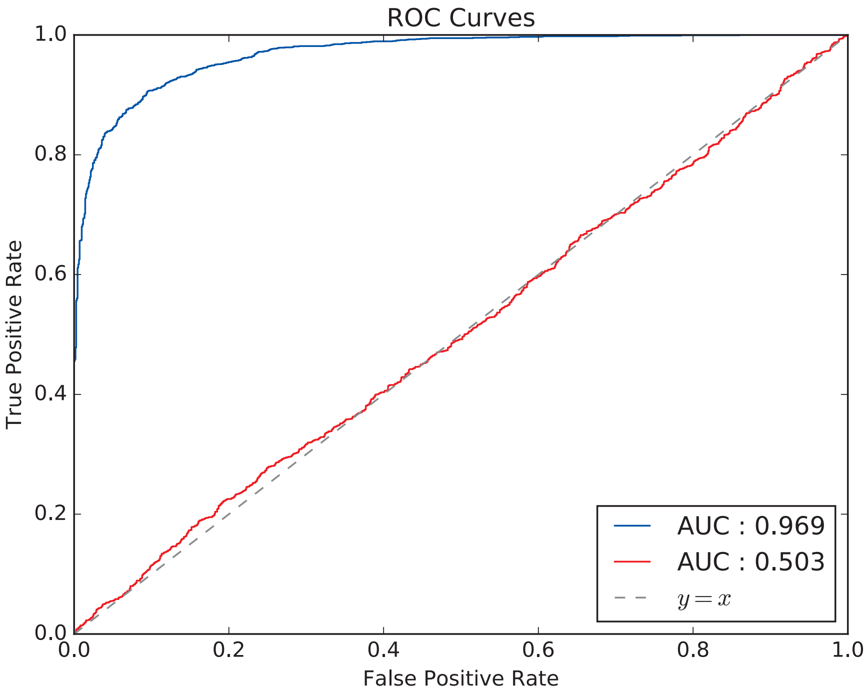


Figure 2.3: ROC curves and AUC scores for a good classifier (blue) and a poor classifier (red).

The ROC curve is the parameterized curve

$$\tau \mapsto (FPR(\tau), TPR(\tau)).$$

Of course, since the test set is finite, the image of the ROC curve consists of finitely many points. When plotting the ROC curve, it is conventional to linearly interpolate between neighboring points under the dictionary ordering.

Note that ROC curves are contained in the unit square $[0,1] \times [0,1]$. Generally speaking, the ROC curve of a good classifier will have a steep positive slope near the origin and will approach the ceiling of the unit square quickly. The ROC curve of a poor classifier will have a gentler slope and lie closer to the line $y = x$. See Figure 2.3 for a depiction of these cases.

The AUC is area bounded by the bottom of the unit square and the ROC curve. An AUC score closer to one indicates good performance, while an AUC closer to 0.50 indicates poor performance. The AUC can be interpreted as (an estimate of) the probability that

$f(x) > f(x')$, where $x \in \mathcal{X}$ is a randomly chosen example with label 1 and $x' \in \mathcal{X}$ is a randomly chosen example with label 0.

2.5 Regularization

A common problem in machine learning generally, and in classifier training in particular, is that of *overfitting*. We say that a classifier is overfit to the training data if it fits the training set well but has poor generalization. Ideally, the training dataset is sufficiently large and representative so as to ensure that any classifier trained on it will generalize well. These conditions are frequently not met, however, and even in the best of times, overfitting can be a real problem.

One major cause of overfitting is a training set that is too small or noisy. In this situation, random noise that might otherwise cancel out in larger datasets is at greater threat to influence the likelihood or cost function in training. When this happens, the trained classifier will be biased by the chance noise in the training set and may generalize poorly. Because of this issue, it is vital to obtain training data that is both plentiful and as representative as possible.

Another root of overfitting is when the complexity of the indexed family of classifiers used in training exceeds the data complexity of the training set. A classic example, albeit not related to classification, is the problem of fitting a curve to a set of points. Suppose the training set consists of several points that nearly lie on a line. If the person trying to fit a curve is unaware that the training set has low data complexity, and fits a high-degree polynomial to the points, she will have a curve that fits the training data perfectly, but generalizes horribly, since it differs vastly from the line on which the points nearly lie.

Regularization refers to any attempt to curb overfitting by altering the training process. One common form of regularization is to introduce some kind of penalty term into the training objective function which discourages the parameters from growing too much. More specifically, let z be the parameter vector, and assume without loss of generality that the training problem is given by

$$\min_{z \in \mathcal{Z}} \operatorname{fit}(z).$$

It is often customary to include either a weighted L_2 penalty

$$\min_{z \in \mathcal{Z}} \operatorname{fit}(z) + C\|z\|_2^2$$

or a weighted L_1 penalty

$$\min_{z \in \mathcal{Z}} \operatorname{fit}(z) + C\|z\|_1.$$

The weight C determines how strong the regularization. The L_2 penalty generally prevents any entries of z from growing too large, while the L_1 penalty has the effect of encouraging sparsity, i.e. favoring parameter vectors that have many entries equal to zero.

Another regularization strategy is to prematurely terminate the training procedure. This can be done, for example, by halting the optimization before it converges, or by shrinking the set of allowable parameters over which to optimize. The motivation for this technique is the notion that the longer a classifier is trained, the more complex it becomes. Thus, early stopping can prevent excessive complexity. A separate set of labeled data called a *validation set* can be handy when deciding when to halt training. The idea is to evaluate the partially-trained classifier on the validation set every so often, and terminate training once the performance on the validation set ceases to improve.

Regularization is often more of an art than a science. Choosing which type of regularization, what weight value, and so forth, is frequently informed more by empirical observation than theory. Nonetheless, there is interesting and sometimes enlightening theoretical grounding for the concepts of overfitting, model and data complexity, and the regularization techniques [22] [23].

2.6 Logistic Regression

We have encountered logistic regression (LR) several times already, both in the context of likelihood maximization and loss minimization. In this section, we tie together the previous material and expand the discussion.

Recall that logistic regression is a soft linear classifier with parameters $(w, b) \in \mathcal{X} \times \mathbb{R}$. Its usual formulation is as a discriminative model, with its classification function given by

$$f(x) = p(Y = 1 \,|\, X = x \,;\, w, b) = \sigma(\langle x, w \rangle - b),$$

where σ is the logistic function as defined previously. Training is done by maximizing the conditional log likelihood of the training set $\mathcal{D} = \{(x^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y} : 1 \leq i \leq n\}$. That is, in training we solve the problem

$$\max_{(w,b) \in \mathcal{X} \times \mathbb{R}} \sum_{i=1}^{n} \log p(Y = y^{(i)} \,|\, X = x^{(i)} \,;\, w, b),$$

with the possible addition of a regularization term to the objective function. See Figure 2.4 for a visual example.

How does this relate to the loss minimization formulation? If we adopt the notation $\mathcal{Y} = \{\pm 1\}$, we observe that for any $x \in \mathcal{X}$,

$$p(Y = 1 \,|\, X = x \,;\, w, b) = \frac{1}{1 + e^{-(\langle x, w \rangle - b)}}$$

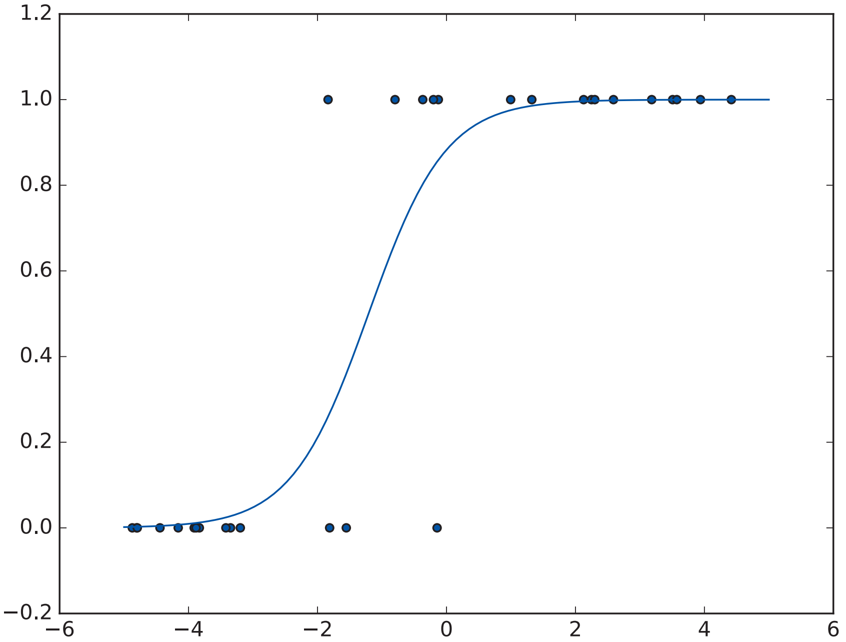


Figure 2.4: A logistic regression classifier together with the data on which it was trained.

and

$$\begin{aligned}
p(Y=-1 \,|\, X=x\,;\, w,b) &= 1 - p(Y=1 \,|\, X=x\,;\, w,b)\\
&= 1 - \frac{1}{1+e^{-(\langle x,w\rangle - b)}}\\
&= \frac{1+e^{-(\langle x,w\rangle - b)} - 1}{1+e^{-(\langle x,w\rangle - b)}}\\
&= \frac{1}{1+e^{\langle x,w\rangle - b}}.
\end{aligned}$$

Hence, for any $(y,x) \in \mathcal{Y} \times \mathcal{X}$, we in fact have

$$p(Y=y \,|\, X=x\,;\, w,b) = \frac{1}{1+e^{-y(\langle x,w\rangle - b)}}.$$

With this in mind, we see that we can rewrite our conditional log likelihood as follows:

$$\begin{aligned}\sum_{i=1}^{n} \log p(Y = y^{(i)} \,|\, X = x^{(i)}\,;\, w, b) &= \sum_{i=1}^{n} \log \frac{1}{1 + e^{-y^{(i)}(\langle x^{(i)}, w\rangle - b)}} \\ &= -\sum_{i=1}^{n} \log(1 + e^{-y^{(i)}(\langle x^{(i)}, w\rangle - b)}) \\ &= -\log 2 \sum_{i=1}^{n} \phi_l(y^{(i)}(\langle x^{(i)}, w\rangle - b)),\end{aligned}$$

where ϕ_l is as defined previously. Since the loss minimization problem

$$\min_{(w,b)\in\mathcal{X}\times\mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \phi_l(y^{(i)}(\langle x^{(i)}, w\rangle - b))$$

is equivalent to the likelihood maximization problem

$$\max_{(w,b)\in\mathcal{X}\times\mathbb{R}} \quad -\log 2 \sum_{i=1}^{n} \phi_l(y^{(i)}(\langle x^{(i)}, w\rangle - b)),$$

we conclude that the two formulations of logistic regression are likewise equivalent.

The probabilistic formulation of logistic regression affords a nice interpretation of the parameters w and b. To get at this interpretation, first consider the *odds* of an object $x \in \mathcal{X}$ having the label 1:

$$\begin{aligned}\mathrm{Odds}(Y = 1 \,|\, X = x) &= \frac{p(Y = 1 \,|\, X = x\,;\, w, b)}{p(Y = -1 \,|\, X = x\,;\, w, b)} \\ &= \frac{1 + e^{\langle x, w\rangle - b}}{1 + e^{-(\langle x, w\rangle - b)}} \\ &= e^{\langle x, w\rangle - b} \frac{1 + e^{-(\langle x, w\rangle - b)}}{1 + e^{-(\langle x, w\rangle - b)}} \\ &= e^{\langle x, w\rangle - b}.\end{aligned}$$

Note that no matter the value of x, the odds of the label 1 always has the multiplicative factor e^{-b}. Because of this, we call b the *bias*. A large positive bias decreases the odds of the label 1, and a large negative bias increases the odds of the same. Each entry of the parameter

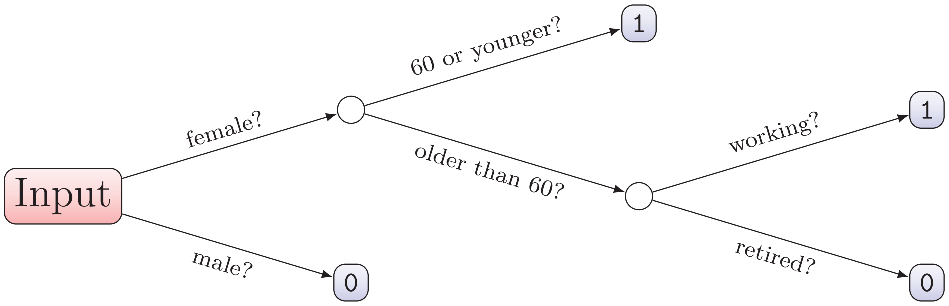


Figure 2.5: A simple decision tree.

vector $w = (w_1, \dots, w_m)$ also influences the odds of the label 1. Specifically, the quantity e^{w_i} gives the multiplicative change in the odds given a marginal unit increase in the i-coordinate of x. To see why, let $x = (x_1, \dots, x_i, \dots, x_n)$ and let $x' = (x_1, \dots, x_i + 1, \dots, x_n)$. Observe that

$$\begin{aligned}
\text{Odds}(Y = 1 \,|\, X = x') &= e^{\langle x', w\rangle - b} \\
&= e^{w_i + \langle x, w\rangle - b} \\
&= e^{w_i}\text{Odds}(Y = 1 \,|\, X = x).
\end{aligned}$$

If w_i is large in absolute value, this indicates that the feature x_i has a strong influence on the label. The sign of w_i determines whether this influence is toward the positive or negative label.

2.7 Decision Trees and Random Forests

A decision tree classifier computes its output using a tree structure in which each branch point is associated with a binary condition on the input data, and each leaf is associated with a classification output. An input datapoint starts at the root of the tree, and traverses the tree according to the binary conditions until it reaches a leaf, at which point it outputs the specified label (or soft prediction). See Figure 2.5 for a toy example.

Decision trees are generally trained in a loss-minimization setting with a greedy algorithm that constructs each binary condition in an iterative manner. Because of the transparent way in which a decision tree computes its output, decision tree classifiers are readily interpretable. They are also computationally cheap to use at test time. Unfortunately, they are known to be susceptible to overfitting, and can struggle on difficult classification tasks. This can be partially offset by pruning the tree after training, or by early stopping. The conditional inference tree was developed to address these issues [24].

A random forest, or RF, is a soft classifier that is built out of several individual decision tree classifiers. This approach is based on the idea of *boosting*, or combining the efforts of an ensemble of potentially weak classifiers to produce a single, stronger classifier. The RF accomplishes this by training a specified number of decision tree classifiers using two crucial randomizing operations:

- each tree is trained on a random bootstrap sample of the available training data, and
- each tree only uses a randomly selected subset of the features of the data.

The trained decision trees then produce a single prediction by averaging the individual votes.

RF classifiers have better performance than individual decision trees, can be trained quickly using parallel computing, and have proven to be a strong baseline model for many classification tasks. However, the simple interpretability of decision trees does not carry over to random forests. Instead, there is a measure of variable importance that indicates roughly which features of the input data have the largest influence on predictive performance of the classifier. Further details are provided in [25].

Various extensions and variations on the idea of boosting decision tree classifiers have been proposed, such as extremely randomized trees [26] and gradient boosting [27]. These provide additional options for strong baseline models.

Chapter 3. Sequence Data

3.1 Introduction

Sequence data are defined by two characteristics:

- they are represented as sequences of elements, and
- the order of these elements is critical.

Symbolic sequence data are sequence data in which the elements of each sequence come from a discrete set of symbols (such as letters or integers). *Temporal sequence data* are sequence data in which the elements of each sequence are paired with time stamps, and the ordering is determined by time. *Time series* are temporal sequence data in which the time stamps occur at regular intervals over a period of time.

Sequence data are pervasive. We review a few examples below.

Example 3.1 (Natural Language)**.** Language is probably the most important way that we store and transmit information, whether through speech or writing. With the ubiquity of electronic word processing and the digitization of historical documents, natural language data are computer-accessible as never before, and hence ripe for machine learning. Natural language can be regarded as symbolic sequence data, but is usually not temporal. The symbol set consists of words, and the ordering is crucial.

Example 3.2 (Computational Genomics)**.** As biotechnologies have enabled us to sequence DNA, computational and algorithmic approaches to understanding genetics have become increasingly important. The field of computational genomics is based on analyzing DNA and RNA data, which are symbolic sequence data consisting of sequences of nucleic acids (often encoded as letters: A,T,C,G,U). As with natural language, there is generally no temporal aspect to this kind of data.

Example 3.3 (Diagnosis Codes)**.** Medical diagnosis codes provide a glimpse into an individual's health history. A dataset formed by collecting the diagnosis codes given to each person in a population over a period of time is an example of temporal symbolic sequence data. The symbol set is the set of codes (such as the ICD-9 diagnosis codes), and the time stamps are the date and time at which the codes were given. Since people visit the doctor irregularly, diagnosis code sequence data are not time series data.

Example 3.4 (Stock Market)**.** Stock market data consist of daily prices for a given stock or collection of stocks. Such data play a central role, of course, in financial mathematics and on Wall Street. Since stock prices are measured each day, these data can be regarded as time series data.

Unexample 3.5 (Demographic Data)**.** Suppose we have a dataset consisting of the age, sex, ethnicity, and yearly income for a population of adults. While each point in this dataset could be regarded as a sequence of elements, namely the four attributes given above, there is no important notion of ordering to these elements. This type of data, therefore, can't be regarded as sequence data.

In the sequel, we focus primarily on symbolic sequence data.

3.2 Data Representation

In order to use sequence data for machine learning tasks like classification, we need to find a suitable way to represent it. While some models and algorithms are purpose-built to handle sequence data, others only take fixed-length numerical vector input. In this section we present a few different possibilities for symbolic sequence data representation.

For notational convenience, we assume that the elements of the sequence data are taken from a symbol set that we enumerate as $\mathcal{S} = \{s_1, s_2, \ldots, s_p\}$.

3.2.1 One-hot Encoding. It is often necessary to convert symbolic data to numeric data. The most immediately obvious approach to this might be to simply use the enumeration of the symbols, i.e. map s_i to i for $1 \leq i \leq p$. However, this imposes an arbitrary numerical ordering of the symbols, which might lead to artificially regarding certain symbols as more important than others (for example, s_p might be weighted more that s_1 due simply to the fact that $p > 1$).

One-hot encoding offers an alternative approach that avoids such artificial distinctions. In particular, one-hot encoding is a map $e : \mathcal{S} \to \mathbb{R}^p$ defined by

$$e(s_i) = e_i,$$

where e_i is the i-th standard basis vector of $\mathbb{R}^p$, the vector of all zeros except for the i-coordinate, which is 1. One-hot encoding thus gives a sparse – and, if the symbol set is large, a high-dimensional – numerical encoding of the symbols.

Example 3.6 (DNA Sequences)**.** DNA can be modeled as a sequence of four possible nucleotides, often encoded with the letters A,T,C, and G. We can enumerate our symbol set as $\mathcal{S} = \{s_1 = \text{A}, s_2 = \text{T}, s_3 = \text{C}, s_4 = \text{G}\}$. The one-hot encoding of the sequence AATCTG, then, is

$$\left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right).$$

3.2.2 Bag-of-words. The bag-of-words (BOW) representation of symbolic sequence data maps each sequence (which can be of arbitrary finite length) to a fixed-length numeric vector. This is advantageous, since these fixed-length vector representations can then be input into many conventional models like logistic regression or random forest.

The BOW encoding of a sequence w is the vector whose i-th coordinate gives the number of times that symbol s_i occurs in w. This can be realized as the sum of the one-hot encodings of the elements of w.

Example 3.7 (DNA Sequences)**.** Continuing the previous example, the BOW representation of the sequence AATCTG is just

$$\begin{bmatrix} 2 \\ 2 \\ 1 \\ 1 \end{bmatrix}.$$

Note that the sequential ordering is completely lost. Thus, the BOW representation precludes the possibility of gleaning any predictive insight or patterns present in the sequential structure of the data.

3.2.3 Dimensionality Reduction. When the symbol set is large, i.e. when p is a large integer, both the BOW representation and one-hot encoding are high-dimensional. When any individual sequence only contains a relatively small number of the possible symbols, these representations are also sparse. Sparse, high-dimensional data are often very *compressible*, i.e. it is often possible to find a map from $\mathbb{R}^p$ to $\mathbb{R}^q$, with $q \ll p$, that preserves the structure of the data in some appropriate sense. The process of finding such a map is called *dimensionality reduction*. While a large field in its own right, here we explore just a couple of possibilities for dimensionality reduction.

One possibility is to map the symbol set $\mathcal{S}$ directly to a smaller symbol set. This is feasible when there is a natural partition of $\mathcal{S}$ into groups of related symbols. For example, if $\mathcal{S}$ is the set of ICD-9 diagnosis codes, we could truncate each code to its first three digits, or apply the CCS grouping.

Another option is to use the popular topic model latent Dirichlet allocation (LDA). This model was originally developed and is most commonly used for understanding the structure of natural language text by discovering a set of topics prevalent in a collection of documents, and inferring the prominence of each topic in a given document [28]. Despite its associations with natural language processing, LDA can be used on any type of symbolic sequence data and relies on the BOW representation.

In LDA, a *topic* is a discrete distribution on the set of symbols $\mathcal{S}$, and a *topic distribution* is a discrete distribution on the set of topics. When the set of topics is enumerated, each topic distribution can be realized in the obvious way as a stochastic vector in $\mathbb{R}^q$, where q is the number of topics. LDA uses a training set of symbolic sequences to construct a specified number of topics. Once trained, LDA can determine the topic distribution of any symbolic sequence. Thus, given a trained LDA model with q enumerated topics, we have a map from $\mathbb{R}^p$ to $\mathbb{R}^q$ that sends a BOW sequence representation to its topic distribution.

Example 3.8 (Diagnosis Codes)**.** The collection of diagnosis codes given to an individual over a period of time can be viewed as a sort of document describing the health state during that time. By training LDA on the diagnosis code sequences for an entire population, we obtain not only a dimensionality-reduction map, but also a set of topics that are often clinically coherent. These topics provide an interpretation for the lower-dimensional topic distribution vectors. See Table 3.1.

3.3 Sequence Learning

Machine learning techniques for dealing with sequence data have enjoyed active attention from researchers for several decades. For a taste of some of this research, see for example [29] and [30]. Sequence data may be used in conventional tasks such as classification or clustering, but there are also learning tasks that are specific to sequence data, such as sequence labeling or segmentation.

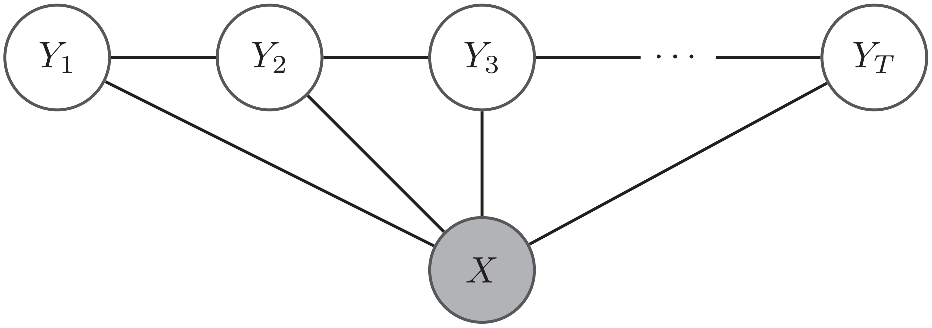
Sequence data are often challenging to fully exploit in the machine learning setting. Reasons for this include variability in sequence length, irregular sampling frequency in temporal sequence data, sparsely occuring symbols, and potentially subtle yet critical longe-range dependencies between elements in the sequences. In this section, we review two machine learning models for sequence data that have overcome these challenges on various tasks.

Topic 1	**Topic 16**
ULCER OF OTHER PART FOOT	ASTHMA UNSPECIFIED
NEUROPATHY IN DIABETES	ACUTE SINUSITIS NOS
DM NEURO TYPE II CNTRL	COUGH
FIT/ADJ VASCULAR CATH	ACUTE URI NOS
LONG TERM USE ANTIBIOTIC	ACUTE BRONCHITIS
DM W/MANIF TYP II CNTRL	ACUTE PHARYNGITIS
HYPERTENSION NOS	CHRONIC SINUSITIS NOS
EDEMA	HEADACHE
PAIN IN LIMB	AC MAXILLARY SINUSITIS
CELLULITIS OF LEG	ESOPHAGEAL REFLUX
Topic 19	**Topic 30**
ANXIETY STATE NOS	LUMBAGO
DEPRESSIVE DISORDER NEC	LUMBOSACRAL NEURITIS NOS
GENERALIZED ANXIETY DIS	LUMB/LUMBOSAC DISC DEGEN
LONGTERM USE OTH MED	LUMBOSACRAL SPONDYLOSIS
RHEUMATOID ARTHRITIS	LUMBAR DISC DISPLACEMENT
ATTN DEFICIT W HYPERACT	BACKACHE NOS
RECURR DEPR PSYCHOS-MOD	STENOS LUMB W/O NEU CLAU
INSOMNIA, UNSP	PHYSICAL THERAPY NEC
ATTN DEFIC NONHYPERACT	JOINT PAIN-PELVIS
ALCOHOL ABUSE-UNSPEC	POSTLAMINECT SYND-LUMBAR

Table 3.1: ICD-9 code descriptions associated with a selection of topics recovered by a LDA topic model.

3.3.1 Conditional Random Fields. A conditional random field (CRF) is a probabilistic model that defines a conditional probability model via a graph formalism. In particular, given random vectors X and Y, along with an undirected graph G whose vertices are in one-to-one correspondence with the elements of Y, a conditional random field is a distribution of $Y \mid X$ that factorizes according to the graph G in the sense of undirected graphical models (also known as Markov random fields). The vector X is understood to be observed input data, and Y the set of variables to be predicted. Additional detail about undirected graphical models can be found in [31]. CRFs have been successfully applied to problems in text and image processing and bioinformatics, among other fields. See [32] for an introduction to CRFs and a review of their applications.

Where sequential data are concerned, the most used CRF structure is the *linear-chain* CRF. In a linear-chain CRF, the graph G is a linear chain, i.e. a connected tree where each



$$X = (X_1, X_2, X_3, \ldots, X_T)$$

Figure 3.1: Graphical representation of a linear-chain CRF.

node has degree at most 2. The input variable X is generally taken to be a sequence, while the target variable Y shares the linear chain structure of G, and usually aligns with the sequence X. The graphical representation of such a model is given in Figure 3.1. When this is the case, we can write $X = (X_1, X_2, \ldots, X_T)$, $Y = (Y_1, Y_2, \ldots, Y_T)$, and

$$p(Y \mid X) = \frac{1}{Z} \prod_{i=1}^{T} \exp\left(\sum_{j=1}^{K} w_j f_j(Y_i, Y_{i-1}, X, i) \right).$$

Here, $w_j \in \mathbb{R}$, f_j is a positive real-valued function for $1 \leq j \leq K$, and Z is a the normalization constant (which depends on X) that ensures the above expression is truly a probability distribution. The modeler must define the functions f_j, otherwise called the *feature functions*, and the model itself learns the parameters w_j during the training process. We observe that T, the length of the sequence, can change to conform to the length of any input sequence, as the number of parameters is constant with respect to T.

The most crucial part of using a linear-chain CRF for sequence learning is constructing effective feature functions. This can be quite difficult for the unexperienced, since from the model definition it is not immediately clear how the feature functions even influence the probability distribution. One common approach to defining feature functions is known as *windowing*, which we describe in the case that X and Y are symbolic sequences with symbol sets $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. First, specify a nonnegative integer ω, the window size. For each $1 \leq i \leq T$, define the *window of size ω centered at i* to be the subsequence

$$X_{i-\omega\,:\,i+\omega} = (X_{i-\omega}, X_{i-\omega+1}, \ldots, X_{i+\omega}).$$

Next, enumerate each element in the product set $\mathcal{S}_2^2 \times \mathcal{S}_1^{1+2\omega}$, and notate this enumeration with $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$, which we call the set of *window configurations*. Finally, define the feature function f_j for $1 \leq j \leq K$ by the formula

$$f_j(Y_i, Y_{i-1}, X, i) = \begin{cases} 1 & (Y_i, Y_{i-1}, X_{i-\omega\,:\,i+\omega}) = c_j \\ 0 & \text{otherwise} \end{cases}.$$

Essentially, the feature function f_j acts as a detector for configuration c_j, returning 1 if that configuration is present at a given point in the sequence data, and 0 otherwise. The associated weight w_j, which is learned by the model during training, indicates the extent to which this configuration accounts for the structure of the data in the training set. With a bit of care, these learned weights can afford an interpretation of the model. Other types of feature functions can, of course, be added in, including features that detect long-range configurations rather than just the local window configurations. The flexibility of these feature functions account for the linear-chain CRF outperforming the more conventional hidden Markov model (HMM) in several sequence learning tasks.

We can use a linear-chain CRF for sequence classification in the following manner. For each labeled sequence pair $(X = (x_1, x_2, \ldots, x_T), y)$ in the training set, build a corresponding target sequence $Y = (y_1, y_2, \ldots, y_T)$ with $y_i = y$ for $1 \leq i \leq T$ (so Y is a sequence of all ones or all zeros, for example). Construct appropriate feature functions (such as window features described above). Train the linear-chain CRF on the input-target sequence pairs in the training set. For any new input sequence $X = (x_1, x_2, \ldots, x_T)$, the resulting soft classifier is just the map

$$X \mapsto p(y_T = 1 \mid X).$$

(This marginal probability can be computed from the trained CRF model using a message-passing algorithm.)

Another variant of the CRF, called the *hidden conditional random field*, or HCRF, expands and improves on this approach to sequence classification much in the same way that the HMM expands on the vanilla Markov model. See [33] for additional details.

3.3.2 Recurrent Neural Networks. Neural networks have become very popular in machine learning research of late, due largely to their often superior performance compared to other algorithms in various data competitions and in the literature. The neural network paradigm allows researchers to build flexible and complex models, often without needing to go through the hard work of defining domain-specific features (as is necessary, for example, with CRFs). Recent advances in neural network specific optimization techniques as well as software and hardware support have allowed non-experts to build and train models with relative ease and speed. On the flip side, neural network models often amount to black box techniques, failing to yield insight into the problem at hand. Interpretability is not a strength of neural networks.

Recurrent neural networks (RNNs) are neural network architectures that have a recurrent structure which allows for processing of arbitrary sequence input. The recurrent structure is basically a feedback loop: the internal state and output of a recurrent layer at position i in the input sequence X is a function of the current input X_i and the internal state of the recurrent layer at position $i - 1$. Of course, one can stack multiple recurrent layers and combine these with other types of neural network layers to build deep and complex models. See Figure 3.2 for a graphical representation.

The internal state of a recurrent layer includes any internal variables that are not passed to subsequent layers in the network. State variables can be thought of as memory cells:

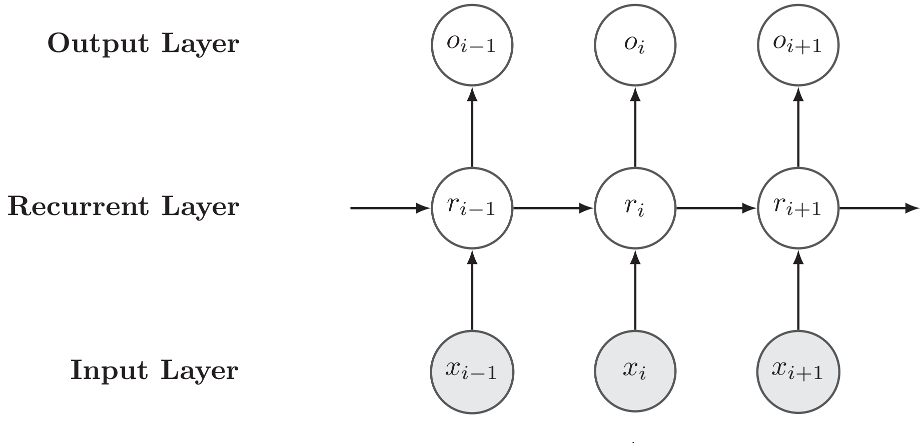


Figure 3.2: A graphical depiction of a recurrent layer.

they allow the network to model long-range dependencies in the sequences, or "remember" what came before. The exact nature of the internal state and the update equations for the recurrent layer depend on which flavor of RNN is in question. In the vanilla RNN, each neuron in the recurrent layer has a standard activation function such as the logistic sigmoid, and the internal state variable is updated through this activation. In response to shortcomings of the vanilla RNN, a more sophisticated recurrent layer was developed nearly two decades ago called long short-term memory, or LSTM [34]. More recently, another type of recurrent layer has been introduced, known as the gated recurrent unit, or GRU [35]. Both of these recurrent structures aim to be able to capture and model long-range dependencies in the sequence data. The authors in [36] conclude that the performance of LSTM and of GRU layers are comparable on a number of sequence learning tasks. Additional technical details and general information about RNNs and their successes can be found at [37].

We now describe a simple RNN architecture for symbolic sequence classification. The first input layer just takes the input sequence data in the one-hot representation. The next layer is a recurrent layer, either LSTM or GRU. The third layer is a conventional feedforward layer. The final output layer is a two-dimensional softmax layer, which means it outputs a stochastic vector of length 2. We interpret this output vector as giving the probability of each label. This simple model is fully specified once the number of neurons in each layer is set and the settings for the recurrent layer are given. The model is trained by minimizing cross entropy loss.

Chapter 4. Experiments

4.1 Introduction

In this chapter, we put several machine learning approaches described in previous chapters to the test. We are primarily interested in finding successful techniques for utilizing medical code sequence data in disease prediction and survival analysis tasks. To this end, we focus exclusively on using code data rather than the full complement of healthcare data at our disposal. In a real-world clinical or insurance setting, we would attempt to incorporate all kinds of available data. Given the academic goals of the present work, however, it is appropriate to focus solely on medical codes.

The structure of this chapter is as follows. We first survey related work, and then describe our particular datasets. We next detail the various experiments that we performed, and present the results. We follow with a discussion of the experimental results, and finish the chapter with some concluding remarks.

4.2 Related Work

The use of statistical models and machine learning algorithms for predictive tasks in healthcare is well established in the literature. For the task of disease prediction, two especially popular techniques are logistic regression and the Cox proportional hazards model. For example, Echouffo-Tcheugui et al. [38] review 30 CKD risk models found in the literature since the 1980s, each of which is either based on logistic regression or Cox proportional hazards. Further examples are found in [39], [40], [41], [42], and [43].

Researchers have also used other machine learning methods for disease prediction. Cruz et al. [17] give an overview of a few machine learning techniques used for cancer prognosis, including decision trees, neural networks, and nearest neighbor classifiers. Conroy et al. [44] build a Weibull hazards model on basic clinical and demographic data to predict risk of cardiovascular disease. Khalilia et al. [19] use a random forest for disease prediction on highly imbalanced data.

Topic modeling techniques based on LDA and its variants have been applied to a variety of healthcare tasks in recent years. Researchers have used both freetext and coded medical data in these efforts. Halpern et al. [10] investigate both supervised and unsupervised topic models as a means of dimensionality reduction and feature extraction for clinical prediction tasks. In particular, they train the models on emergency department nurse triage notes (freetext data) and use the learned topic distributions to predict patient risk for developing sepsis and for being admitted to the ICU. Perotte et al. [11] address the problem of automatically assigning ICD-9 codes to patient discharge summaries using a supervised topic model with hierarchical labels. Lehman et al. [45] train a topic model on UMLS codes extracted from unstructured nurse notes for the purposes of predicting hospital mortality. They demonstrate that the learned topic weights associated with a patient can improve traditional risk stratification algorithms. Salleb et al. [12] use a topic model on freetext from EHRs for the purpose of exploring issues related to infant colic. They hypothesize that the learned topics may be useful in automatically flagging cases of infant colic from EHRs even when the keyword "colic" is not present in the record. Luo et al. [46] apply topic modeling to ICD-10 codes for purposes of summarizing clinical information and generating medical research hypotheses.

Researchers have also focused specifically on incorporating ICD-9 code data into prediction algorithms. Sun et al. [47] develop a method to combine both knowledge and data driven risk factors, including ICD-9 codes, in heart failure prediction. For each individual, they map the ICD-9 codes to a BOW representation indicating the frequency of each code. Singh et al. [48] utilize the hierarchical structure of ICD-9 codes to develop feature vector representations of patients based on their ICD-9 codes. The four types of feature vectors presented are variants of the simple BOW representation of the code sequence. Tsui et al. [49] use ICD-9 codes to develop an epidemic detection algorithm. It should be noted that in this study, the ICD-9 codes are aggregated accross the population, so the predictive value of ICD-9 codes at the individual level was not assessed. Davis et al. [50] develop a collaborative filtering approach to disease prediction using ICD-9 codes.

Of particular note is recent work on developing more sophisticated disease progression models. For example, Wang et al. [14] build an unsupervised model based on Markov jump processes and Markov chains. The model infers a set of disease stages and associated comorbidities, as well as the progression of these stages over time. Tangri et al. [51] focus on the task of modelling the ways in which CKD progresses to end stage renal failure. They model this progression using a sequence of Cox proportional hazards models. Finally, Lipton et al. [52] use a RNN with a LSTM recurrent layer to predict which diagnosis codes are assigned to health episodes.

Our present work distinguishes itself from the literature by focusing on the use of past medical code sequences for *future* disease prediction, and by comparing various machine learning methods head-to-head on this task.

4.3 Data

We perform our experiments on two healthcare datasets from different regions in the United States. We refer to these two datasets as $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. These datasets contain, among other things, insurance claims records for a number of people over the span of several years. They also indicate the insurance enrollment status of each person throughout this interval, so that we can determine whether someone is continuously covered by the insurance, or has gaps in his coverage. This is important, since we do not observe any insurance claims data for an individual during periods of no coverage, and therefore we have no way of knowing whether the individual was diagnosed with some target disease during that time. We note that $\mathcal{D}_1$ is considerably larger than $\mathcal{D}_2$.

For each dataset, after fixing the length of the observation period N_o and the followup time N_f (both in years), as well as the type of coded data to use (whether diagnosis codes, procedure codes, drug codes, or all of the above) and the target disease, we construct a prepared dataset as follows:

- ■ for each person in the dataset:
    - • if the person has a continuous insurance enrollment period of at least $N_o + N_f$ consecutive years:
        - ∗ record the codes that occur in the first N_o years of the first continuous enrollment period (the observation period);
        - ∗ if the person was diagnosed with the target disease before the end of the observation period, leave this person out;
        - ∗ if the person was first diagnosed with the target disease within N_f years of the end of the observation period (the followup time), set the target variable equal to 1;
        - ∗ if the person was not diagnosed with the target disease at any point before the followup time, set the target variable equal to 0.
    - • otherwise:
        - ∗ leave this person out.

We carried out this process for the target diseases CKD and DM. The ICD-9 diagnosis codes used to determine target disease diagnosis for both of these diseases are given in Table 4.1 The resulting data allow us to address the following question: given the medical codes for an undiagnosed individual collected during an observation period of N_o years, can we predict whether this person will be diagnosed with the target disease during the followup period of N_f years?

Target Disease	ICD-9 Codes
Chronic Kidney Disease (CKD)	403.x, 404.x, 582.x, 583.x, 585.x, 586.x, 588.0
Diabetes Mellitus (DM)	250.x0, 250.x2

Table 4.1: ICD-9 codes associated with CKD and DM.

4.4 Disease Prediction Experiments

In this section, we describe the various experiments we performed and report their results.

4.4.1 E1: Finding the Best Classifier. In this experiment, we use $\mathcal{D}_1$ with only ICD-9 diagnosis codes to train and evaluate several different classifiers using a variety of different data representations. We predict both CKD and DM, with $N_o \in \{1, 2\}$ and $N_f = 2$.

For each target disease, we randomly partition the prepared dataset into a training set (50%), validation set (25%), and test set (25%). We do so in a *stratified* manner, so that the proportion of diseased to non-diseased cases is the same in all three sets. The validation set is used for hyperparameter optimization as follows: for each family of classifiers that we consider, there may be a set of options over which we wish to optimize, such as the type and weight of the regularization term in logistic regression, the number of trees in a random forest, the window size for feature functions in a conditional random field, or the sizes of the layers in a recurrent neural network. We call these options *hyperparameters*. For each hyperparameter setting under consideration, we train the classifier on the training set and then test its performance on the validation set. The hyperparameter setting and trained classifier which correspond to the best validation performance are retained, and we finally evaluate and report the performance of just this classifier on the test set.

We now list the classifiers and data representations that we include in the experiment, along with the hyperparameter sets over which we optimize.

LR Models. We train logistic regression classifiers using three data representations: the simple BOW representation (LR+BOW), the lower-dimensional representation given by the CCS mapping (LR+CCS), and the LDA topic distribution representation (LR+LDA). In each of these three cases, we include an L_2 penalty term and we optimize the penalty weight

C over the set $\{0.1, 0.5, 1.0, 5.0\}$. For LR+LDA, we additionally optimize the number of topics in the LDA model over the set $\{10, 30, 50, 100\}$.

RF Models. We train random forest classifiers using the same three data representations as with logistic regression, and we employ similar notation to indicate each of these data representations paired with random forest: RF+BOW, RF+CCS, and RF+LDA. In each of these cases, we optimize the number of trees in the forest over the set $\{50, 100, 250, 500\}$. As before, in the case of RF+LDA, we additionally optimize the number of topics over the set $\{10, 30, 50, 100\}$.

CRF Models. We train conditional random fields using window features consisting of both the ICD-9 codes and the CCS groupings, plus a bias term. We optimize the window size over the set $\{0, 1, 3\}$.

RNN Models. We train recurrent neural networks using both one-hot encoded ICD-9 code sequences (RNN) and one-hot encoded CCS symbolic sequences (RNN+CCS). The network architecture consists of an input layer, a GRU recurrent layer, a dense feedforward layer, and a 2-dimensional softmax output layer. We train using the cross-entropy loss function and two training epochs. We optimize the number of neurons in both the recurrent layer and the feedforward layer over the set $\{128, 256\}$.

The AUC scores for each method are listed in Table 4.2. The ROC curves of the best LR, RF, CRF, and RNN models are displayed in Figure 4.1. The hyperparameters of the best models for each disease and each instantiation of N_o are given in Table 4.3.

4.4.2 E2: Inter-population Prediction. In this experiment, we train a classifier on $\mathcal{D}_1$ and test it on $\mathcal{D}_2$, and vice versa, again only using ICD-9 diagnosis codes. We prep both datasets using $N_o = 1$ and $N_f = 2$. We use the RNN+CCS method with recurrent and feedforward layers both of size 256, and we train using two epochs. Results are given in Figure 4.2 and Table 4.4.

Disease	Method	AUC ($N_o = 1, N_f = 2$)	AUC ($N_o = 2, N_f = 2$)
CKD	LR+BOW	0.745	0.701
	LR+CCS	0.773	0.767
	LR+LDA	0.776	0.801
	RF+BOW	0.755	0.784
	RF+CCS	0.756	0.813
	RF+LDA	0.737	0.776
	CRF	0.590	0.617
	RNN	0.798	0.823
	RNN+CCS	**0.812**	**0.842**
DM	LR+BOW	0.741	0.716
	LR+CCS	0.769	0.784
	LR+LDA	0.757	0.766
	RF+BOW	0.714	0.756
	RF+CCS	0.714	0.753
	RF+LDA	0.694	0.737
	CRF	0.649	0.582
	RNN	0.789	0.778
	RNN+CCS	**0.792**	**0.803**

Table 4.2: E1 results. The AUC scores of various models for the disease prediction task using only ICD-9 diagnosis code data.

4.4.3 E3: Predicting over Variable Followup Periods. In this experiment, we vary N_f over the set $\{1, 2, 3, 4, 5, 6\}$ to investigate how prediction performance changes as we increase the length of the followup period. We use $\mathcal{D}_1$ with all coded data (diagnosis, procedure, and drug codes), and set $N_o = 2$. We use the RNN+CCS method with 256 neurons in the feedforward layer and 128 neurons in the recurrent layer. We create a stratified random partition of the data into a training set (75%) and test set (25%), and we train with two epochs. We report the AUC scores for each disease and each followup time N_f in Table 4.5 and Figure 4.3.

4.5 Discussion

We now discuss the results of the experiments detailed in the previous section.

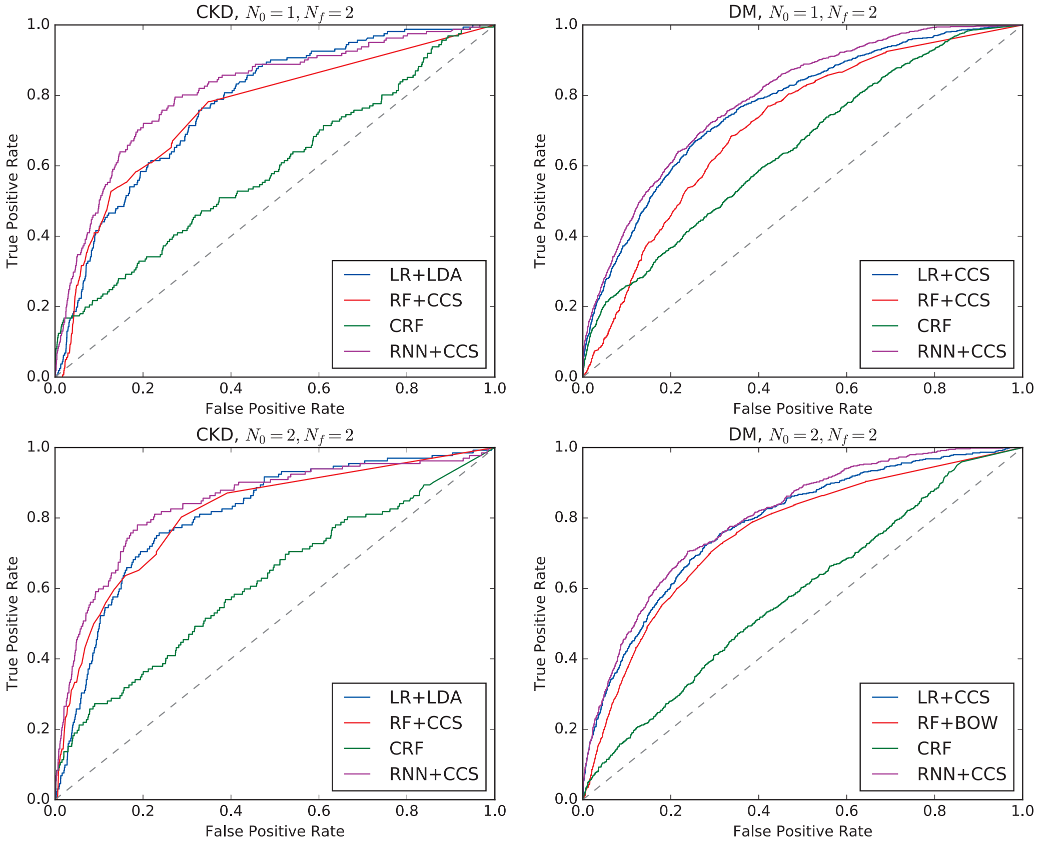
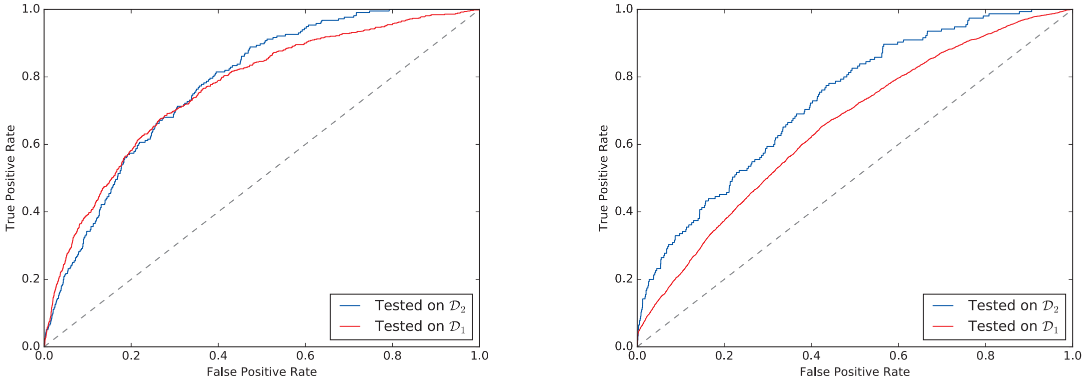


Figure 4.1: E1 results. ROC curves for best LR, RF, CRF, and RNN classifiers.

4.5.1 E1 Discussion. As far as the AUC score metric is concerned, the clear winner among all the classifiers tested is the recurrent neural network trained on the CCS representation of the ICD-9 code sequences. This is true for both CKD and DM, and for both one- and two-year long observation periods. Interestingly, the advantage of the RNN approach over the others is more pronounced for CKD prediction than for DM prediction. The best-performing RNN classifiers all had feedforward layers of size 256. This suggests that perhaps further gains can be achieved by increasing the number of neurons in that layer. The optimal number of neurons in the recurrent layer, however, was sometimes 128 and sometimes 256. This suggests that increasing the size of the recurrent layer may not be beneficial. Of course, one could experiment with all sorts of different – and deeper – RNN architectures.

Disease	Hyperparameters ($N_o = 1$)	Hyperparameters ($N_o = 2$)
CKD	method: RNN+CCS feedforward layer size: 256 recurrent layer size: 256	method: RNN+CCS feedforward layer size: 256 recurrent layer size: 128
DM	method: RNN+CCS feedforward layer size: 256 recurrent layer size: 128	method: RNN+CCS feedforward layer size: 256 recurrent layer size: 256

Table 4.3: E1 results. Hyperparameters for the best classifiers.



(a) CKD

(b) DM

Figure 4.2: E2 results. Performance of RNN+CCS classifier in the inter-population prediction experiment for both CKD and DM.

The LR classifiers generally outperformed the RF classifiers, although not in every case. This result is somewhat surprising, since random forest classifiers can approximate linear classifiers, but are in general capable of learning much more complex decision boundaries. Perhaps the RF classifiers in this case are suffering either from overfitting or from the imbalance problem in the data.

The clear loser is the CRF classifier, with AUC scores that are lower than all the others by quite a margin. This suggests that the window features may not be effective for disease prediction, or possibly that the model is simply not suitable as a whole. A next step would be to test a hidden CRF, which is more tailored for the sequence classification task.

The logistic regression models trained fastest, followed by random forest, conditional random field, and then recurrent neural network. Using the LDA data representation added on additional training time for LR and RF when used. Thus, even though the RNN classifiers

	Tested on $\mathcal{D}_1$	Tested on $\mathcal{D}_2$
CKD	0.765	0.778
DM	0.653	0.732

Table 4.4: E2 results. AUC scores for RNN+CCS classifier in the inter-population prediction experiment for both CKD and DM.

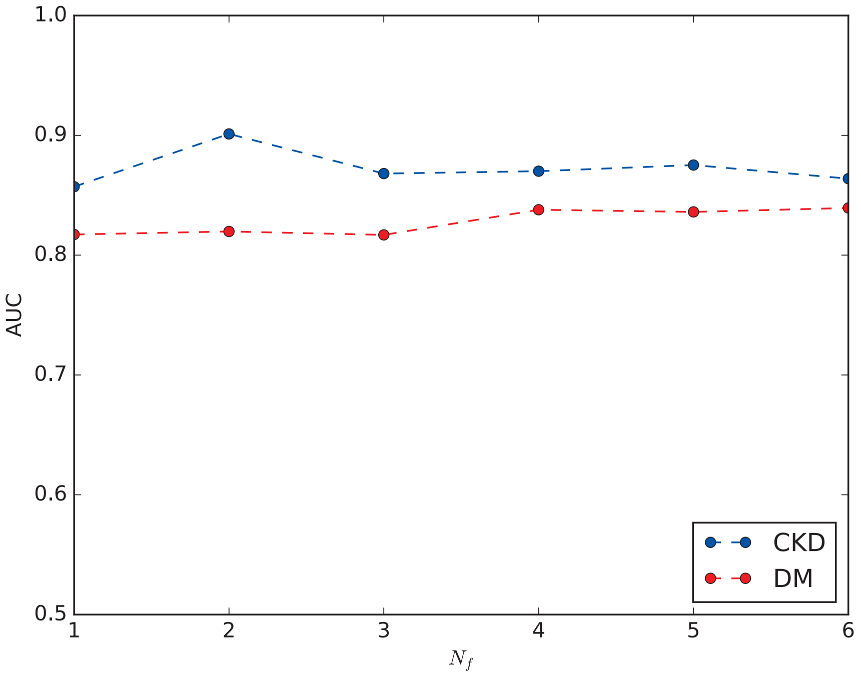


Figure 4.3: E3 results. AUC scores for CKD and DM over several followup times.

have the best AUC scores, they have the downside of longer training time. This fact alone may partly explain the superiority of the RNN models; they were allowed to perform more computation when fitting to the data.

As far as data representation goes, the results of this experiment indicate that the CCS representation of the ICD-9 diagnosis codes is preferable to the plain codes. This suggests that the CCS grouping is an effective dimensionality reduction technique. The LDA representation has a mixed showing; it is sometimes better than the plain codes, but usually worse than the CCS representation, except for one case (compare LR+LDA and LR+CCS on the CKD prediction task with $N_o = N_f = 2$). We can't conclude, then, that the LDA dimensionality reduction technique is particularly effective. Given the extra training time

Disease	$N_f = 1$	$N_f = 2$	$N_f = 3$	$N_f = 4$	$N_f = 5$	$N_f = 6$
CKD	0.857	0.901	0.868	0.870	0.875	0.864
DM	0.817	0.820	0.817	0.838	0.836	0.839

Table 4.5: E3 results. AUC scores for the RNN+CCS classifier over a variety of followup periods.

required in LDA, we don't see a compelling reason to use it for disease prediction models.

We also note that there is generally, although not always, an increase in performance when using two years of observed data ($N_o = 2$) rather than just one year. This makes sense, as having a more complete picture of one's health history should only aid in disease prediction. On the other hand, it is reasonable to assume that the most relevant part of one's health history is usually the most recent part, and so there may be a point at which increasing N_o no longer yields better predictive performance.

Finally, we note that the CKD prediction task generally yields better AUC scores than the DM prediction task. It would be interesting to determine if doctors find CKD prognosis easier than DM prognosis.

4.5.2 E2 Discussion.

The results of E2 show that while classifiers do offer some performance when classifying data from a different population than the training set, the performance is decidedly lower than what is achieved on the original population. The populations in $\mathcal{D}_1$ and $\mathcal{D}_2$ differ in important ways, many of which are due to differences in geographical location. For example, someone living in a rural part of the East Coast will face a different set of health challenges and lead a different kind of life than someone in an urban center in the Midwest. Further, medical coding practices may differ from one hospital system to the next. All this is to say that while there are broad patterns and structure shared by both $\mathcal{D}_1$ and $\mathcal{D}_2$, there are invariably differences in pattern and statistical structure between the two datasets.

Another factor that may account for the diminished performance of the classifiers in this experiment is the small size of $\mathcal{D}_2$. Overfitting and small-sample bias may be at play here.

4.5.3 E3 Discussion. The results of E3 highlight two important things. Firstly, incorporating all coded data, including drug and procedure codes, boosts the performance of the classifier by an appreciable amount. For CKD, the AUC score of the RNN+CCS classifier with $N_o = N_f = 2$ increases from 0.842 to 0.901 when including the additional codes. For DM, the AUC score increases from 0.803 to 0.820. Thus, we conclude that all three types of medical code data have useful and at least partially complementary predictive value.

Second, we note that the AUC scores remain fairly stable as N_f increases from one to six years. This suggests that we can reliably predict disease occurrence over relatively long periods of time. For both CKD and DM, we do observe an increase in AUC score from $N_f = 1$ to $N_f = 2$. This may in part be due to the relatively large increase in the number of positive cases (diseased people) in the dataset when changing the followup time. In particular, the data imbalance problem is substantially reduced when increasing the followup time from one year to two years.

4.6 Conclusion

The results of our experiments show that an individual's history of medical codes can be successfully harnessed for disease prediction. Further, we conclude that modeling the full sequential nature of these data rather than collapsing them to their BOW representation can lead to better predictive performance. Additionally, the CCS grouping on ICD-9 diagnosis codes is a useful and easy means of dimensionality reduction. We observe that simple recurrent neural networks with GRU recurrent layers already provide a rather effective solution to the problem, while logistic regression classifiers provide a very simple, albeit somewhat less accurate, baseline.

There are several directions for future work. We only investigated a very rudimentary RNN architecture, and didn't attempt to employ many of the tricks of the trade out there that can improve training time and performance of neural networks. It is likely that different learning algorithms, activation functions, and deeper architectures (i.e. more layers) will lead to better classifiers. Unfortunately, there is very little by way of robust theory that can guide the development of more effective RNNs at the moment. Nevertheless, one is bound to find improvements given enough experimentation.

Another direction for future work is to develop methods of incorporating additional types of data into the classifier, such as demographic and clinical data. By and large, this shouldn't be an extremely difficult task. However, there may be clever ways to model the interactions between the different data types in a manner that both improves prediction performance and yields clinical insight. Further, there may be better ways of utilizing the code sequences. One could include the time elapsed between successive codes, for example, to capture more of the fine-grained timing.

Finally, just predicting if someone will get a disease within a certain period of time might not be good enough. Predicting *when* the disease is likely to occur can facilitate more targeted and effective preventive care and early treatment. The task of predicting when an event will occur is generally known as survival analysis. Developing survival analysis models using the same type of medical code sequence data considered in this work would be a valuable next step.

Bibliography

[1] David Squires and Chloe Anderson. U.s. health care from a global perspective: Spending, use of services, prices, and health in 13 countries. 2015.

[2] Joachim O Hero, Robert J Blendon, Alan M Zaslavsky, and Andrea L Campbell. Understanding what makes americans dissatisfied with their health care system: An international comparison. *Health Affairs*, 35(3):502–509, 2016.

[3] Harald Schmidt. Chronic disease prevention and health promotion. In *Public Health Ethics: Cases Spanning the Globe*, pages 137–176. Springer, 2016.

[4] Chronic disease overview. `http://www.cdc.gov/chronicdisease/overview/`. Accessed: 2016-06-14.

[5] Age-Adjusted Prevalence. National chronic kidney disease fact sheet, 2014.

[6] Centers for Disease Control, Prevention (CDC), Centers for Disease Control, Prevention (CDC), et al. National diabetes fact sheet: national estimates and general information on diabetes and prediabetes in the united states, 2011. *Atlanta, GA: US Department of Health and Human Services, Centers for Disease Control and Prevention*, 201, 2011.

[7] World Health Organization et al. International classification of diseases:[9th] ninth revision, basic tabulation list with alphabetic index. 1978.

[8] Healthcare Cost, Utilization Project, et al. Clinical classifications software (ccs) for icd-9-cm. *Available at: www. hcup-us. ahrq. gov/toolssoftware/ccs/ccs. jsp. Accessed May*, 11, 2011.

[9] Nips 2015 workshop on machine learning in healthcare. `https://sites.google.com/site/nipsmlhc15/`. Accessed: 2016-06-14.

[10] Yoni Halpern, Steven Horng, Larry A Nathanson, Nathan I Shapiro, and David Sontag. A comparison of dimensionality reduction techniques for unstructured clinical text. In *ICML 2012 Workshop on Clinical Data Analysis*, 2012.

[11] Adler J Perotte, Frank Wood, Noemie Elhadad, and Nicholas Bartlett. Hierarchically supervised latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 2609–2617, 2011.

[12] Ansaf Salleb-Aouissi, Axinia Radeva, R Passonneau, A Tomar, D Waltz, et al. Diving into a large corpus of pediatric notes. *Proc. ICMLWorkshop on Learning from Unstructured Clinical Text*, 2011.

[13] Dina Demner-Fushman, Wendy W Chapman, and Clement J McDonald. What can natural language processing do for clinical decision support? *Journal of biomedical informatics*, 42(5):760–772, 2009.

[14] Xiang Wang, David Sontag, and Fei Wang. Unsupervised learning of disease progression models. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 85–94. ACM, 2014.

[15] Rafid Sukkar, Bradley Wyman, Elyse Katz, Yanwei Zhang, and David Raunig. Modeling alzheimer's disease progression using hidden markov models. *Alzheimer's & Dementia*, 7(4):S147, 2011.

[16] M Ohlsson, C Peterson, and M Dictor. Using hidden markov models to characterize disease trajectories. In *Proceeding of the neural networks and expert systems in medicine and healthcare conference*, pages 324–326, 2001.

[17] Joseph A Cruz and David S Wishart. Applications of machine learning in cancer prediction and prognosis. *Cancer informatics*, 2:59, 2006.

[18] Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 2004.

[19] Mohammed Khalilia, Sounak Chakraborty, and Mihail Popescu. Predicting disease risks from highly imbalanced data using random forest. *BMC medical informatics and decision making*, 11(1):1, 2011.

[20] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[21] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[22] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[24] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674, 2006.

[25] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[26] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[27] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[28] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[29] Thomas G Dietterich. Machine learning for sequential data: A review. In *Structural, syntactic, and statistical pattern recognition*, pages 15–30. Springer, 2002.

[30] Nam Nguyen and Yunsong Guo. Comparisons of sequence labeling algorithms and extensions. In *Proceedings of the 24th international conference on Machine learning*, pages 681–688. ACM, 2007.

[31] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[32] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Machine Learning*, 4(4):267–373, 2011.

[33] Ariadna Quattoni, Sybor Wang, Louis-Philippe Morency, Michael Collins, and Trevor Darrell. Hidden conditional random fields. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10):1848–1852, 2007.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[35] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[36] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[37] Recurrent neural networks. `http://people.idsia.ch/~juergen/rnn.html`. Accessed: 2016-06-15.

[38] Justin B Echouffo-Tcheugui and Andre P Kengne. Risk models to predict chronic kidney disease and its progression: a systematic review. *PLoS medicine*, 9(11):e1001344, 2012.

[39] Heejung Bang, Suma Vupputuri, David A Shoham, Philip J Klemmer, Ronald J Falk, Madhu Mazumdar, Debbie Gipson, Romulo E Colindres, and Abhijit V Kshirsagar. Screening for occult renal disease (scored): a simple prediction model for chronic kidney disease. *Archives of internal medicine*, 167(4):374–381, 2007.

[40] Julia Hippisley-Cox and Carol Coupland. Predicting the risk of chronic kidney disease in men and women in england and wales: prospective derivation and external validation of the qkidney® scores. *BMC family practice*, 11(1):49, 2010.

[41] Kuo-Liong Chien, Hung-Ju Lin, Bai-Chin Lee, Hsiu-Ching Hsu, Yuan-Teh Lee, and Ming-Fong Chen. A prediction model for the risk of incident chronic kidney disease. *The American journal of medicine*, 123(9):836–846, 2010.

[42] Maria Inês Schmidt, Bruce B Duncan, Heejung Bang, James S Pankow, Christie M Ballantyne, Sherita H Golden, Aaron R Folsom, and Lloyd E Chambless. Identifying individuals at high risk for diabetes the atherosclerosis risk in communities study. *Diabetes care*, 28(8):2013–2018, 2005.

[43] Abhijit V Kshirsagar, Heejung Bang, Andrew S Bomback, Suma Vupputuri, David A Shoham, Lisa M Kern, Philip J Klemmer, Madhu Mazumdar, and Phyllis A August. A simple algorithm to predict incident kidney disease. *Archives of internal medicine*, 168(22):2466–2473, 2008.

[44] RM1 Conroy, K Pyörälä, AP el Fitzgerald, S Sans, A Menotti, Gui De Backer, Dirk De Bacquer, P Ducimetiere, P Jousilahti, U Keil, et al. Estimation of ten-year risk of fatal cardiovascular disease in europe: the score project. *European heart journal*, 24(11):987–1003, 2003.

[45] Li-wei Lehman, Mohammed Saeed, William Long, Joon Lee, and Roger Mark. Risk stratification of icu patients using topic models inferred from unstructured progress notes. In *AMIA Annual Symposium Proceedings*, volume 2012, page 505. American Medical Informatics Association, 2012.

[46] Wei Luo, Dinh Phung, Vu Nguyen, Truyen Tran, and Svetha Venkatesh. Speed up health research through topic modeling of coded clinical data.

[47] Jimeng Sun, Jianying Hu, Dijun Luo, Marianthi Markatou, Fei Wang, Shahram Edabollahi, et al. Combining knowledge and data driven insights for identifying risk factors using electronic health records. *American Medical Informatics Association*, 2012.

[48] Anima Singh, Girish Nadkarni, John Guttag, and Erwin Bottinger. Leveraging hierarchy in medical codes for predictive modeling. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 96–103. ACM, 2014.

[49] Fu-Chiang Tsui, Michael M Wagner, Virginia Dato, and CC Chang. Value of icd-9 coded chief complaints for detection of epidemics. In *Proceedings of the AMIA Symposium*, page 711. American Medical Informatics Association, 2001.

[50] Darcy A Davis, Nitesh V Chawla, Nicholas A Christakis, and Albert-László Barabási. Time to care: a collaborative engine for practical disease prediction. *Data Mining and Knowledge Discovery*, 20(3):388–415, 2010.

[51] Navdeep Tangri, Lesley A Stevens, John Griffith, Hocine Tighiouart, Ognjenka Djurdjev, David Naimark, Adeera Levin, and Andrew S Levey. A predictive model for progression of chronic kidney disease to kidney failure. *Jama*, 305(15):1553–1559, 2011.

[52] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.