Computer Science 181: Machine Learning

Austin Li
awli@college.harvard.edu

Spring 2022

Abstract

These are notes[1] for Harvard's *Computer Science 181*, an undergraduate class on machine learning, as taught by Professor Finale Doshi-Velez in Spring 2022.

Course description: Introduction to machine learning, providing a probabilistic view on artificial intelligence and reasoning under uncertainty. Topics include: supervised learning, ensemble methods and boosting, neural networks, support vector machines, kernel methods, clustering and unsupervised learning, maximum likelihood, graphical models, hidden Markov models, inference methods, and computational learning theory. Students should feel comfortable with multivariate calculus, linear algebra, probability theory, and complexity theory. Students will be required to produce non-trivial programs in Python.

Contents

[1] With thanks to Eric K. Zhang for the template.

1 January 25th, 2022

1.1 Logistics

The syllabus can be found here.

Lectures will be remote for the first week, and then in person. The structure of lectures will be an example, lecture content, and concept checks.

Recordings are available on Canvas immediately after class. There is a course textbook.

Homework will be due Friday at 8PM EST and will be biweekly and some programming assignments. There will also be a practical, done in groups of 2-3 people.

There are **six late days** available, at most two per problem set. There will be two timed, closed-book midterms. Office hours and calendars can be found here. This year, there will also be a new "Beyond CS 181" sections to cover more advanced/modern material.

We will use Ed as a discussion forum for questions on course content.

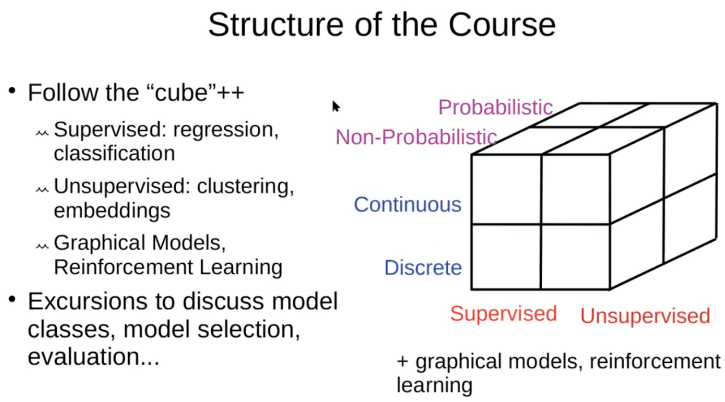
1.2 Introduction

Professor Doshi-Velez's lab is focused on using machine learning for health applications to help doctors make decisions.

This class is focused on the core fundamentals of how machine learning works. We will mainly see current (and older) mainstream systems.

An important thing to keep in mind is ethics. ML promises to offload tasks to a machine, but there may be unintended bias.

We will focus on the "cube" model. Everything we cover in the course will be along three parameters: supervised or unsupervised, continuous or discrete data, and non-probabilistic and probabilistic. The model is described in the following diagram:



Example 1.1 (Deepfakes)**.** Generative adversarial networks are used to create deepfakes. It's somewhat of a "race" against folks detecting deepfakes, who find ways to distinguish or identify them, and the deepfake designers who specifically improve their algorithms to address these use cases.

There's other forms of impersonation and manipulation too, that are enabled by technology. But impersonation in itself is not a new phenomenon: photos and media have been faked for years. What is it that makes deepfakes so concerning? There's been a large amount of media about deepfakes as a political problem. But one of the most pressing and popular uses of deepfakes is in revenge porn. A lot of the social consequences are not necessarily political, but deeply interpersonal, shaping the fabric of our relationships.

Much of machine learning is used to

- Make appropriate modeling choices
- Have sufficient understanding to apply new techniques
- Anticipate and identify potential sources of error
- Evaluate carefully

1.3 Taxonomy

The methods we study can be split into three groups, supervised, unsupervised, and reinforcement learning (RL).

1.3.1 Supervised learning

In *supervised learning*, we have some data $(\boldsymbol{x}, \boldsymbol{y})$ and some metric. Given a new input $\boldsymbol{x}$, we want to make a prediction $\boldsymbol{y}$.

One example of this is *regression analysis*, where labels $\boldsymbol{y}$ are continuous and numeric, or real numbers. For example, Virtu Financial uses regression to predict a stock's future price.

There are also classification problems where labels $\boldsymbol{y}$ are discrete and categorical. For example, *swing typing* uses a language model to predict which word is intended from typing.

1.3.2 Unsupervised learning

One distinction between supervised and unsupervised learning is that in unsupervised learning, there are no labels $\boldsymbol{y}$ available when training. All that is available is the data $\boldsymbol{x}$.

two types of these problems are *clustering* and *embedding*. Clustering is used to find natural groupings of examples in the data, for example Google News, which delivers groupings of stories on the same topic. Clustering involves discrete data.

An example of an embedding technique is point-of-sales data from supermarkets. We can use Principal Component Analysis to embed the time-series into a lower-dimensional space to determine the effect of the 2008 financial crisis along different parameters. This uses continuous variables.

1.3.3 Reinforcement learning

In reinforcement learning (RL), the data is a sequence of triples $(\boldsymbol{s}, a, r)$, a triple of states, actions, and rewards. The essential question is to determine the next action a given a new state $\boldsymbol{s}$.

For example, consider a robot rolling around Cambridge, its state is the current location, and the action is the direction it moves. The reward it gets will be based on what happens to it after it takes the action.

2 January 27th, 2022

For the first half of the semester, we will be focused on supervised learning. Today, we will be focused on regression.

2.1 Regression

Regression is the problem of estimating or predicting some value given some other values.

Example 2.1 (Predicting gate arrival)**.** For example, it is very important for airlines to be able to accurately predict gate arrival. This is important because there may exist many constraints such as planes can only fly for a certain amount of time before maintenance, or certain crews need to get to certain places at certain times in order to get back to their families, etc. To predict gate arrival times for upcoming and current flights, we have lots and lots of historical data available. Each point in this dataset consists of two parts: an example of a flight and a measurement of the gate arrival (the thing we are hoping to predict).

Example 2.2 (Movie recommendations)**.** Another example is sorting movies to recommend. Suppose we are tasked with trying to predict how a particular viewer will rate a particular movie. We have loads of examples of other people's previous movie ratings. Giving this past data, we can train some function (or model) that predicts ratings for movies some person has not seen yet.

2.1.1 Supervised learning

Recall that *supervised learning* is when we have some input $\boldsymbol{x}^*$ and need to predict an output $\boldsymbol{y}^*$. We want to know how we choose or calculate this $\boldsymbol{y}$ and how to measure the *goodness* of our prediction?

Note. $\boldsymbol{y}$ may be continuous or discrete.

2.2 Non-parametric methods

The core idea of non-parametric methods is to make predictions based on similar points in the training set.

Non-parametric models are models that grow with the data. Parametric models have finite parameters.

Note. We note that relevant textbook sections are 2.1, 2.2.1.

2.2.1 k-nearest neighbors (k-NN)

Consider a training dataset of datapoints

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N} = \{(x_1, y_1), \cdots, (x_n, y_N)\} \tag{1}$$

We can predict some label $\hat{y}^*$ for an example $\boldsymbol{x}^*$ by finding the k closes training examples $\{x_k : k \in \mathcal{K}^*\}$ to $\boldsymbol{x}^*$ and calculating

$$\boxed{\hat{y}^* = \frac{1}{k} \sum_{k \in \mathcal{K}^*}^{K} y_k} \tag{2}$$

Implicitly, $\boldsymbol{y}^*$ is a function of input $\boldsymbol{x}^*$ because the training labels we used to calculate $\hat{y}^*$ were selected according to the proximity to $\boldsymbol{x}^*$.

One of the benefits of k-NN is that it works regardless of the curve. We do **not** make assumptions about the underlying relationships, whether parabolic, linear, quartic. This is an appealing property. Moreover, we do not have many parameters to keep track of, only k.

One disadvantage is that we may need to keep the training data. Consider if we have terabytes of training data. For k-NN to work, we need to hold onto the training data to query it every time we want to make a prediction. This is the cost of *not making any assumptions*.

A more important issue is that we need to define similarity or closeness. How do we determine the k nearest neighbors? In one dimension, this is easy. If our training data have high dimensionality, where the units are different for each component, how do we define a distance function that captures the relevant variation? This problem is nontrivial.

2.2.2 Kernelized regression

Another idea is to take a weighted average of all training labels, but upweight labels corresponding to examples close to $\boldsymbol{x}^*$. Close training examples have lots of influence in predicting $\hat{y}^*$ and far-away points don't have any. We can aggregate training labels by

$$\hat{y}^* = \sum_{n=1}^{N} K(\boldsymbol{x}^*, \boldsymbol{x}_n) y_n. \tag{3}$$

Note that $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is our kernel, it measures the closeness of our new point $\boldsymbol{x}^*$ to a training point $\boldsymbol{x}_n$ where d is the dimension of our vector space.

The kernel weights $\{K(\boldsymbol{x}^*, \boldsymbol{x}_n)\}_{n=1}^N$ might not sum to one, so we can ensure a convex combination by normalizing

$$\boxed{\hat{y}^* = \frac{\sum_{n=1}^{N} K(\boldsymbol{x}^*, \boldsymbol{x}_n) y_n}{\sum_n K(\boldsymbol{x}^*, \boldsymbol{x}_n)}.} \tag{4}$$

One main disadvantage, however, is that we cannot predict a value outside the min-max range of the training labels.

2.3 Parametric methods: linear regression

For today, we will focus on *linear regression*.

Note. The relevant reading for this section are textbook sections 2.3, 2.4, 2.5, 2.6.1, and 2.7.1.

Our goal is to make predictions based off of some parameters. This is different than non-parametric regression here our prediction is explicitly a function of our training data and not some weights $\boldsymbol{w} \in \mathbb{R}^w$.

More specifically,we want to find some linear combination of the $\{x_i\}_{i=1}^N$ input values that predict our target y.

Definition 2.3 (Linear regression)**.** Suppose we have an input $\boldsymbol{x} \in \mathbb{R}^D$ and a continuous target $y \in \mathbb{R}$. Linear regression determines weights $w_i \in \mathbb{R}$:

$$y = w_0 + w_1 x_1 + \cdots + w_D x_D \tag{5}$$

Note. We note that w_0 has no corresponding x_0. This is known as the *bias* term and accounts for data that has a nonzero mean.

Linear regression deals with a **continuous** output domain. It is a **supervised** technique. Finally, it is **non-probabilistic**.

In linear regression, we consider training data

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N. \tag{6}$$

Our goal is to learn a function $f(\boldsymbol{x}^*, \boldsymbol{w}) = \hat{y}^*$ that predicts our labels. How do we identify $\boldsymbol{w}^*$?
In general, for regression, we choose a loss function

$$\mathcal{L} : \mathbb{R}^w \to \mathbb{R}, \tag{7}$$

choose a class of functions $f : \mathbb{R}^d \times \mathbb{R}^w \to \mathbb{R}$ and identify the best weights as

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{w}). \tag{8}$$

In linear regression, we choose our function class to be linear functions

$$\hat{y}^* = f(\boldsymbol{x}^*, \boldsymbol{w}) = w_0 + \sum_{n=1}^{N} w_n \boldsymbol{x}_n. \tag{9}$$

2.3.1 Bias trick

One trick is to use the *bias trick* by adding 1 as the zeroth element to each of our training points to absorb the constant into the sum. Then our function is simply

$$f(\boldsymbol{x}^*, \boldsymbol{w}) = \boldsymbol{w}^\intercal \boldsymbol{x}. \tag{10}$$

2.3.2 Least squares loss

We now define the loss. Commonly, loss is a function of the residuals produced by a model.

Definition 2.4 (Residual)**.** The *residual* is the difference between the target y and predicted value $\hat{y}$:

$$\text{residual} = y - f(\boldsymbol{x}, \boldsymbol{w}) = y - \boldsymbol{w}^\intercal \boldsymbol{x} \tag{11}$$

Note (L1 loss)**.** The sum of the residuals is sometimes referred to as *L1 loss*.

The most standard loss **least squares or L2 loss**. It is given by

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 = \frac{1}{2} \sum_{n=1}^{N} \left[(y_n - f(\boldsymbol{x}_n, \boldsymbol{w}) \right]^2 = \frac{1}{2} \sum_{n=1}^{N} \left[y_n - \boldsymbol{w}^\intercal \boldsymbol{x} \right]^2. \tag{12}$$

To find the best weights $\boldsymbol{w}^*$, we can take a derivative of $\mathcal{L}(\boldsymbol{w})$ to find a global minimum:

$$\boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)(-\boldsymbol{x}_n). \tag{13}$$

Generally, we use gradient descent to approach a better setting of our weights:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \boldsymbol{w}^{(t)} - \eta \sum_{n} (y_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)(-\boldsymbol{x}_n). \tag{14}$$

Today, in the case of linear regression, there is an analytic solution. In lecture, we will handwave the solution and derivation. Essentially, we can write everything into matrix form and we have the solution

$$\boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = -2X^\intercal \boldsymbol{y} + X^\intercal X \boldsymbol{w}. \tag{15}$$

Note (Matrix form derivation)**.** Consider the residuals squared in matrix form:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\left(\boldsymbol{y} - X\boldsymbol{w}\right)^{\intercal}\left(\boldsymbol{y} - X\boldsymbol{w}\right). \tag{16}$$

The gradient is given by

$$\boldsymbol{\nabla}_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = -2X^{\intercal}(\boldsymbol{y} - X\boldsymbol{w}) \tag{17}$$

where we use

$$\boldsymbol{\nabla}_{\boldsymbol{b}}(\boldsymbol{a} - C\boldsymbol{b})^{\intercal}D(\boldsymbol{a} - C\boldsymbol{b}) = -2C^{\intercal}D(\boldsymbol{a} - C\boldsymbol{b}) \tag{18}$$

where D is a symmetic matrix. For us, $D = I$. The expression for $\boldsymbol{w}^*$ follows.

In this derivation, there are D dimensions in data and N points. $\boldsymbol{y}$ is an N-dimensional column vector, $\boldsymbol{w}$ is a $(D+1)$-dimensional column vector of weights and X is a $N \times (D+1)$ matrix of data. X is called the *design matrix*.

If we let the derivative vanish, we obtain

$$\boxed{\boldsymbol{w}^* = \left(X^{\intercal}X\right)^{-1}X^{\intercal}\boldsymbol{y}.} \tag{19}$$

The term $\left(X^{\intercal}X\right)^{-1}$ is like the variance of the inputs and $X^{\intercal}\boldsymbol{y}$ is the covariance of inputs with the outputs.

The a given datapoint $\boldsymbol{x}^*$, the prediction is

$$\hat{y}^* = \left[\left(X^{\intercal}X\right)^{-1}X^{\intercal}\boldsymbol{y}\right]^{\intercal}\boldsymbol{x} = \left(X^{\intercal}\boldsymbol{y}\right)^{\intercal}\left(X^{\intercal}X\right)^{-1}\boldsymbol{x}. \tag{20}$$

Note. To take the inverse, we require X to have full rank, which makes $X^{\intercal}X$ positive definite.

Note (Linear algebra)**.** We have D $\boldsymbol{x}$ vectors in N dimensions. We are trying to produce a length N vector $\hat{y}$. $\hat{y}$ is the *closest projection* of $\boldsymbol{y}$ onto the row space of X.

2.3.3 Basis regression

This is a key extension of linear regression. Basis regression is an intermediate regime between linear regression input and neural networks (NN).

3 February 1st, 2022

Announcements

- Problem set 1 is now due February 11
- Office hours will be online this week because dining halls are still running at limited capacity
- The reading is 2.6.2-2.6.3

3.1 Probabilistic regression

Recall that in the first half of the course, we are focused on the supervised learning part of the cube. Last time, we studied a non-probabilistic model with continuous outcomes — regressions. Today, we will focus on continuous outcomes, but in a probabilistic manner.

Example 3.1 (Bat mating systems)**.** This is an example of how regressions can be *off*. A paper on mating systems and brain size, the authors wanted to predict brain size using

$$w_0 + w_1 \times \text{testes size} \implies w_1 > 0. \tag{21}$$

Running this regression, they noted that w_1 is positive. After running another regression, we see that

$$w_0 = w_1 \times \text{bat size} + w_2 \times \text{testes size} \implies w_2 < 0. \tag{22}$$

They then found the errors and found that when the bats are monogamous, their brains are bigger and testes are smaller. The opposite is true for polygamous bats.

Example 3.2 (OKCupid)**.** OKCupid ran a regression and found that men have particular preferences in race, and even when women claimed they have no racial preference, there is a slight preference for men of the same race. This shows that implicitly, people are still taking race as a factor in mating and dating.

Recall last time, we discussed *linear regression*. Recall that our predictions are denoted $\hat{\boldsymbol{y}} = f(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x}$ where $\boldsymbol{x}$ is prepended with a column of 1s for the bias term. We note that the squared loss is given by

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2} \sum_n (y_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2. \tag{23}$$

We chose this loss function. because it is easy to optimize. The optimal weights are given by

$$\boldsymbol{w}^* = (XX^\intercal)^{-1} X \boldsymbol{y}^\intercal. \tag{24}$$

Today, we will discuss a probabilistic view of this loss.

3.1.1 The probabilistic view

We will introduce *generative models*. A generative model is like a story. Given some way of producing the data, we want to produce an estimator that maximizes the probability of the data given the model. We want to ask for this property.

We can consider some data with some Gaussian perturbation

$$\boldsymbol{y} = \boldsymbol{w}^\intercal \boldsymbol{x} + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2). \tag{25}$$

Note (Normal distribution PDF)**.** Recall that the probability distribution function is given by

$$\mathcal{N}(z, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(z-\mu)^2}{2\sigma^2}\right]. \tag{26}$$

This means that we are likely to get some perturbation close to the mean and unlikely to get some perturbation that is far from the mean.

Using this PDF, we can define the probability of the data given the model $p(\text{data}|\text{model})$. This is called the *likelihood of the model*. The model in this case is the choice of $\boldsymbol{w}$ and the model class is the class of linear functions.

Our goal is to maximize the likelihood to find the most likely model.

In our case, this is

$$p(\text{data}|\text{model}) = \prod_{n=1}^{N} p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2). \tag{27}$$

Note. We can write this as a product because there are N predictions, from $\boldsymbol{x}_n \to \boldsymbol{y}_n$. These are all driven by the same $\boldsymbol{w}, \sigma^2$. These are all independent.

Professor Doshi-Velez represented this using *plate notation*.

If we **did not know** $\boldsymbol{w}$, however, this is no longer independence and the product is no longer valid.

Note. We will later take $\boldsymbol{w}$ to be a random variable, but for now, we hold it constant.

We can take the logarithms of both sides and we can write

$$\log p(\text{data}|\text{model}) = \sum_{n=1}^{N} \log\left(p(\boldsymbol{y}_n|\boldsymbol{x}_n, \boldsymbol{w}, \sigma^2)\right). \tag{28}$$

Then $y \sim \mathcal{N}(\boldsymbol{w}^\intercal \boldsymbol{x}, \sigma^2)$ because $\boldsymbol{w}^\intercal \boldsymbol{x}$ is a constant. Then

$$\begin{aligned}\log p(\text{data}|\text{model}) &= \sum_n \log \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma^2}(\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2\right] \\ &= \sum_n \log \frac{1}{\sigma\sqrt{2\pi}} + \sum_n \left(-\frac{1}{2\sigma^2}\right)(\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2 = N\left(-\frac{1}{2}\log 2\pi - \frac{1}{2}\log\sigma^2\right) + -\frac{1}{2\sigma^2}\sum_n (\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2\end{aligned} \tag{29}$$

By observation, the first term is a constant and there is a scale factor on the second term. Note that the model that maximizes the likelihood (minimizing the second term) is the same as minimizing the square loss. Then solving for the optimal $\boldsymbol{w}$ gives us the same solution as the least squares loss.

Note. This means that least squares loss requires an assumption that the noise is Gaussian. This is because these two are mathematically equivalent. The probabilistic regression assumes Gaussian noise to reproduce least squares loss.

We can also think about least squares loss as reducing the error with respect to the second moment. The second moment is our notion of *loss*. We can also apply least squares loss to another form of noise. This would be an approximation. We note that in the case of Gaussian noise, the problems overlap exactly.

3.1.2 Matrix form

Let us consider matrices X, Y and our goal is to maximize $p(Y|X, \boldsymbol{w}, \sigma^2)$. We not that $Y \sim \mathcal{N}(\boldsymbol{w}^\intercal X, \mathbb{I}_n\sigma^2)$. Let $\Sigma = \mathbb{I}\sigma^2$ be the variance and $\Sigma^{-1} = \frac{1}{\sigma^2}\mathbb{I}$ is the covariance. Then for a multivariate Gaussian distribution, we have

$$\begin{aligned}p(Y|X, \boldsymbol{w}, \mathbb{I}\sigma^2) = \log\left\{\frac{1}{\sqrt{2\pi|\Sigma|}}\exp\left[-\frac{1}{2}(Y - \boldsymbol{w}^\intercal X)\Sigma^{-1}(Y - \boldsymbol{w}^\intercal X)^\intercal\right]\right\} \\ = \log \text{constants} - \frac{1}{2\sigma^2}(Y - \boldsymbol{w}^\intercal X)(Y - \boldsymbol{w}^\intercal X)^\intercal\end{aligned} \tag{30}$$

where we recover the above.

3.1.3 Optimal variance

We can determine the optimal σ^2. Consider taking the derivative of the log-likelihood with respect to σ^2 and find the maximum. Then we have

$$\begin{aligned}\frac{\partial}{\partial\sigma^2}\log p(\text{data}|\text{model}) &= -\frac{N}{2}\frac{1}{\sigma^2} - \frac{1}{2}\sum_n (\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2(-1)\left(\frac{1}{\sigma^2}\right)^2 \\ &= -\frac{N}{2}\sigma^2 + \frac{1}{2}\sum_n (\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2 = 0 \implies \boxed{\sigma^2_{\text{ML}} = \frac{1}{N}\sum_n (\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2} \end{aligned} \tag{31}$$

which is just the empirical variance! This makes sense because given some Gaussian noise, in expectation, we will get the mean value $\boldsymbol{w}^\intercal \boldsymbol{x}_n$. This gives us the expected variance.

3.1.4 The geometric perspective

Thus far, we have discussed an optimization view and probabilistic view of least squares loss. Now we, will discuss the geometric view.

We will think about projecting Y onto the space spanned by X_d. The *optimal projection* is given by

$$\sum_d X_d \frac{\langle X_d, Y\rangle}{\langle X_d, X_d\rangle} = X^\intercal (XX^\intercal)^{-1} XY^\intercal = X^\intercal \boldsymbol{w}. \tag{32}$$

Example 3.3 (Different generative models over ε)**.** We can consider a Laplace error with $\lambda = 1$ with PDF $\exp -\lambda|\varepsilon|$. The Gaussian as above, and the T-distribution for $\nu = 1$ with PDF $(1+\varepsilon^2)^{-1}$.

1. Convert each PDF on ε into a loss function
2. Suppose we have some regression lines — which ones will correspond to each loss?

The main strategy for the first question is to substitute the PDF into the log maximum likelihood. For the Laplace distribution, we have

$$\sum_n \log\exp\left[-\lambda|\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n|\right]. \tag{33}$$

and for the T-distribution, we have

$$\sum_n (-1)\log\left[1 + (\boldsymbol{y}_n - \boldsymbol{w}^\intercal \boldsymbol{x}_n)^2\right]. \tag{34}$$

Note that the Laplace loss will want to be closer to the two data points.

4 February 3rd, 2022

Announcements

- Today's lecture will focus on textbook sections 3.1-3.5
- Homework 1 will be due February 11
- Our first midterm will be on March 1

4.1 Classification

Today, we will continue on supervised learning, focusing on non-probabilistic models with *discrete* targets $\hat{y}$.

Example 4.1 (Insect classification)**.** Insect classification based on visuals and images by Hassan et al 2014.

Example 4.2 (COVID-19 patients)**.** A validated, real time prediction model for favorable outcomes in hospitalized COVID-19 patients, Razavian et al 2020.

We can consider a plot of two parameters $\boldsymbol{x}_1, \boldsymbol{x}_2$ where $\boldsymbol{y} = f(\boldsymbol{x}_1, \boldsymbol{x}_2)$ and we want to create a boundary to partition or classify target values. We want to decide what model class to use and what our objective function is — to define our loss.

We will use c_i to represent class i, for example: $\boldsymbol{y} \in \{c_1, c_2, \cdots, c_k, \cdots, c_K\}$. We can represent $\boldsymbol{y}$ as a row vector with entries $\{0, 1\}$ where 1 in position i represents membership of class c_i.

We can also use the encodings $y \in \{-1, 1\}$ or $y \in \{0, 1\}$. The former will be used in this class.

Note. We want to note that k-NN still works. We just change the average to a vote and we are finished.

We note that linear regression for classificaiton does not work. We can consider adding points to a step function. The more points we add, the partition point may change, even when outlier points should *not matter* in determining the class.

4.1.1 Parametric models for classification

This is an example of a model class. We can consider target values defined by

$$\hat{y} = \text{sign}(f(\boldsymbol{x}, \boldsymbol{w})), \qquad f(\boldsymbol{x}, \boldsymbol{w}) \equiv \boldsymbol{w}^\intercal \boldsymbol{x}, \tag{35}$$

a linear model.

Note. We are using the most simple model to understand the underlying mechanics. Once we get to more complicated objectives and functions, we simply just change the objective function to obtain new models.

We can consider an equation for the definition of a boundary

$$0 = w_0 + w_1 x_1 + w_2 x_2 = w_0 + \langle w_1 + w_2, x_1 + x_2 \rangle. \tag{36}$$

The w_0 is the offset. We note that the terms of x_1, x_2. The vector (w_1, w_2) is perpendicular to the decision boundary. The weights are the quickest way to move to flip the class determined by the classifier.

4.1.2 Objective function

The loss function is generally dependent on the application.

We can define

$$\ell_{0/1}(z) \equiv \begin{cases} 1, & z > 0 \\ 0, & \text{else} \end{cases}, \qquad \mathcal{L}_{0/1}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell_{0/1}(-y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n)). \tag{37}$$

Note that $\boldsymbol{w}^\intercal \boldsymbol{x} > 0 \implies y = 1$ and $\boldsymbol{w}^\intercal \boldsymbol{x} < 0 \implies y = -1$. This loss function tells us that if we classify correctly, we have loss 0 and if we classify incorrectly, we have loss 1.

Note. We are simply counting the number of cases and does not account for uncertainty — the distance from the boundary. Because of this, the classifier will not know where to focus.

We can't take gradients to solve/minimize this and the solution is NP-hard.

4.1.3 Hinge loss

We can define *hinge loss* as

$$\ell_{\text{hinge}}(z) = \begin{cases} z, & z > 0 \\ 0, & \text{else} \end{cases}, \qquad \mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \ell_{\text{hinge}}(\text{sign}(\boldsymbol{w}^\intercal \boldsymbol{x})) = \sum_{\text{incorrect } m} -(\boldsymbol{y}_m)\boldsymbol{w}^\intercal \boldsymbol{x}_m, \tag{38}$$

that is, the loss is proportional to how far you are from the boundary. We can take gradients of this loss function! We have

$$\boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \boldsymbol{\nabla}_{\boldsymbol{w}} - \sum_{\text{bad}n} y_n \boldsymbol{w}^\intercal \boldsymbol{x}_n = - \sum_{\text{bad } n} y_n \boldsymbol{x}_n \implies \boldsymbol{w} \to \boldsymbol{w} - \eta \boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) \tag{39}$$

where we follow the gradient to optimize/minimize the loss.

4.1.4 Stochastic gradient descent

In general form, *stochastic* gradient descent involves choosing a batch of data of size M and compute the gradient with respect to $\boldsymbol{w}$ for the batch $\boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}_M(\boldsymbol{w})$.

There are ways to *anneal* the step size η, but we want η to be big enough that we can reach the optimum.

Stochastic gradient descent is a good algorithm. An older algorithm involves doing SGD with $M = 1$. If $\hat{y}_m$ is incorrect, we update with $\boldsymbol{w} \to \boldsymbol{w} - \eta \boldsymbol{x}_m$. This *Perceptron algorithm* converges to a solution if the data are separable.

This algorithm is understood as optimizing a hinge loss in a particular setting.

Note. If the loss is convex and the boundary is linear, we will converge to the minimum.

4.2 Evaluating classification

We can first consider errors. There are four cases:

Error	$\boldsymbol{y}$	$\hat{y}$
True positive	1	1
False positive	-1	1
False negative	1	-1
True negative	-1	-1

Table 1: Errors

We may care more or less about certain types of errors. Note that $\ell_{0/1}$ treats both false positive and false negative errors the same. We can define accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{40}$$

We can define true positive rate (TPR), false positive rate (FPR).

$$\text{TPR} = \frac{TP}{TP+FN}, \qquad \text{FPR} = \frac{FP}{FP+TN}. \tag{41}$$

We can also define *precision* and *recall*:

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall} = \frac{TP}{TP+FN} \tag{42}$$

We can interpret precision as the number of true alarms over the number of alarms. Recall is the proportion of identified positives over true positives. This is the same as TPR.

We can plot the FPR against the TPR on $[0,1] \times [0,1]$ and obtain the ROC curve. This ROC curve tells us our trade-off in model choice. The area under the ROC curve, the AUC is often used as a metric for classification. The maximum AUC is 1.

Example 4.3. Assume that the curve is always constant or increasing. If a system has AUC $= 0.75$ and we care about the maximum TPR we can get if the FPR $= 1/8$.

What is the min and max TPR that is possible? We can minimize and maximize the TPR by considering different plots. We can achieve TPR $= 0$ at FPR $= 1/8$ by considering

$$\text{TPR} = \begin{cases} 0, & \text{FPR} < \frac{1}{4} \\ 1, & \text{else} \end{cases} \tag{43}$$

and achieve a maximum of TPR $= 6/7$ with

$$\text{TPR} = \begin{cases} 0, & \text{FPR} < \frac{1}{8} \\ \frac{6}{7}, & \text{else} \end{cases}. \tag{44}$$

5 February 8th, 2022

5.1 Probabilistic classification

Announcements

- Homework 1 is due Friday and homework 2 will be released the same day
- Today's textbook section is 3.6

The core idea in probabilistic classification is to determine the probability of a class y given some input x. We will motivate our discussion with some examples:

Example 5.1 (Emails)**.** Classifying email as spam

Example 5.2 (Bad weather)**.** Predicting parking bans in inclement weather. What is the probability there will be greater than five inches of snow?

We are often not only interested in predicting class labels, but are also interested in approximating the *probability* that an example belongs to a particular class.

We recall that last week, we used the parametric classification model to predict label $\hat{y}_n$ as

$$\hat{y}_n = \begin{cases} +1, & \boldsymbol{w}^\intercal \boldsymbol{x} + w_0 > 0 \\ -1, & \text{otherwise} \end{cases} \tag{45}$$

where we learn weights $\boldsymbol{w}$ by training using the hinge loss.

Today we discuss two approaches to calculating the *probability* that an example is positive. In the two approaches, we will use *maximum likelihood estimation* to learn model parameters.

Today, we will assume that the two classes are $\{0, 1\}$ where 1 is positive and 0 is the negative label. We have

$$\hat{y} = \begin{cases} 1, & p(y = 1|\boldsymbol{x}) > p(y = 0|\boldsymbol{x}) \\ 0, & \text{otherwise} \end{cases} \tag{46}$$

5.2 Discriminative approach

Our goal is to find parameters $\boldsymbol{w}$ that maximize the *conditional probability* of labels in the data:

$$\boxed{\text{argmax}_{\boldsymbol{w}} \prod_n p(y_n|\boldsymbol{x}_n, \boldsymbol{w})} \tag{47}$$

where the term $p(y_n|\boldsymbol{x}_n, \boldsymbol{w})$ is called the conditional likelihood.

In this setting, the labels y_n are generated based on covariates or features $\boldsymbol{x}_n$. We take the produce here because we think of pairs $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ as independently and identically distributed.

We need to choose a model class. In this lecture, we will model $p(y|\boldsymbol{x})$ using a sigmoid (logistic) function

$$p(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\intercal \boldsymbol{x})}. \tag{48}$$

The sigmoid function is often denoted using a σ and takes scalar values as an input:

$$\sigma(z) = \frac{1}{1 + \exp -z}. \tag{49}$$

The sigmoid flattens its input to output values between 0 and 1.

In logistic regression $p(y=0|\boldsymbol{x}) = 1 - p(y=1|\boldsymbol{x})$. If we write $h = \boldsymbol{w}^\intercal \boldsymbol{x}$, then

$$p(y=1|\boldsymbol{x}) = \frac{1}{1+e^{-h}}, \qquad p(y=0|\boldsymbol{x}) = \frac{1}{1+e^{h}}. \tag{50}$$

Because $y \in \{0,1\}$, we can rewrite $p(y|\boldsymbol{x})$ using the power trick:

$$p(y|\boldsymbol{x}) = p(y=1|\boldsymbol{x})^y \cdot p(y=0|\boldsymbol{x})^{1-y} \tag{51}$$

The parameters that *maximize* a likelihood function (the MLE) also *minimize* the negative log-likelihood function. We can treat the negative log-likelihood — the expression we are minimizing — as the "loss function." Thus the negative log-likelihood or loss over the dataset $\mathcal{D}$ can be written as

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{w}) = -\sum_n \ln\left[p(y_n|\boldsymbol{x}_n)\right] = \sum_n y_n \ln\left[1+\exp -h_n\right] + \sum_n (1-y_n)\ln\left[1+\exp h_n\right] \tag{52}$$

where we have used log ruls and the power trick. The second term is understood as the loss of the classifier on a negative example $(\boldsymbol{x}_n, y_n = 0)$. The components pick out the loss contributions from both samples.

We can draw a picture of the logistic loss and compare it to 0/1 loss and hinge loss.

Note. The logistic loss function is differentiable and convex. We can use gradient descent methods to minimize the loss. The logistic loss penalizes the classifier's predictions on data points where the classifier predicts the correct label because it is proportional to our belief. The logistic loss prefers to make better decisions on correct predictions, pushing away from the decision boundary.

The gradient of the loss is given by

$$\boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L}_{\mathcal{D}}(\boldsymbol{w}) = \sum_n -y_n x_n p(y_n=1|\boldsymbol{x}_n) + (1-y_n) x_n p(y_n=0|\boldsymbol{x}_n) \tag{53}$$

5.3 Generative approach

Our goal is to find parameters $\boldsymbol{w}$ that maximize the *joint distribution* of features $\boldsymbol{x}_n$ and labels y_n:

$$\boxed{\text{argmax}_{\boldsymbol{w}} \prod_n p(\boldsymbol{x}_n, y_n|\boldsymbol{w})} \tag{54}$$

where the term $p(\boldsymbol{x}_n, y_n|\boldsymbol{w})$ is called the *joint likelihood*.

In this model, the label comes first and the label creates the data.

The generative model is flexible. It can add knowledge and handle missing labels elegantly.

We will illustrate the generative approach using two methods: *multi-variate Gaussian* models and *naive Bayes* models.

Steps for specifying and learning a generative model are

1. Decide on a data-generating process
2. Choose a parametric model
3. Minimize the negative-log likelihood

Recall the product rule:

$$p(\boldsymbol{x}, y) = p(\boldsymbol{x}|y) \cdot p(y). \tag{55}$$

$p(\boldsymbol{x}, y)$ is called the class-conditional, $p(y)$ is called the class prior. This implies that labels y are used to generate covariates $\boldsymbol{x}$.

Today, if the covariates $\boldsymbol{x}$ are continuous, we will assume they are Gaussian. If they are discrete, we will use naive Bayes. Today we will also assume a Bernoulli class prior.

To find the MLE of parameters $\boldsymbol{w}$, we can apply the chain rule:

$$\text{argmax}_{\boldsymbol{w}} \prod_n p(\boldsymbol{x}_n, y_n|\boldsymbol{w}) = \text{argmax}_{\boldsymbol{w}} \prod_n p(\boldsymbol{x}_n|y_n, \boldsymbol{w}) p(y_n|\boldsymbol{w}). \tag{56}$$

We can transform the likelihood into a negative log-likelihood and applying log rules gives us our loss function

$$\text{argmin}_{\boldsymbol{w}} \left\{ -\sum_n \ln \left[p(\boldsymbol{x}_n|y_n, \boldsymbol{w}) \right] - \sum_n \ln \left[p(y_n|\boldsymbol{w} \right] \right\}. \tag{57}$$

We will predict labels y for covariates $\boldsymbol{x}$ using Bayes' Rule:

$$p(y = 1|\boldsymbol{x}) \propto p(y = 1) p(\boldsymbol{x}|y = 1). \tag{58}$$

5.3.1 Class prior

We will model our class prior as a Bernoulli random variable with probability θ. Thus

$$p(y) = \theta^y (1 - \theta)^{(1-y)} \tag{59}$$

and taking $\partial_\theta \mathcal{L}(\theta)$ we find that the MLE θ^* of θ is

$$\theta^* = \frac{1}{N} \sum_n y_n. \tag{60}$$

5.3.2 Class conditional with continuous x

Say that $\boldsymbol{x}$ are continuous. We assume that covariates are distributed as Gaussians, where the parameters are distributed differently depending on the true label y:

$$\boldsymbol{x}|y = 0 \sim \mathcal{N}(\mu_0, \Sigma_0), \qquad \boldsymbol{x}|y = 1 \sim \mathcal{N}(\mu_1, \Sigma_1), \qquad \boldsymbol{w} = \{\mu_0, \Sigma_0, \mu_1, \Sigma_1\}. \tag{61}$$

We think of the training data as being in two piles, one for each class. For class 0, we use $\{(\boldsymbol{x}_n, y_n)\} : y_n = 0$ and we estimate μ_0, Σ_0 for the negative class. We then estimate μ_1, Σ_1 for class 1.

When we derive the MLE for parameter μ_0, we find that

$$\boxed{\hat{\mu}_0 = \frac{1}{N_0} \sum_{n:y_n=0} \boldsymbol{x}_n} \tag{62}$$

where N_0 is the total number of examples with label 0. This is the empirical average of the covariates for all of the training data points in class 0.

Note. If the class-conditional distributions have the same covariance matrix, the learned decision boundaries will be linear. Otherwise, they will be quadratic.

The decision boundary for our generative model is

$$\hat{y} = \begin{cases} 1, & \text{if } p(y = 1) p(\boldsymbol{x}|y = 1) > p(y = 0) p(\boldsymbol{x}|y = 0) \\ 0, & \text{otherwise} \end{cases} \tag{63}$$

5.3.3 Class-conditional with discrete x

We now consider the case with discrete data x_d takes on one of J values $\{1, \ldots, J\}$. For example, x_d could be hair color.

The naive Bayes assumption is that each dimension of data $\boldsymbol{x}$ is independent, conditioned on the class:

$$\boxed{p(\boldsymbol{x}|y=1) = \prod_d p(x_d|y=1).} \tag{64}$$

We use naive Bayes to limit the number of parameters needed to specify our model. If features were dependent on each other, then we need to explicitly model this dependence using additional parameters.

We model each feature x_d as a categorical distribution. This is a generalization of the Bernoulli. For example, if there are three hair colors, then the vector $\pi_{dk} = [\pi_{dk1} = 0.2, \pi_{dk2} = 0.5, \pi_{dk3} = 0.3]$ parameterizes x_{dk} where π_{kdj} is the probability in class k of feature d taking on value j.

We can do a maximum likelihood fit of parameters π_0 for class 0 and π_1 for class 1.

Note. In the discrete case, our naive Bayes classifier has linear decision boundaries. Moreover, in the course, we will use the notation $C = \{C_1, \ldots, C_k\}$ to denote the k classes $C_1, C_2, \ldots$.

5.4 Multi-class classification

To move from binary to a multi-class setting, the generative case is easy. We just use a categorical class prior and estimate class conditions. Classify as

$$\text{argmax}_k p(\boldsymbol{x}|y_k)p(y_k). \tag{65}$$

In a discriminative setting, we need to have separate parameters $\boldsymbol{w}_k$ for each class. We classify using the softmax function

$$\boxed{p(y = C_k|\boldsymbol{x}) = \frac{\exp \boldsymbol{w}_k^\intercal \boldsymbol{x}}{\sum_{k=1}^K \exp \boldsymbol{w}_k^\intercal \boldsymbol{x}} = 1.} \tag{66}$$

To learn non-linear decision boundaries, we can apply basis functions to our data. See the lecture slides for visualizations.

6 February 10th, 2022

Announcements

- Midterm 1 is on March 1st

Today's lecture is perhaps the most important lecture in terms of application of knowledge in the real world. Today, we discuss the frequentist view of model selection. The Bayesian view is discussed in the next lecture. Though the Bayesian view is elegant, the frequentist view is much more practical.

6.1 Model selection

Today's goal is to figure out how to find models that will perform well on new data.

6.1.1 Examples of challenges in model selection

Example 6.1 (Sunspots and Republicans)**.** We consider using complex bases to fit the data. However, by making bases too complex, we will overfit to the training data and models that did well on our training data will not likely extend well to future data.

We need a model selection process to help us distinguish between truly good models that generalize to new data and bad models.

Example 6.2 (High dimensionality)**.** We can consider the number of dimensions that is much higher than the number of data points. Specifically, we have $x_{d,n}$ where d the dimension. Let $d = 2000, n = 8$ and the first dimension is a perfect predictor: $x_{1,n} = 1 \implies y_1 = 1$. Our model should ideally recreate this.

Hoewever, if dimensions are large, there may be another dimension in x that also perfectly predicts y, which makes it impossible to predict. This is another situation where our model selection process helps us determine whether models will or will not generalize.

6.2 Checking for generalizability

There are two methods to check for generalizability.

6.2.1 Using train, validation, and test split

Let us have a full dataset. We can split it into three parts randomly. We assume here that the data comes from the same distribution.

Note. There are more complicated things we can do, like splitting temporally, training on past data and validating on more present data. For the purpose of this class, we will consider random splits.

Training split. We can train all of our models on this part. For example, we can try all of our linear regression models with different bases on these models.

Validation set. We will evaluate each model here. This is the set of data where we can choose our best performing model.

Test set. We can get the final measure of quality of the best model we selected. We need to test again because in the validation set, our selected model could have just been lucky and done really well on our validation set. This is the safest way to test the models to evaluate the true accuracy of the final model.

If our model doesn't do well on the test, we can keep extra test sets if we have enough data. If there is not enough data, we must do *cross validation*.

6.2.2 Cross validation

We may find ourselves in situations where we cannot split the data into three parts. We can do *cross validation* in this case. This means that we split the data randomly into parts. For example, suppose we have five parts. We will train our model on all but one of the parts. We can train on parts 2-5 and validate on part 1. Next, we train the data on parts 1, 3-5 and validate on part 2, and we continue the pattern for each of the five parts. This tends to well in practice.

6.3 Bias-variance tradeoff

Here, we will discuss why models fail.

Generally, as a model size (complexity) gets bigger, the bias goes down, because we get a greater ability to fit our model onto the training set. The potential for a strangely specific curve to show up in a more complicated model increases; in other words, as the model size increases, the variance tends to go up.

There is a sweet spot that minimizes overall test error (which is made of bias and variance). We want a model that is at the minimal test error.

Note. The model size is roughly the same as the number of parameters. This is a reasonable proxy, but some parameters may not be independent of each other.

Note. We assume that the optimization is not an issue. There is one exception to the bias-variance tradeoff, i.e. certain neural networks.

We can consider the equation for expected least square error on some new unseen sample:

$$E_y[(y-\hat{y})^2] = E[(y-f_{\mathcal{D}}(\boldsymbol{x}))^2] \tag{67}$$

where y is a function of $\boldsymbol{x}$. We can incorporate the mean $\bar{y}$ and write

$$E[(y-\bar{y}+\bar{y}-f_{\mathcal{D}}(\boldsymbol{x}))^2] = E[\underbrace{(y-\bar{y})^2}_{\text{true noise}}] + E[\underbrace{(\bar{y}-f_{\mathcal{D}}(\boldsymbol{x}))^2}_{\text{model error}}] + 2E[\underbrace{(y-\bar{y})}_{0}(\bar{y}-f_{\mathcal{D}}(\boldsymbol{x}))] \tag{68}$$

where the third term vanishes because $E[(y-\hat{y})] = 0$ and the two terms are independent. Now we can write

$$E[(\bar{y}-f_{\mathcal{D}}(\boldsymbol{x}))^2] = E[\underbrace{(\bar{y}-\bar{f}_{\mathcal{D}}(\boldsymbol{x}))^2}_{\text{bias squared}}] + E[\underbrace{(\bar{f}_{\mathcal{D}}(\boldsymbol{x})-f_{\mathcal{D}}(\boldsymbol{x}))^2}_{\text{variance of model fit}}] + 2E[(\bar{y}-\bar{f}_{\mathcal{D}}(\boldsymbol{x}))\underbrace{(\bar{f}_{\mathcal{D}}(\boldsymbol{x})-f_{\mathcal{D}}(\boldsymbol{x}))}_{0}] \tag{69}$$

where $\bar{f}_{\mathcal{D}}(\boldsymbol{x}) = E_{\mathcal{D}}[f_{\mathcal{D}}(\boldsymbol{x})]$. We can describe this by consider a bunch of data sets and doing this experiment one million times. $\bar{f}$ represents if we took 100 data points, fit it, took 100 more data points, fit it again, and averaged all the models.

Note. $\bar{f} \neq \bar{y}$. If we had a linear model, the average model would be a line but if our true y was a parabola, then $\bar{f} \neq \bar{y}$.

The first term is the bias squared. It shows that if we have the wrong model class, we will suffer an error no matter how good everything else is. *Underfit* models have errors due to bias.

The second term is the variance of model fit. If all models are very far from the mean, it will contribute to the error term here. This means that the model is *too complicated*. If the model is too complex, it can *overfit* because we will overfit to the data. Overfit models have error due to variance.

The third term vanishes by the same argument above. Overall, we have

$$E[(\bar{y}-\hat{y})^2] = \text{noise} + \text{bias}^2 + \text{variance}. \tag{70}$$

Note. Only the bias and variance are controllable by our model.

6.3.1 Managing the bias-variance tradeoff

We can use regularization, for example ridge regression w^2 or lasso regression $|w|$. We can add terms to the loss:

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{w}) + \lambda R(\boldsymbol{w}) \tag{71}$$

This makes the model class smaller. We are basically saying only a few dimensions matter and we ask our model to pick out one or two important covariates. Conceptually, we are trying to reduce variance via a *soft penalty* on more complex models.

We can also *ensemble*. We can do classification by committee, random forest, bagging, extra random forest, gradient boost, bootstrapping, etc. We can take many fits and then take the average of predictors. We are effectively reducing the variance since we are taking an average — without affecting the bias.

7 February 15th, 2022

Announcements

- Get started on homework 2! It is due *next Friday*
- Today's relevant textbook sections are 2.8 and 2.9

Today, we will continue to discuss model selection. Last time, we were concerned with the question: given a situation, how to choose the *best* model.

Example 7.1 (2008 mortgage crisis)**.** In 2008, banks wanted to know how risky loans were.

Note. We note that today's lecture will not be practical. Last lecture was practical — the bias variance trade-off is very importantly conceptually, and cross validation is important for small data sets.

Today's lecture is not as practical, but mathematically elegant.

7.1 Bayesian models

Before, we assumed that $\boldsymbol{y}$ was determined by some $\boldsymbol{x}$ and some global parameters $\boldsymbol{w}$. For example, linear regression produces a prediction $\hat{y}$ from some corresponding $\boldsymbol{x}$ and some weights $\boldsymbol{w}$.

Today, the difference is that our global parameters are no longer fixed. We imagine that $\boldsymbol{w}$ are random variables. This is the *key idea*.

If this is true, we cannot solve for a single value. In particular, this gives us a framework for the following:

1. Computing the posterior: $p(\boldsymbol{w}|\mathcal{D})$, i.e. what models are likely given the data.
2. Computing the posterior predictive: $p(y^*|\boldsymbol{x}^*, \mathcal{D})$, i.e. what output is likely given the data
3. Computing the probability of the data: $p(\mathcal{D}) = p(Y|X)$.

Note. The last part is the model selection. We want to select a class of models for which Y is likely given X.

The key idea in Bayesian thinking is that we are always going to be marginalize over the random variables.

7.1.1 Posterior over parameters

We will simply apply Bayes' rule:

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(Y|X, \boldsymbol{w})p(\boldsymbol{w}|X)}{p(Y|X)} = \frac{p(Y|X, \boldsymbol{w})p(\boldsymbol{w})}{p(Y|X)} \tag{72}$$

where X, Y are the data. We call $p(\boldsymbol{w}|X)$ the prior and we assume that the probability of the weights is independent of the data because the data is fixed.

Example 7.2 (Flipping coins)**.** Consider flipping N coins where $\mathcal{D} = x_1, \cdots, x_N$ where $x_i \in \{0, 1\}$. The likelihood of the dataset is

$$p(\mathcal{D}|\theta) = \theta^{N_H}(1-\theta)^{N_T} \tag{73}$$

where θ is the unknown fairness of the coin, N_H, N_T are the number of heads and number of tails.

A frequentist approach where θ is unknown but fixed, we would choose the value

$$\theta_{\text{MLE}} = \frac{N_H}{N_H + N_T} \tag{74}$$

that maximizes the likelihood.

From a Bayesian perspective, θ is a random variable and we must estimate the posterior given the coin flips we have seen. We need to decide on our prior beliefs by deciding a prior distribution on θ. We assume $\theta \sim \text{Beta}(\alpha, \beta)$. Then $p(\theta) = \theta^{\alpha-1}(1-\theta)^{\beta-1}$. We can calculate the posterior

$$\begin{aligned} p(\theta|\mathcal{D}) &= p(\mathcal{D}|\theta)p(\theta) \\ &= \theta^{N_H}(1-\theta)^{N_T}\theta^{\alpha-1}(1-\theta)^{\beta-1} = \theta^{H_N+\alpha-1}(1-\theta)^{N_T+\beta-1} = \theta^{\alpha'-1}(1-\theta)^{\beta'-1} \end{aligned} \tag{75}$$

where $\alpha' \equiv \alpha + N_H, \beta' \equiv \beta + N_T$.

Note. We note that the beta distribution is a conjugate prior for the Bernoulli distribution.

In Bayesian probability theory, if the posterior distribution $p(\theta|X)$ is in the same probability distribution family as the prior probability distribution $p(\theta)$, the prior and posterior are then called *conjugate distributions*, and the prior is called a *conjugate prior* for the likelihood function $p(X|\theta)$.

7.1.2 Predictive posterior

Now that we have a posterior, we want to be able to use this posterior $p(\boldsymbol{x}|X)$ to predict things.

We saw that we can choose parameters to optimize our data likelihood $p(\mathcal{D}|\theta)$. We see that

$$\theta_{\text{MLE}} \in \operatorname*{argmax}_{\theta} p(\mathcal{D}|\theta) = \frac{N_H}{N_H + N_T}. \tag{76}$$

We can use the posterior $p(\theta|\mathcal{D})$ by taking the single most likely value of θ to make our decision:

$$\theta_{\text{MAP}} \in \operatorname*{argmax}_{\theta} p(\theta|\mathcal{D}) = \frac{N_H + \alpha - 1}{N_H + N_T + \beta + \alpha - 2}. \tag{77}$$

However, any true Bayesian will use $p(\theta|\mathcal{D})$ to average over all possible models θ.

$$p(x^* = 1|x_1, \cdots, x_N) = \int_\theta d\theta \; p(x^* = 1|\theta)p(\theta|\mathcal{D}). \tag{78}$$

The above is a general equation. We will now apply this to our specific example.

$$p(x^* = 1|x_1, \cdots, x_N) = \int_\theta d\theta \; p(x^* = 1|\theta)p(\theta|\mathcal{D}) = \int d\theta \; \theta p(\theta|\boldsymbol{x}) = E_{p(\theta|\mathcal{D})}[\theta] = \frac{\alpha + N_H}{N_H + N_T + \alpha + \beta}. \tag{79}$$

Note. In this example, $p(\boldsymbol{x}|\theta)$ simplified cleanly. Note this is not an estimate for a single θ, we *averaged out* the θ. We get the posterior predictive by taking an integral over the unknown parameter.

Note. We also note that as $|\mathcal{D}|$ grows, these prediction techniques approach the same predictions.

7.1.3 Prior selection

We can also consider different values of α, β that give us a different posterior prediction. We then ask what is the best way to select our prior?

In the Bayesian setting, model selection corresponds to model class selection and prior selection.

For example, when we fit data, we can select what curve to fit and what prior to put on the parameters on the curve. There is no one single correct line or parabola since we will average all possible ones equally.

We can select between two models by calculating the likelihood of the data $p(\mathcal{D})$ for different model classes:

$$p(\mathcal{D}) = \int_{\boldsymbol{w}} d\boldsymbol{w} \; p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}). \tag{80}$$

If we are considering too many possible models $\boldsymbol{w}$, then $p(\boldsymbol{w})$ is spread out an the integral will be very low. This is how the Bayesian approach penalizes overly complicated models. However, if $p(\boldsymbol{w})$ doesn't contain good models, then $p(\mathcal{D}|\boldsymbol{w})$ will be low and the integral will still be low.

8 February 17th, 2022

Announcements

- Today's relevant textbook sections are 4.4-4.6
- Problem sets will moved from 8 PM to 11:59 PM
- There will be review sessions for the midterm coming up!
- The material will cover up to next lecture's lecture
- One cheat sheet front and back is allowed, including practice questions and topics on the midterm and not on the midterm

8.0.1 Deep learning today

Deep learning is increasingly commonplace in our lives: auto-completing words on phones, auto-focusing on faces, Google translate. They are also becoming more inclusive.

Deep learning models are entirely enabled by the massive amount of data. The importance of data for deep learning implies that deep learning works well as an interpolator, but not an extrapolator. Deep learning uses nonlinear functions.

The ideas for deep learning models have existed since the 1990s. However, it has only been able to take off because of computing power and GPU technology.

It is important to think about the architectures and optimization methods.

8.1 Deep learning models

Two models we are *not* going to talk about are gradient boosted trees and extra random forests. These fall into the class of ensemble models and methods. These are models where we train a bunch of models and take an average of some form.

Let us consider logistic regression. Recall that in probabilistic classification, we have

$$p(y=1|\boldsymbol{x}) = \sigma(-(\boldsymbol{w}^\intercal \boldsymbol{x} + \boldsymbol{b})) = \frac{1}{1+\exp\left[-(\boldsymbol{w}^\intercal \boldsymbol{x} + \boldsymbol{b})\right]}. \tag{81}$$

Given a slope and offset $\boldsymbol{w}, \boldsymbol{b}$, $\boldsymbol{w}^\intercal \boldsymbol{x} + \boldsymbol{b} = 0$ marks a line where $p(y=1|\boldsymbol{x}) = 0.5$. This means the boundary in logistic regression is always linear. We need a more *expressive* model class to draw non-linear boundaries to fit the structure of non-linearly separable data.

One thing we can do is use a basis $\phi : \mathbb{R}^D \to \mathbb{R}^{D'}$ to transform $\boldsymbol{x}$. This would solve the problem by allowing us to achieve non-linear boundaries, but we don't always know what basis ϕ to use. One fundamental idea of deep learning is to *learn the basis*.

Note. Deep learning is considered *adaptive basis regression* in the statistics community.

For notation, let $f = \boldsymbol{w}^\intercal \boldsymbol{\phi} + \boldsymbol{b}$ and $p(y=1|\boldsymbol{x}) = \sigma(-f)$. The big question is finding the form of ϕ.

When we talk about *architecture* of a neural network, we are talking about the structure and form of ϕ. One option is to have data point $\boldsymbol{x} \in \mathbb{R}^D$ as inputs. We can define

$$\phi \equiv \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \tag{82}$$

as our basis transformed data where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{J\times D}, \boldsymbol{b}^{(1)} \in \mathbb{R}^J$. These parameters map $\boldsymbol{x}$ from a D to J dimensional vector ϕ. The superscripts indicate that these weights are done in the first step or *layer*.

8.1.1 Expressiveness

This is an expressive model class. As $J \to \infty$, this becomes a universal function approximator. That means we can express any f.

For some intuition, consider the $J = 2$ scenario and picking a sigmoid as our activation function. Essentially, we can build a staircase of sigmoids. When we add more and more sigmoids, which we get when adding more lodes in the hidden layers, we can represent increasingly complicated functions.

Note. Certain functions may be hard to fit — for example, polynomials may be hard to fit with this stacked sigmoid. This relates to the *inductive bias* of the architecture. By making this bias (model class) choice, it makes it hard for us to do certain things.

8.1.2 Ease of computation

Another reason we can choose this architecture is because it is computationally easy. Computing ϕ requires matrix multiplication and addition. Computers are good at this and this ease facilitates prediction and optimization.

We can also change the non-linear function we use. We call these functions *activation functions*. We can use the tanh or rectified linear unit ReLU:

$$\text{ReLU}(x) = \max(0, x). \tag{83}$$

The sigmoid function might have had an easy time representing something like stairs and radial basis activation might have a good time representing wavey functions. ReLU may allow us to make jagged functions. The inductive bias is our choice of activation function and how that impacts what types of functions we can model.

Note. In classification, we will always need a softmax at the end to ensure a proper probability distribution.

8.2 Additional architecture

We can add more layers. Consider if we had L *hidden layers*. We can say

$$\boldsymbol{\Phi}^{(1)} = a(\boldsymbol{W}^{(0)}\boldsymbol{X} + \boldsymbol{b}^{(0)}), \qquad \boldsymbol{\Phi}^{(\ell)} = a(\boldsymbol{W}^{(\ell-1)}\boldsymbol{X} + \boldsymbol{b}^{(\ell-1)}) \tag{84}$$

for $1 < \ell \leq L$. Then our output is in terms of our last hidden layer $\Phi^{(L)}$. In classification, this looks like

$$\boldsymbol{O} = \text{softmax}(\boldsymbol{W}^{(L)}\boldsymbol{\Phi}^{(L)} + \boldsymbol{b}^{(L)}). \tag{85}$$

This is called a *fully connected, feed-forward network*.

We note that multiple layers allow for feature reuse which allows us to model more easily.

8.3 More architectures

We discussed some fully-connected deep learning networks. We will now discuss different architectures.

8.3.1 Convolutional neural networks

We can consider image recognition and a filter applied to parts of an image to detect certain aspects. We can use the filter to scan across the image to produce a new image that consists of filter values. We can then apply a *pooling step*, where take a max value in parts of the image. We can then apply a feed-forward neural network to produce a function f.

Convolutional approaches let us discover motifs in our image. This is the way modern computer vision works.

Example 8.1. We can learn to find a ball in an image this way because learning an image once in a particular region can be applied elsewhere in the image.

8.3.2 Recurrent neural networks

If we suppose that our data is time-dependent, we can apply different models to different columns of $\boldsymbol{X}$. We can pick this type of model if we believe that the way to process data is sequential updates along the D dimensions of data.

9 February 22nd, 2022

Announcements

- Homework 2 is due Friday
- Midterm 1 is next Tuesday
- We will have our first ethics module on March 3

9.1 Optimizing a neural network

Today, we will continue our discussion of model classes. Last time, we talked about neural network architectures. Today, we discuss how neural networks can be optimized (involving a lot of chain rule).

We will use σ as our activation function in lecture but we can replace this with tanh or ReLU as well.

To optimize a neural network, we want to choose parameters $\boldsymbol{W}$ that minimize the network's loss. We can find parameters using gradient descent but we first need to calculate the gradients of network loss with respect to the parameters:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}. \tag{86}$$

For some input $\boldsymbol{x}_n$, the loss function $\mathcal{L}$ measures the difference between the true value $\boldsymbol{y}_n$ and the value f_n our model predicts for $\boldsymbol{x}_n$. We will use the chain rule because the loss function depends on f_n:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \frac{\partial \mathcal{L}}{\partial f_n}\frac{\partial f_n}{\partial \boldsymbol{W}}. \tag{87}$$

Note. f_n here is the same as $\hat{y}_n$, which is the model prediction.

9.1.1 Loss function for regression

We have used the least squares loss for regression:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\sum_n (y_n - f_n)^2 \implies \frac{\partial \mathcal{L}}{\partial f_n} = -\sum_n (y_n - f_n) \tag{88}$$

Note. If we swapped $\mathcal{L}$ for another loss function, the only change to $\partial_{\boldsymbol{W}}\mathcal{L}$ is $\partial_{f_n}\mathcal{L}$. $\partial_{\boldsymbol{W}} f_n$ stays the same.

9.1.2 Loss function for classification

For classification, a good function choice is logistic loss. For binary classification, we define

$$p(y=1|\boldsymbol{x}) = \sigma(f_n) = \frac{1}{1+e^{-f_n}}, \qquad p(y=0|\boldsymbol{x}) = 1-\sigma(f_n) = \frac{1}{1+e^{f_n}}. \tag{89}$$

We can define the logistic loss function using the above terms:

$$\mathcal{L}(\boldsymbol{w}) = \sum_n y_n \log p(y=1|\boldsymbol{x}) + (1-y_n)\log p(y=0|\boldsymbol{x}) = \sum_n y_n \log \sigma(f_n) + (1-y_n)\log\left(1-\sigma(f_n)\right). \tag{90}$$

Recall

$$\partial_z \sigma(z) = \sigma(z)(1-\sigma(z)) \tag{91}$$

and

$$\partial_{f_n}\mathcal{L}_n = y_n\frac{1}{\sigma(f_n)}\sigma(f_n)(1-\sigma(f_n)) + (1-y_n)\frac{1}{1-\sigma(f_n)}\sigma(f_n)(1-\sigma(f_n))(-1)$$
$$= y_n(1-\sigma(f_n)) - (1-y_n)\sigma(f_n) \quad (92)$$

We can calculate this if we know what f_n was. To get the gradient, we will use our current weights $\boldsymbol{W}$ to calculate f_n in a *forward pass*, and use f_n to find the gradient using a *backward pass* to update our step along the gradient.

Note. We note that for a forward pass, we start with input $\boldsymbol{x}_n$ and move forward through layers of the model to figure out the output f_n. In the backward pass, the values we calculated flow backwards and are used in chain rule products. The chain rule products become the gradients we are looking for.

9.2 Vector chain rule

We will take a detour to gain tools needed to do back propagation.

Consider if we had nested functions

$$y = f(u) = f(g(h(x))), \qquad u = g(v) = g(h(x)), \qquad v = h(x) \quad (93)$$

9.2.1 Scalar values

If we had scalars, we can simplify and apply the chain rule:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}. \quad (94)$$

9.2.2 $\boldsymbol{x}$ a size D vector

Given $\boldsymbol{x}$ a dimension D vector we note that

$$\boldsymbol{\nabla}_{\boldsymbol{x}} y = \frac{\partial y}{\partial u}\frac{\partial u}{\partial v}\boldsymbol{\nabla}_{\boldsymbol{x}} v \quad (95)$$

where $\partial_u y, \partial_v u$ are scalars and $\boldsymbol{\nabla}_{\boldsymbol{x}} v$ is a $1 \times D$ vector describing how each dimension of $\boldsymbol{x}$ affects the scalar v.

Note. We are using numerator layout notation, known as the *Jacobian formulation* where $\boldsymbol{y}, \boldsymbol{m}$ are size M and N vectors. Then $\partial_{\boldsymbol{x}}\boldsymbol{y}$ is a $M \times N$ matrix with

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \frac{\partial y_M}{\partial x_2} & \cdots & \frac{\partial y_M}{\partial x_N} \end{bmatrix}. \quad (96)$$

$\partial_{\boldsymbol{x}}\boldsymbol{y}$ is the *Jacobian* matrix, denoted $\mathbb{J}_{\boldsymbol{x}}[\boldsymbol{y}]$ and the elements tell us how each element of $\boldsymbol{y}$ depend on each element of $\boldsymbol{x}$.

9.2.3 $\boldsymbol{x}, \boldsymbol{v}$ size D, J vectors

We have

$$\boldsymbol{\nabla}_{\boldsymbol{x}} y = \frac{\partial f}{\partial u}\boldsymbol{\nabla}_{\boldsymbol{v}} u \mathbb{J}_{\boldsymbol{x}}[\boldsymbol{v}] \quad (97)$$

where $\partial_u y$ is a scalar, $\boldsymbol{\nabla}_{\boldsymbol{v}}\boldsymbol{u}$ is a $1 \times J$ vector and $\mathbb{J}_{\boldsymbol{x}}[\boldsymbol{v}]$ is a $J \times D$ matrix.

9.2.4 $\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{u}$ size D, J, J' vectors

In this case

$$\boldsymbol{\nabla}_{\boldsymbol{x}} y = \boldsymbol{\nabla}_{\boldsymbol{u}} y \mathbb{J}_{\boldsymbol{v}}[\boldsymbol{u}] \mathbb{J}_{\boldsymbol{x}}[\boldsymbol{v}] \tag{98}$$

where $\boldsymbol{\nabla}_{\boldsymbol{u}} y$ is a $1 \times J'$ scalar, $\mathbb{J}_{\boldsymbol{v}}[\boldsymbol{u}]$ is a $J' \times J$ matrix, $\mathbb{J}_{\boldsymbol{x}}[\boldsymbol{v}]$ is a $J \times D$ matrix. We get a vector of how each $\boldsymbol{x}$ dimension changes scalar y.

9.3 Finishing optimization

Recall previously we had

$$\frac{\partial \mathcal{L}_n}{\partial \boldsymbol{w}} = \frac{\partial \mathcal{L}_n}{\partial f_n} \frac{\partial f_n}{\partial \boldsymbol{w}}. \tag{99}$$

We seek to calculate $\partial_{\boldsymbol{w}} f_n$. Suppose $f_n = \boldsymbol{w}^\intercal \boldsymbol{\phi}_n + \boldsymbol{b}$. Then we have

$$\frac{\partial f_n}{\partial \boldsymbol{w}} = \boldsymbol{\phi}_n. \tag{100}$$

Example 9.1 (Regression case)**.** We consider the regression case where we have $\partial_{f_n} \mathcal{L}_n = (y_n - f_n)(-1)$ and $\partial_{\boldsymbol{w}} f_n = \boldsymbol{\phi}_n$ so we have

$$\frac{\partial \mathcal{L}_n}{\partial \boldsymbol{w}} = (y_n - f_n)(-1)\boldsymbol{\phi}_n \tag{101}$$

and notice that we still need f_n to compute the gradient. We get this term by computing $\boldsymbol{x} \to \boldsymbol{\phi}_n \to \boldsymbol{f}_n$ and use the value of $\boldsymbol{f}_n$ in the gradient.

We recall that $\boldsymbol{\phi}_n = \sigma(\boldsymbol{W}^1 \boldsymbol{x} + \boldsymbol{b}^1$. We can get $\boldsymbol{\nabla}_{\boldsymbol{W}} \mathcal{L}_n$ using

$$\boldsymbol{\nabla}_{\boldsymbol{W}} \mathcal{L}_n = \frac{\partial \mathcal{L}_n}{\partial f_n} \boldsymbol{\nabla}_{\boldsymbol{\phi}_n} f_n \mathbb{J}_{\boldsymbol{W}'}[\boldsymbol{\phi}_n] \tag{102}$$

9.4 Generalization

The process of computing gradients for optimizing neural networks is:

- Compute $\mathcal{L}(\phi^L(\phi^{L-1}(\cdots \phi^1(\boldsymbol{x}))))$ by passing $\boldsymbol{x}$ through the network $\boldsymbol{x} \to \phi^1 \to \cdots \to \phi^L \to \mathcal{L}$ in a *forward pass* and store all ϕ^ℓ.
- Perform the chain-rule backwards:

$$\frac{\partial \mathcal{L}}{\partial \phi^\ell} = \boldsymbol{\nabla}_{\phi^L} \mathcal{L} \mathbb{J}_{\phi^{L-1}}[\phi^L] \cdots \mathbb{J}_{\phi^\ell}[\phi^{\ell+1}]. \tag{103}$$

  We can then compute the parameter gradients

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^\ell} = \boldsymbol{\nabla}_{\phi^L} \mathcal{L} \mathbb{J}_{\phi^{L-1}}[\phi^L] \cdots \mathbb{J}_{\phi^\ell}[\phi^{\ell+1}] \mathbb{J}_{\boldsymbol{W}^\ell}[\phi^\ell] = \frac{\partial \mathcal{L}}{\partial \phi^\ell} \frac{\partial \phi^\ell}{\partial \boldsymbol{W}^\ell}. \tag{104}$$

Note. This is called *reverse-mode autodiff* or *back propagation*. This is good when y is small and there are many parameters $\boldsymbol{W}$. We can also do *forward-mode diff*. This goes from variable $\boldsymbol{x}_{nd} \to \mathcal{L}_n$ rather than $\mathcal{L}_n \to \boldsymbol{x}_{nd}$. We compute $\partial_{\boldsymbol{x}_{nd}} \phi^1$ and then $\partial_{\phi^1} \phi^2$ and all the way up.

10 February 24th, 2022

Announcements

- Homework 2 is due tomorrow
- Homework 3 is released tomorrow
- Next week is a break! Midterm 1 is on Tuesday, and our ethics module will be on Thursday.
- Today's relevant textbook sections are 5.1-5.3

Today, we discuss support-vector machines.

10.0.1 Introduction and motivation

support vector machines were the workhorse of machine learning. For example, we can use an SVM to sort cells into different types. Another example is the Casella plant that sorts trash and recycling. We can manually sort the recycled items, or use SVMs. There is a conveyor belt that takes a picture of the plastics that move through and using the SVM, we can sort the type of plastic and send it down the right belt.

Note. SVMs are mostly used in classification problems.

10.1 Max margin

The *max margin* is an objective function that is used in SVMs.

Note. Because this is an objective function, we can use this in other techniques; there is a family of techniques called margin methods.

In the past, our objective functions included negative log likelihood, squared loss, 0/1 loss, hinge loss, log loss, etc. For classification, we wanted to start with 0/1 loss but this is difficult to optimize, so we made proxies. Now we define the max margin objective.

Definition 10.1 (Margin)**.** The *margin* on a correctly classified example is the absolute normalized orthogonal distance to the boundary. The *margin on the data* is the minimum margin on correctly labeled examples.

Our goal is to find the separator that *maximizes the margin of the data*.

We can consider models that are indifferent between different boundaries. We seek to quantitatively describe our heuristic choice.

10.2 Hard margin SVM

We can consider a binary classification setting where $\hat{y} = \{0, 1\}$. Moreover, we focus on the linear case where

$$f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^\intercal \boldsymbol{x} + w_0. \tag{105}$$

We will also assume that the data are separable. This specifies the *hard margin* formulation. This hard margin SVM forces all points to be classified correctly.

10.2.1 Geometric intuition

We can consider a boundary $\boldsymbol{w}^\intercal \boldsymbol{x} + w_0 = 0$ and $\boldsymbol{w}^\intercal \boldsymbol{x} + w_0 = c$ a parallel line. We can pick an $\boldsymbol{x}_n$ and decompose $\boldsymbol{x}_n = \boldsymbol{x}_p + (\boldsymbol{x}_n - \boldsymbol{x}_p)$, where $\boldsymbol{x}_n - \boldsymbol{x}_p \perp \boldsymbol{w}^\intercal \boldsymbol{x}$. Note that $\boldsymbol{x}_n - \boldsymbol{x}_p || \boldsymbol{w}$ so we can write

$$\boldsymbol{x}_n = \boldsymbol{x}_p + r \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}. \tag{106}$$

We can do some algebra and obtain

$$\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0 = \underbrace{\boldsymbol{w}^\intercal \boldsymbol{x}_p + w_0}_{=0} + r\frac{\boldsymbol{w}^\intercal \boldsymbol{w}}{\|\boldsymbol{w}\|_2} \implies r = \frac{\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0}{\|\boldsymbol{w}\|_2} \tag{107}$$

where r is the *signed distance*. We can multiply by y_n to get the unsigned distance, which is the margin.

Note. We assume that everything is correctly classified, so the margin is always positive.

Then we have

$$\boxed{\text{margin}(\boldsymbol{x}_n|\boldsymbol{w}, w_0) = y_n r, \qquad r \equiv \frac{\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0}{\|\boldsymbol{w}\|_2}.} \tag{108}$$

Note. The equation above without normalization is called the *functional margin*. The margin is invariant to multiplying the weights by a scale factor $\beta > 0$. The margin is also negative on a misclassified example.

We want the minimum margin across all data, and we want to maximize the whole term. Thus we want

$$\boxed{\max_{\boldsymbol{w}, w_0} \min_{\boldsymbol{x}_n} \frac{1}{\|\boldsymbol{w}\|_2} y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0).} \tag{109}$$

This finds a separator if our data is separable.

10.2.2 Overparameterization and simplify objective

There are too many parameters in the above expression, where the extra parameters are meaningless.

Note. Consider $\boldsymbol{w}^\intercal, w_0$, we can substitute instead the scale factors. Any scale factor of $\boldsymbol{w}^\intercal, w_0$ works because they are all perpendicular to the same boundary.

We can then choose our own scale factors with

$$y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1 \tag{110}$$

and our objective becomes

$$\max_{\boldsymbol{w}, w_0} \frac{1}{\|\boldsymbol{w}\|_2} \min_n y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) : \qquad y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1. \tag{111}$$

Note (Some intuition)**.** We know that we can scale the weights and the results are the same. We want $\boldsymbol{w}$ to be small because we are maximizing $1/\boldsymbol{w}$ and minimize something linear in $\boldsymbol{w}$. Thus the minimum condition is redundant and we simply want

$$\boxed{\min_{\boldsymbol{w}, w_0} \|\boldsymbol{w}\|_2^2 : \qquad y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1.} \tag{112}$$

The above equation is nice because it is convex with a quadratic objective and linear constraint.

We can solve this using `cvxopt`, `cvxpy`, `sklearn`.

10.3 Soft margin SVM

We now consider non-separable data. We can have a relaxed soft margin formulation. We can define how far the point is on the wrong side of the margin using slack variables.

Definition 10.2 (Slack variable)**.** We define ξ_n to be the *slack variable* for the nth data point. That is

$$\xi_n = \begin{cases} 0, & \text{correctly classified, outside margin} \\ 1 - y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0), & \text{incorrect, inside margin} \end{cases} \tag{113}$$

Now our new formulation of the equations are

$$\min_{\boldsymbol{w}, w_0} \frac{1}{\|\boldsymbol{w}\|_2^2} + C \sum \text{slack}_n \tag{114}$$

such that

$$y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1 - \xi_n, \qquad \xi_n \geq 0. \tag{115}$$

We are pretending that the margin is 1 and writing our constraint the same way, but allowing for some slack for misclassified points. If we make C large, we enforce the constraints more. We can set C using cross validation, but we can use `sklearn` to find it automatically. This is still convex, so we can still use solvers. We can write the equivalent formulation

$$\boxed{\min_{\boldsymbol{w}, w_0} \frac{1}{2} \boldsymbol{w}^\intercal \boldsymbol{w} + C \sum_n \max\{0, 1 - y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0)\}} \tag{116}$$

We note that this is a convex function and differentiable almost everywhere so we can optimize using stochastic gradient descent!

Note. The central idea is that the max margin loss starts penalizing you before we start being wrong and penalizes us even when we are correct.

11 March 1st, 2022

Today is the first midterm.

12 March 3rd, 2022

Today, we discuss ethics in machine learning.

We need to be careful about discriminatory stereotyping through data, such as was seen with redlining policies in the 1930s. In particular, data features that seem nondiscriminatory are often actually proxies for features like race and income.

Exclusion from a data set is another form of biased data. Consider a model trained on a medical database with a lot of data on white patients, but little data on people of color. This model may learn harmful associations that lead to poor care for patients of color, simply because they were not included in the data. For example it might make predictions as though "people of color are less likely to get sick because they don't come to this hospital much", which is clearly undesirable.

Question: Should organizations be prohibited from collecting data on race? This initially sounds like a way to prevent bias, but it also makes it harder to check our models and systems for bias.

12.1 Causal chains

We can represent decisions in a *causal chain* of events. There are causal chains where single or multiple agents can be *causally responsible* for the outcome.

We note that decisions contribute to the outcome where the decisions are *choice points*. These choice points are represented by ovals.

Some outcomes can be traced back to multiple choice points. Some decisions are causally relevant but not morally loaded. For example, choosing to rob a bank and choosing to drive the a blue or gray getaway car have different levels of moral relevance.

12.2 Moral responsibility

There is *backward-looking responsibility* and *forward-looking responsibility*. Backward-looking includes different actions the agent could have done. The agent may deserve blame, penalty, or retribution. Forward-looking responsibility for an agent is the responsibility to prevent such an outcome in the future.

Example 12.1. Suppose two agents decide to rob a bank. These two agents **should not have robbed a bank** and both deserve some retribution, *and* both agents are **obligated to refrain from robbing another bank**.

13 March 8th, 2022

Announcements

- Today's relevant textbook sections are 5.4
- Homework 3 is due Friday!
- Midterms have been graded
- Masks are still expected in lecture

Example 13.1 (Credit scores)**.** One interesting application of machine learning and ethics is defining scoring mechanisms for evaluating or predicting credit scores. We want to consider how transparent we want our parameters to be.

13.1 SVMs continued

Recall from last (last) week, we looked at the max margin problem. The idea is that if we have two clusters of points, there is some intuitive notion of *good* and *bad* boundaries — we pick the line that is far from the data. Assuming a *linear boundary*, we formalized this notion by defining the *margin*:

$$\text{margin}(\boldsymbol{x}_n|\boldsymbol{w}, w_0) = y_n r, \qquad r \equiv \frac{\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0}{\|\boldsymbol{w}\|_2}. \tag{117}$$

We recall our optimization problems that focus on maximizing the minimum margin. We had a *hard margin problem* where we wanted to minimize with respect to $\boldsymbol{w}, w_0$:

$$\min_{\boldsymbol{w}, w_0} \frac{1}{2}\|\boldsymbol{w}\|_2^2 : \qquad y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1, \tag{118}$$

which is a quadratic objective with linear constraints.

Note. A hard margin refers to separability of data.

We also have a *soft margin* version where we are *not* guaranteed separability. The objective is

$$\min_{\boldsymbol{w}, w_0} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum \xi_n : \qquad y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1 - \xi_n, \qquad \xi_n \geq 0 \tag{119}$$

where ξ_n is our *slack*.

Note. We can stick in a basis if we wanted to as well!

Today, we will discuss how SVM can be efficiently solved for high dimensional $\boldsymbol{x}$. Namely, we will solve the dual form of the max-margin function, which will allow us to rewrite the discriminant function and use the *kernel trick* to learn in high-dimensional bases.

13.2 Reframing the problem

We can reframe the minimization from last class. We can rewrite the hard-margin form as a Lagrangian. The original form is

$$\min_{\boldsymbol{w}, w_0} \frac{1}{2}\boldsymbol{w}^\intercal \boldsymbol{w} : \qquad y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1. \tag{120}$$

We can rewrite this as

$$\min_{\boldsymbol{w}, w_0}\left[\max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{w}, \alpha, w_0)\right] = \min_{\boldsymbol{w}, w_0}\left[\max_{\boldsymbol{\alpha}}\left\{\frac{1}{2}\boldsymbol{w}^\intercal \boldsymbol{w} - \sum_n \alpha_n\left[y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) - 1\right]\right\}\right] \tag{121}$$

with constraint $\alpha_n \geq 0$. We can think about the inner sum as softening the linear constraint $\mathbb{I}(y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) \geq 1)$.

We can verify that these formulations are equivalent by the following:

1. **Optimal solutions satisfy the same constraints.** If $y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) - 1 \geq 0$ is violated for some n, then $\alpha_n \left[y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) - 1\right] < 0$. Then because the sum is negated, the nth term is positive and the value approaches ∞ by setting α_n arbitrarily large. However, we want the max to be small, so a solution violating this constraint would *not be optimal* and *not be selected*.
2. **Optimal solution sets** $\alpha_n = 0 \forall \boldsymbol{x}_n : y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) > 1$**.** If the constraint is satisfied with slack, then the maximizer makes $\alpha_n = 0$. We want to keep positive terms so to maximize the entire constraint, we want to subtract nothing.
3. **Optimal solution satisfies** $\alpha_n \left[y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) - 1\right] = 0 \forall n$**.** For each n, we either have $\alpha_n = 0$ or $y_n(\boldsymbol{w}^\intercal \boldsymbol{x}_n + w_0) - 1 = 0$. The sum term being subtracted will be 0, leaving us with an expression that is the same as the former expression.

Note. At optimality, the constraints are satisfied, many $\alpha_n = 0$, and $\alpha \neq 0$ for the cases that are met with equality. These are places where points are close to the margin.

13.2.1 Strong duality

We will now apply strong duality to swap the max and the min.

We will first discuss *weak duality*. Our Lagrangian formulation reformulated the hard-SVM as a min-of-max problem. Swapping the max and the min leads to a smaller value. Intuitively, we had to pick $\boldsymbol{w}$ first and then pick the maximizing $\boldsymbol{\alpha}$.

But now, if we swap the max and min, we can pick any $\boldsymbol{w}$ we want after $\boldsymbol{\alpha}$ is set, giving the minimizer an advantage. Thus the minimizer will achieve a value *at least as small* as before.

$$\min_{\boldsymbol{w}, w_0} \max_{\boldsymbol{\alpha} \geq 0} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}, w_0) \geq \max_{\boldsymbol{\alpha} \geq 0} \min_{\boldsymbol{w}, w_0} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}, w_0). \tag{122}$$

This problem satisfies *strong duality* where the two sides are *equal*. We can thus switch the min and the max.

$$\min_{\boldsymbol{w}, w_0} \max_{\boldsymbol{\alpha} \geq 0} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}, w_0) = \max_{\boldsymbol{\alpha} \geq 0} \min_{\boldsymbol{w}, w_0} \mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}, w_0). \tag{123}$$

The idea is that the property holds because our objective is quadratic and the constraints are linear. This is easier for us to solve now because we can now rewrite the expression in terms of $\boldsymbol{\alpha}$ only. The optimal $\boldsymbol{w}, w_0$ satisfies

$$\nabla_{\boldsymbol{w}} \mathcal{L} = \boldsymbol{w} - \sum_n \alpha_n y_n \boldsymbol{x}_n = 0 \iff \boxed{\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n} \tag{$*$}$$

and

$$\nabla_{w_0} \mathcal{L} = \boxed{-\sum_n \alpha_n y_n = 0} \tag{$\square$}$$

as desired. Then we can expand terms in our reformulated objective:

$$\max_{\boldsymbol{\alpha}\geq 0}\min_{\boldsymbol{w},w_0}\left\{\frac{1}{2}\boldsymbol{w}^\intercal\boldsymbol{w}-\boldsymbol{w}^\intercal\sum_n\alpha_n y_n\boldsymbol{x}_n - w_0\sum_n\alpha_n y_n+\sum_n\alpha_n\right\}$$
$$=\boxed{\max_{\boldsymbol{\alpha}}\left\{-\frac{1}{2}\sum_n\sum_{n'}\alpha_n\alpha_{n'}y_n y'_n\boldsymbol{x}_n\boldsymbol{x}'_n+\sum_n\alpha_n\right\}} \tag{124}$$

with constraints $\boxed{\sum_n\alpha_n y_n=0,\alpha_n\geq 0}$ where we substituted $*$ and $\square$ and combined like terms.

This is our **dual hard-margin formulation of SVM**. There is also a soft-margin formulation where we introduce c with constraint $c\geq\alpha_n\geq 0$ and the discriminant function looks like

$$h(\boldsymbol{x},\boldsymbol{\alpha},w_0)=\sum_n\alpha_n y_n\boldsymbol{x}_n^\intercal\boldsymbol{x}+w_0 \tag{125}$$

with support vectors

$$Q=\{\alpha_n:\alpha_n>0\}. \tag{126}$$

The α_n are found by the optimization and w_0 is found by noting that $y_n(\boldsymbol{w}^\intercal\boldsymbol{x}_n+w_0)=1$ for points on the decision boundary. Thus can find some point $\boldsymbol{x}_n$ on the decision boundary and use this to find w_0.

13.3 Kernel trick

We can consider learning with a basis function $\phi:\mathbb{R}^D\to\mathbb{R}^M$. We need to solve

$$\max_{\boldsymbol{\alpha}}\left\{-\frac{1}{2}\sum_n\sum_{n'}\alpha_n\alpha_{n'}y_n y'_n\phi(\boldsymbol{x}_n)^\intercal\phi(\boldsymbol{x}'_n)+\sum_n\alpha_n\right\} \tag{127}$$

Note that in the objective, $\boldsymbol{x}$ only appears in the inner product and to predict a new $\boldsymbol{x}^*$, we use $\boldsymbol{w}^\intercal\boldsymbol{x}^*+w_0+\sum_n\alpha_n y_n\boldsymbol{x}_n^\intercal\boldsymbol{x}^*$ where the $\boldsymbol{x}^*$ appears only in the inner product.

We can thus replace all appearances of ϕ with a kernel function

$$K(\boldsymbol{x},\boldsymbol{z})=\phi(\boldsymbol{x})^\intercal\phi(\boldsymbol{z}). \tag{128}$$

This is because the ϕ never appears outside of the produce and we can computer the inner product K without computing $\phi(\boldsymbol{x}),\phi(\boldsymbol{z})$. We can thus take the distance between $\boldsymbol{x},\boldsymbol{z}$ in a high-dimensional basis without actually bringing $\boldsymbol{x},\boldsymbol{z}$ into the high dimension.

Example 13.2 (Quadratic and polynomial kernels)**.** Consider the kernel

$$K(\boldsymbol{x},\boldsymbol{z})=(\boldsymbol{x}^\intercal\boldsymbol{z})^2=(x_1^2,x_1x_2,x_2x_1,x_2^2)^\intercal(z_1^2,z_1z_2,z_2z_1,z_2^2)=\phi(\boldsymbol{x})^\intercal\phi(\boldsymbol{z}) \tag{129}$$

where ϕ maps into a basis using all degree-2 terms. For further generalization, we can consider

$$K(\boldsymbol{x},\boldsymbol{z})=(1+\boldsymbol{x}^\intercal\boldsymbol{z})^2 \tag{130}$$

which is a kernel for the degree-2 basis including linear and constant terms. For $q\geq 2$, we have

$$K_{\text{poly}}(\boldsymbol{x},\boldsymbol{z})=(1+\boldsymbol{x}^\intercal\boldsymbol{z})^q \tag{131}$$

which is a kernel for polynomials including all terms up to degree q.

Note. For polynomials degree q, the basis would have $O(D^q)$ terms. Without using the kernel trick, the computing power for higher bases grow at exponential rate.

Example 13.3 (Gaussian kernel)**.** Consider

$$K_{\text{Gauss}}(\boldsymbol{x}, \boldsymbol{z}) = \exp\left[-\frac{1}{\lambda}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2\right] \tag{132}$$

with bandwidth $\lambda > 0$, decays exponentially in squared distance. This corresponds to a basis of infinite dimension.

13.3.1 Valid kernels

One way to view a kernel K is as a $n \times n$ Gram matrix on the data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ where $K(\boldsymbol{x}_n, \boldsymbol{x}'_n) = a_{nn'}$.

At a high level, we consider Mercer's theorem that states that K is a valid kernel for some basis $\iff$ the Gram matrix defined by K is positive semidefinite.

14 March 10th, 2022

Announcements

- Relevant sections for today are chapter 6
- Homework 3 is due this Friday
- Homework 4 is released tomorrow, but is designed to take one week
- Spring break is next week!

14.1 Unsupervised learning

In the supervised case, we predict labels y given data $\boldsymbol{x}$. In the *unsupervised* case, we only have $\boldsymbol{x}$ and no targets. The goal is to *summarize* the data. This is helpful for

1. Figuring out what classes or labels to use later with data in a supervised learning model
2. Compressing high-dimensional data to lower dimensions
3. Organizing data, like grouping news articles covering the same topic or songs with the same style together

Today, we will focus on clustering that assumes a discrete structure that is nonprobabilistic. There are **two methods**.

Note. We note that there are many clustering model classes, discrete, and continuous.

14.2 k-means

We will first define our problem. We have data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and we want to find k clusters. We want an algorithm that outputs group assignments z_{nk} where z_{nk} tells us if $\boldsymbol{x}_n$ is in cluster k. That is

$$z_{nk} = \begin{cases} 0, & \boldsymbol{x}_n \text{ not in cluster } k \\ 1, & \boldsymbol{x}_n \text{ in cluster } k \end{cases}. \tag{133}$$

We need to find a way to measure how different two data points are. Different metrics include the Euclidean distance, edit distance for text, or Hamming distance. Today, we will use the Euclidean distance

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2. \tag{134}$$

For k-means clustering, we can define a *prototype* of cluster k denoted $\boldsymbol{\mu}_k$. This $\boldsymbol{\mu}_k$ defines the center point of cluster k.

14.2.1 The objective

We use the following objective for the k-means problem. We want to find the $\boldsymbol{\mu}$s and zs so that $\boldsymbol{x}_n$ are close to the centers of the clusters they are assigned to:

$$\min_{z,\boldsymbol{\mu}} \sum_n \sum_k z_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2^2. \tag{135}$$

This optimization is NP-hard and non-convex so it is difficult to solve. We can use Lloyd's algorithm to find a local optimum.

14.2.2 Lloyd's algorithm

We can find clustering assignments for k-means by the following:

1. Randomly initialize prototypes $\boldsymbol{\mu}_k$
2. Repeat until converged:
   (a) Assign each example to its closest prototype:

$$\boldsymbol{x}_n = \text{argmin}_k \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2 \tag{136}$$

   (b) For each cluster k, set $\boldsymbol{\mu}_k$ to the centroid (mean) of the examples assigned to this cluster

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_n z_{nk} \boldsymbol{x}_n \tag{137}$$

The results of the algorithm will depend on how the prototypes are initialized. We usually restart several times to find the best solution.

Note (Correctness)**.** On high level, each step reduces loss until convergence. This arrives at some kind of local minimum.

When we calculate the z's, we assign each point to the cluster with the closest prototype. This intuitively minimizes our loss because we are shortening distances.

For $\boldsymbol{\mu}$, we can take the derivative of our loss and set it to 0. Note that

$$\mathcal{L} = \min_{z,\boldsymbol{\mu}} \sum_n \sum_k z_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2^2 = \sum_n z_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\intercal (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \tag{138}$$

and

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_k} = -2 \sum z_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0 \implies \boldsymbol{\mu}_k = \frac{1}{N_k} \sum_n z_{nk} \boldsymbol{x}_n \tag{139}$$

which is what we set $\boldsymbol{\mu}_k$ to when running our algorithm. Thus we monotonically improve the loss.

Note (Selecting k)**.** We can find the number of clusters to pick by plotting the lowest loss achieved for different k and pick the k that corresponds to a bend in the loss curve. This is known as the *elbow method*.

Note (More comments about k-means)**.** k-means is a *parametric method* where parameters are the prototypes. This method is inflexible and the decision boundary is linear. The method is fast, update steps can be parallelized. There are many variations of the basic k-means algorithm:

1. k-means++ gives more specific ways to intializsed clusters
2. k-medoids chooses the center-most data point in the cluster as the prototype instead of the centroid

14.3 Hierarchical agglomerative clustering (HAC)

The HAC algorithm is as follows:

1. Every example starts in its own cluster
2. While there is more than 1 cluster, we merge the two *closest* clusters

This forms a hierarchy of clusters that can be visualized as a tree over the data where nodes are clusters. A node $\boldsymbol{x}$'s children represent the clusters that were at one point merged into node $\boldsymbol{x}$.

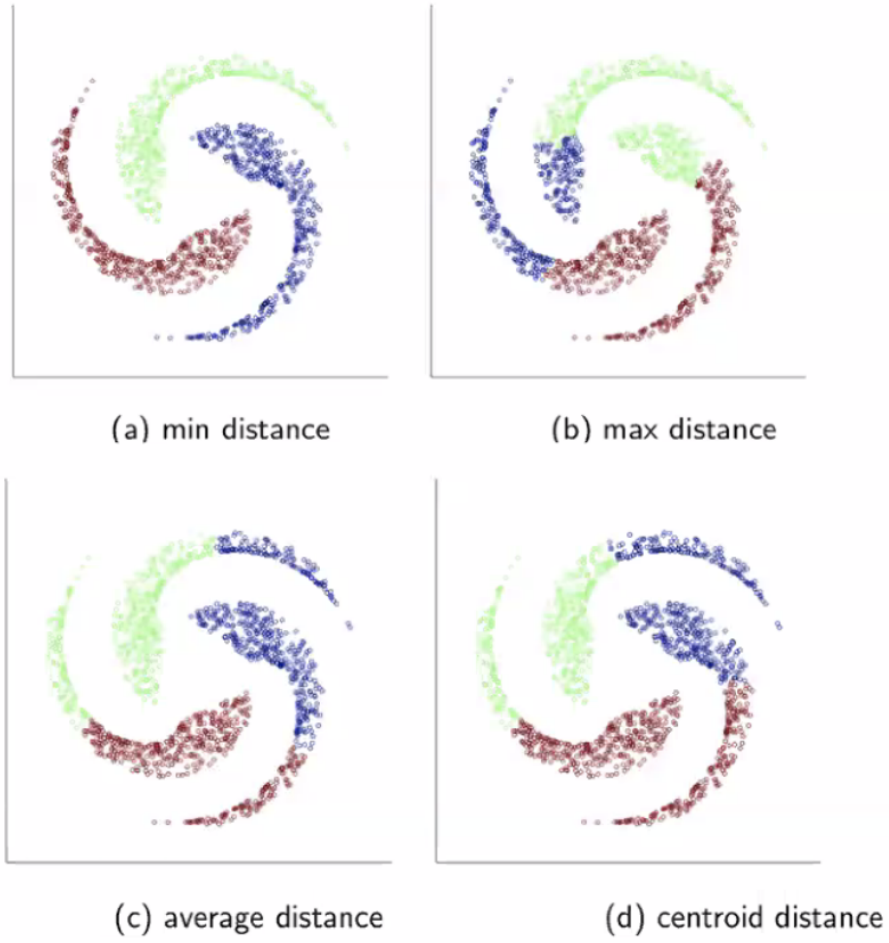
Some features include:

- HAC is non-parametric
- It is deterministic because there is no random initialization
- We do not need to specify the number of clusters
- Complexity scales as $O(Tn^2)$ where T is the number of iterations

We need two measures of distance. We note that $d(\boldsymbol{x}, \boldsymbol{x}')$ will measure the distance between individual points. We also need a *linkage function* that measures the distance between *two clusters of points*. This is how we determine what the *closest* pair of clusters are. Some options include:

- Minimum distance between two elements in the clusters
- Maximum distance between two elements of the clusters
- Average distance between elements in the different clusters
- Distance between the centroids of the clusters

Note. The min linkage is more likely to merge large clusters together. It will also tend to merge clusters into *chains* or *strings*. Max linkage prefers compact clusters instead. The average and centroid linkages are compromises between the min and max ones.



(a) min distance (b) max distance

(c) average distance (d) centroid distance

15 March 22nd, 2022

Announcements

- Problem set 4 is due Friday
- Come take a look at midterms, regrade requests are due Friday
- Relevant textbook sections are 9.1-9.5

15.1 Mixture models

Cube. Unsupervised, discrete, probabilistic.

A *mixture model* is unsupervised, discrete, and probabilistic model. It is in the same categories as clustering except for the fact that it is probabilistic. For intuition, instead of clustering data $\boldsymbol{x}$ into a final cluster z, we want to capture the fact that $\boldsymbol{x}$ my have a breakdown of probabilities that z may be a part of.

Example 15.1. Consider heights in a human population. The height of men and women are approximately Gaussian, with different means and variances. In an unsupervised setting, we get only heights and no the labels. We know that there is an underlying discrete variable (sex) that leads to two Gaussians, but putting them together leads to a funky looking distribution. We can deal with this using a mixture model, which gives us probabilities of being in different clusters (sexes).

15.1.1 Connection to generative classification

Generative classification is the same as mixture models in the sense that it's discrete and probabilistic, but generative classification is supervised whereas mixture models are unsupervised.

In generative classification, we told ourselves a story that y generates $\boldsymbol{x}$. In mixture models, we have a z (hidden variable because there are no y's) that produces our $\boldsymbol{x}$. We also have other parameters θ that affect our data $\boldsymbol{x}$ as well.

Thus we have $z_n \sim \pi, \boldsymbol{x}_n \sim p(\boldsymbol{x}_n|z_n;\theta)$.

In generative classification we did the following:

1. Solved for θ given data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$
2. Solved for y given θ and a specific $\boldsymbol{x}$.

In a mixture model, we only have $\boldsymbol{x}$ so the challenge is to solve despite only having $\boldsymbol{x}$.

Example 15.2 (Gaussian mixture model)**.** Consider what we did in problem set 2 problems 2-3. First we can set up the following:

- $p(z_n = k) = \pi_k$. This is categorical, analogous to representing three stars
- $p(\boldsymbol{x}|z = k) = \mathcal{N}(\mu_k, \Sigma_k)$. This is analogous to representing each stars mean and variance.

We can now calculate our log-likelihoods.

We can try **complete data log-likelihood** $\log p(\boldsymbol{x}, z|\theta)$. This is complete because it assumes we have our observation $\boldsymbol{x}$ and the class z that $\boldsymbol{x}$ came from.

Note. This does not work unless given z_{nk}.

We have

$$\log p(\boldsymbol{x}, z;\theta) = \sum_n \log p(\boldsymbol{x}_n, z_n|\theta) + \log p(z_n;\theta) = \sum_n \sum_k z_{nk} \log \mathcal{N}(\boldsymbol{x}_n;\mu_k,\Sigma_k) + z_{nk} \log \pi_k. \tag{140}$$

Note that if we are given z_{nk}, solving for the MLE of π, μ, Σ is easy, which is the same as generative classification. Without z_{nk}, this is hard and non-convex.

We can try **log-likelihood**. We can argue that we just need $\sum_n \log p(\boldsymbol{x}_n; \theta)$ and ignore the z's because the z's are just a way for us to summarize the data. WE can integrate out the zs and end up with

$$p(\boldsymbol{x}_n|\theta) = \sum_k \pi_k \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k). \tag{141}$$

Here, we are summing across all clusters and for each cluster, we are incorporating π_k by multiplying with $p(\boldsymbol{x}_n, z_n; \theta)$. We can then substitute it into the full term and obtain

$$\sum_n \log p(\boldsymbol{x}_n; \theta) = \sum_n \log \left(\sum_k \pi_k \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k) \right). \tag{142}$$

There are no analytic solutions here and the gradients are messy, so we need to try something else.

We can solve this problem with two different methods.

15.2 Max max

This looks like Lloyd's algorithm:

1. Start with z's randomly initialized
2. Find MLE of π, μ, Σ given z (this is possible in the complete data log-likelihood formulation)
3. Find best z given π, μ, Σ's that minimizes the loss.

15.3 Expectation maximization

This method approximates $\sum_n \log p(\boldsymbol{x}_n; \theta)$. Our goal is to maximize $p(\boldsymbol{x}|\theta)$.

We can set up the problem by noting that the lower bound on $\sum_n \log p(\boldsymbol{x}_n; \theta)$ is given by

$$\sum_n E_z \left[\log p(\boldsymbol{x}_n, z_n; \theta) \right]. \tag{143}$$

Optimizing the log likelihood means optimizing for the worst $p(\boldsymbol{x}|\theta)$ can be. We can expand the expectation and obtain

$$\sum_n E_z \left[\log p(\boldsymbol{x}_n, z_n; \theta) \right] = \sum_n \sum_k \left[q_{nk} \log \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k) + q_{nk} \log \pi_k \right], \qquad q_{nk} \equiv p(z_n = k|\boldsymbol{x}_n; \pi, \mu, \Sigma). \tag{144}$$

Intuitively, q is some arbitrary distribution over the k options for z.

Note. Given q_{nk}, we can find our parameters π, μ, Σ.

15.3.1 The EM algorithm

At a high level, we are going to randomly initialize our parameters and use those to calculate $q_{nk} = p(z|\boldsymbol{x}; \theta)$ by consider expectations across all z. Then we can find the MLE of parameters μ_k, Σ_k using the maximization step. Then with those parameters, we will repeat the expectation step and the maximization step.

Expectation step. We want to find $q_{nk} = p(z_n = k|\boldsymbol{x}; \theta)$. It turns out that

$$p(z|\boldsymbol{x}; \theta) \propto p(z; \theta) p(\boldsymbol{x}|z; \theta), \qquad p(z = k|\boldsymbol{x}; \theta) \propto \pi_k p(\boldsymbol{x}; \theta_k). \tag{145}$$

This is easy because we know this by definition. We will generally know all the values we need to plug in.

Maximization step. We need to maximize $\sum_n E_z \left[\log p(\boldsymbol{x}_n, z_n; \theta)\right]$ which we broke down into components. We can solve for our parameters given q_{nk}. Specifically, we maximize this with respect to π, μ, Σ to find those parameters.

We have

$$\hat{\pi}_k = \frac{1}{N} \sum_n q_{nk}. \tag{146}$$

The q_{nk}'s indicate how likely we are to belong to a particular cluster and N is the number of data points in the data set. This tells you what every single data point believes about the likeliness of a given cluster k which intuitively seems correct.

Moreover, we have

$$\hat{\mu}_k = \frac{\sum_n q_{nk} \boldsymbol{x}_n}{\sum_n q_{nk}}. \tag{147}$$

This is weighted average of the $\boldsymbol{x}$'s that belong in cluster k, weighted by how probable a given $\boldsymbol{x}$ belongs to a cluster. This is divided by the expected cluster size to normalize.

And finally, we have

$$\hat{\Sigma}_k = \frac{1}{\sum q_{nk}} \sum_n q_{nk} (\boldsymbol{x}_n - \mu_k)(\boldsymbol{x}_n - \mu_k)^\intercal. \tag{148}$$

This is how we normally calculate variance, except we weight the probability of given clusters and divide by the expected cluster size to normalize.

Overall, we have analytic updates for both the E step and the M step so we call this the EM algorithm.

15.3.2 General properties

The E step is a posterior over local hiddens, which refers to z variables. The M step maximizes over global parameters.

we have the following properties:

- Monotonically improves the expected complete data likelihood like Lloyd's
- When we have exponential family distributions, the updates are usually analytic
- We will find a local optima, not a maximum optima

Note. If we have a mixture of Gaussians, and imagine we have a small covariance, then we essentially have k-means. This is because the qs will essentially become zs, where they approach hard assignments. Small covariances mean we have a high degree of confidence. Thus given an $\boldsymbol{x}$, we snap that $\boldsymbol{x}$ to the closest distribution with almost 100% certainty.

16 March 24th, 2022

Announcements

- Today's relevant textbook sections are chapter 7
- Homework 5 is out on Friday. We are nearing the end of the semester! The practical and homework 6 are left before midterm 2.

16.1 Embeddings and principal component analysis

Cube. Unsupervised, discrete, probabilistic.

In previous lectures, we covered settings with a discrete hidden variable z. In these situations we wanted to cluster our data into different groups where z told us what group data point $\boldsymbol{x}$ should go in. Today, we switch to a *continuous* z, which we call our *embedding*. These are embeddings of all the information in the data $\boldsymbol{x}$ into a z with lower dimension.

Instead of grouping things into k discrete clusters, we will now try to summarize the data into z continuous dimensions.

Note. Continuous and discrete z are analogs of continuous and discrete h from supervised learning topics we covered. We were given y previously, but now we do not know what categories or embeddings z are needed to infer them ourselves.

Today, we will focus on *principal component analysis (PCA)*, a nonprobabilistic embeddings algorithm.

Example 16.1. Consider two-dimensional data on people's heights and leg lengths and suppose they are highly correlated. The points all lie very close to some line with positive slope (one dimension). It seems like we only need one dimension to approximate each piece of our data.

The question today is how to identify this hidden axis?

16.2 Making this linear

Suppose we have N data points $\boldsymbol{x}_n$ with dimension D. Suppose we try to find a set of K D-dimensional vectors $\{\boldsymbol{u}_k\}$ such that we can approximate some $\boldsymbol{x}_n$ as a linear combination of them:

$$\boldsymbol{x}_n \approx z_{n1}\boldsymbol{u}_1 + z_{n2}\boldsymbol{u}_2 + \cdots + z_{nK}\boldsymbol{u}_K = U\boldsymbol{z}_n. \tag{149}$$

Here $\boldsymbol{z}_n = \begin{bmatrix} z_{n1} & z_{n2} & \cdots \end{bmatrix}$ is a K-dimensional vector that describes the position of $\boldsymbol{x}$ with respect to basis U. We can think of $\boldsymbol{z}_n$ as a vector and U as a matrix with rows corresponding to $\boldsymbol{u}_k$'s.

We are interested in cases where $K < D$ which means we are compressing the data from D to K dimensions. We are *embedding* D-dimensional data into K dimensions.

16.2.1 Minimizing the reconstruction error

We want our approximation of $\boldsymbol{x}_n$ to be accurate. The loss we want to look at is a measure of the difference between $\boldsymbol{x}_n$ and the approximation $U\boldsymbol{z}_n$:

$$\mathcal{L}\left(\{\boldsymbol{z}_n\}_{n=1}^N, U\right) = \frac{1}{N}\sum_n^N \|\boldsymbol{x}_n - U\boldsymbol{z}_n\|_2^2 \tag{150}$$

where $\boldsymbol{x}_n$ is $D \times 1$, U is $D \times K$ and $\boldsymbol{z}_n$ is $K \times 1$, the vector embedding. We are trying to minimize the reconstruction error.

16.2.2 Adding constraints

The solution to the above is not unique. Note that $U = UQQ^{-1}$ where Q is some $K \times K$ invertible matrix. This means we can set $U' = UQ, \boldsymbol{z}' = Q^{-1}\boldsymbol{z}$ and obtain the same loss as before.

We can add constraints to reduce the symmetries in the loss, while adding some nice properties. Depending on the application, we might want U to be sparse or for U to be non-negative.

Today, we will add the constraint that U is *orthonormal*. That is, we have:

- $\langle \boldsymbol{u}_k, \boldsymbol{u}_k \rangle = 1$, which is unit scaling
- $\langle \boldsymbol{u}_k, \boldsymbol{u}_{k'} \rangle = \delta_{kk'}$, which is orthogonality

We get some nice linear algebra properties. We can recover $\boldsymbol{z}$ given $\boldsymbol{x}$. Consider multiplying by $U_k^\intercal$ on both sides

$$\boldsymbol{x}_n U_k^\intercal = z_{n1} U_1 U_k^\intercal + \cdots + z_{nk} U_k U_k^\intercal \implies \boldsymbol{x}_n U_k^\intercal = z_{nk} U_k U_k^\intercal \implies \boldsymbol{x}_n U_k^\intercal = \boldsymbol{z}_{nk} \implies \boxed{\boldsymbol{x}_n U^\intercal = \boldsymbol{z}_n}. \quad (151)$$

16.2.3 Simplifying the problem

Instead of reconstructing $\boldsymbol{x}_n$, we can try to reconstruct the difference in the data from the mean $\bar{\boldsymbol{x}}$. We can write

$$\boldsymbol{x}_n \approx \bar{\boldsymbol{x}} + U\boldsymbol{z}_n \quad (152)$$

where we interpret $U\boldsymbol{z}_n$ as a *perturbation* from the mean. Our new optimization is as follows:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{u}) = \frac{1}{N} \sum_n \|(\boldsymbol{x}_n - \bar{\boldsymbol{x}}) - U\boldsymbol{z}_n\|_2^2 \quad (153)$$

where we now try to reconstruct $(\boldsymbol{x}_n - \bar{\boldsymbol{x}})$. We keep the orthonormality of U as a requirement. From linear algebra, we know that if $K = D$, we incur no loss and have perfect reconstruction.

We can imagine that we have extra $U_{k+1}, \ldots, U_d$ to rewrite the following:

$$\boldsymbol{x}_n = z_{n1} U_1 + \cdots + z_{nk} U_l + \cdots + z_{nD} U_D + \bar{\boldsymbol{x}}. \quad (154)$$

Then we can substitute this into our loss and we have

$$\begin{aligned} \mathcal{L}(\boldsymbol{z}, \boldsymbol{u}) &= \frac{1}{N} \sum_n \left\| \sum_{d=1}^{D} z_{nd} U_d - \sum_{d=1}^{K} z_{nd} U_d \right\|_2^2 \\ &= \frac{1}{N} \sum_n \left\| \sum_{d=K+1}^{D} z_{nd} U_d \right\|_2^2 = \frac{1}{N} \sum_n \left(\sum_{d=K+1}^{D} z_{nd} U_d \right) \left(\sum_{d=K+1}^{D} z_{nd} U_d^\intercal \right). \end{aligned} \quad (155)$$

Due to orthonormality, $U_d^\intercal U_{d'} = \delta_{dd'}$ and we have

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{u}) = \frac{1}{N} \sum_n \sum_{d=K+1}^{D} z_{nd}^2. \quad (156)$$

Substituting in $z_{nd} = U_d^\intercal (\boldsymbol{x}_n - \bar{\boldsymbol{x}})$, we have

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{u}) = \frac{1}{N} \sum_n \sum_{d=K+1}^{D} U_d^\intercal (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^\intercal U_d = \sum_{d=K+1}^{D} U_d^\intercal \underbrace{\frac{1}{N} \sum_n (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^\intercal}_{\boldsymbol{\Sigma}} U_d \quad (157)$$

where the middle term is the empirical covariance of $\boldsymbol{x}, \boldsymbol{\Sigma}$! Thus we have

$$\boxed{\mathcal{L}(\boldsymbol{z}, \boldsymbol{u}) = \sum_{d=K+1}^{D} U_d^\intercal \boldsymbol{\Sigma} U_d} \tag{158}$$

as our objective.

16.3 Solving to minimize reconstruction error

We want to solve for the U that minimizes the above equation with the constraint that $U^\intercal U = 1$.

Note $(K = 1)$**.** If $K = 1$, we note that there is only one U to solve for:

$$\min_U U^\intercal \boldsymbol{\Sigma} U \text{ such that } U^\intercal U = 1. \tag{159}$$

We can put this constraint in via a Lagrange multiplier to create our objective function:

$$\min_U U^\intercal \boldsymbol{\Sigma} U + \lambda(1 - U^\intercal U). \tag{160}$$

Then

$$\boldsymbol{\nabla}_{\boldsymbol{u}} \mathcal{L} = \boldsymbol{\Sigma}\boldsymbol{u} - \lambda \boldsymbol{u} = 0 \implies \boxed{\boldsymbol{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}} \tag{161}$$

and thus, the solution is an **eigenvector of the sample covariance**. We now go back to the main problem given this information.

Recall that the overall goal is to minimize the reconstruction error. Then to minimize, we choose U such that λ is smallest.

It turns out that we can do this is we *leave out* the directions with the smallest eigenvalues. This is like making sure the error is small. Another perspective is that we can minimize the error when we *keep* the eigenvectors with the top K eigenvalues of $\boldsymbol{\Sigma}$. This computation is easy using singular value decomposition (SVD). Because $\boldsymbol{\Sigma}$ is symmetric, it is guaranteed to have D real eigenvalues by the spectral theorem.

Note. The principal components are the top eigenvectors, which get preserved for us in reconstruction. This is what PCA refers to.

16.4 Alternative view: Preserving variance

We can think about this problem as a variance problem. Instead of minimizing the reconstruction error, we can think about it as variance preservation. We want the vectors that capture the most variance in the data $\boldsymbol{x}$. If our bases pointed in directions where the data did **not** vary, then scaling along them would not be useful for better approximating the data.

Suppose we wanted a single vector u that captured most of the variance in $\boldsymbol{x}$

$$z = U^\intercal \boldsymbol{x}. \tag{162}$$

Where z a scalar. We have

$$\text{Var}(z) = U^\intercal \text{Var}(x) U = U^\intercal \boldsymbol{\Sigma} U \tag{163}$$

with the constraint $U^\intercal U = 1$. We want to *maximize* $\text{Var}(z)$:

$$\max_U U^\intercal \boldsymbol{\Sigma} U \text{ such that } U^\intercal U = 1. \tag{164}$$

When we repeat the steps and applying the Lagrange multiplier, we see that $\boldsymbol{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}$ where we pick the largest λ.

Note (Uniqueness and application)**.** The subspace found by PCA is unique, but other good orthonormal solutions exist too! They just aren't found by PCA.

Also, when doing PCA, remember to subtract the mean! Forgetting to subtract the mean will affect our results.

17 March 29th, 2022

Announcements

1. The practical is out!
2. Homework 5 is out
3. Today's relevant textbooks sections are 9.6

Today is the day we finish off the *entire* cube! We have done supervised and unsupervised probabilistic and non-probabilistic discrete structures. Last time we discussed non-probabilistic methods for continuous structures for PCA and today, we will look at probabilistic methods. Next time, we will look at more types of model classes and go deeper into the methods discussed previously.

We will finish off the semester with reinforcement learning. Rather than making predictions about the data, we will focus on making decisions.

Today, we discuss topic modeling.

17.1 Probabilistic embeddings

Topic models were initially designed to organize text collections.

Example 17.1 (Pompeii)**.** There was a study that was done in Pompeii where they looked at rooms in households. They marked down the objects they round and their counts and used a topic model to determine the purpose of the room. They noted objects like doors, chests, cupboards, glass bottles, etc. The reason this is probabilistic is because each room can have more than one purpose. Thus we use a topic model.

17.2 Variations of probabilistic embeddings

We first look at the variations of probabilistic embeddings. Embeddings means unsupervised with continuous z. We will later go into a specific example within probabilistic embeddings.

Recall that the set up is that our zs generate the xs.

17.2.1 Factor analysis

Consider

$$\boldsymbol{z}_n \sim \mathcal{N}(0, \mathbb{1}), \qquad A \sim \mathcal{N}(0, \mathbb{1}), \qquad \boldsymbol{x}_n \sim \mathcal{N} = A\boldsymbol{z} + \varepsilon. \tag{165}$$

In factor analysis, we have some $\boldsymbol{z}$ length k drawn from some normal distribution and $\boldsymbol{x}$ is given by some $A\boldsymbol{z} + \varepsilon$ where A is $D \times k$ and ε is some noise.

This is an example of a probabilistic model where we posit that there is some latent k-dimensional $\boldsymbol{z}$ that leads to the data to become a D-dimensional $\boldsymbol{x}$.

Note. All this stuff is Gaussian. If we recall linear regression, this is basically what we have here. We have $\boldsymbol{x} = A\boldsymbol{z} + \varepsilon$. If A is observed, we are doing linear regression on the $\boldsymbol{z}$'s and vice versa.

17.2.2 Variational autoencoder

We again have

$$\boldsymbol{z} \sim \mathcal{N}(0, \mathbb{1}) \tag{166}$$

with

$$\boldsymbol{x} = f_\theta(\boldsymbol{z}) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \mathbb{1}). \tag{167}$$

Here, our f_θ can be nonlinear with respect to $\boldsymbol{z}$ where f_θ is a neural network characterized by θ, the global parameters.

Note. Now the inference is *harder*. In particular, picking out the local $\boldsymbol{z}_n|\boldsymbol{x}_n, \theta$ is hard. In practice, we learn an *inference network* by making the simplification

$$p(\boldsymbol{z}|\boldsymbol{x}, \theta) \approx q_\phi(\boldsymbol{x}) \tag{168}$$

where q_ϕ is a new neural network with parameters ϕ.

Note. More variations exist! But all approaches are trying to compress the data into something that makes sense. From a use perspective, we want to ask ourselves what we need.

17.3 Topic models

We are now going to discuss topic models. While this is a specific model, many concepts and patterns are generalizable.

For some motivation, suppose we have high-dimensional discrete data. For example, codes in a health record. We believe that these counts of discrete items comes from a lower-dimensional structure (like a disease). We can imagine that a disease can lead to many kinds of health codes.

Example 17.2 (Restaurants)**.** Another perspective in another context is restaurants. A restaurant may have a general food type or fusion of food types $\boldsymbol{z}$ that affects menu items $\boldsymbol{x}$.

Example 17.3 (Documents)**.** We can also think about document analysis. Each document has a mixture of document types that affect the specific words on the document and the counts of words on the document.

17.3.1 The mathematics

To think more concretely, we note that the model is a mixture of multinomials.

Parameters for seeing d items (given mixture k). We have

$$\boldsymbol{\theta}_k \sim \text{Dir}(\beta) \tag{169}$$

is a D-dimensional vector that describes the probability of seeing item d if we are in mixture k.

Example 17.4. If $k = 0$ represents heart disease, θ_0 tells us the distribution of codes given the person has a heart disease and $\text{Dir}(\beta)$ represents the distribution of codes.

Parameters for probabilities of being in mixture k. We have

$$\boldsymbol{\pi}_n \sim \text{Dir}(\alpha) \tag{170}$$

is K-dimensional and tells us the proportions of each $\boldsymbol{\theta}_k$ in data $\boldsymbol{x}_n$.

17.3.2 Procedure to create $\boldsymbol{x}_n$

$\boldsymbol{x}_n$ is a count vector of items per data point which we observe.

Example 17.5 (Pompeii continued)**.** In the Pompeii example, this can be like 3 chairs, 2 chests, 0 spoons, etc.

We will look into how the $\boldsymbol{x}_n$'s are created.

We first create the $\boldsymbol{\theta}$'s. For each data point $\boldsymbol{x}_n$, we are going to sample $\boldsymbol{\pi}_k \sim \text{Dir}(\alpha)$ and then we use our $\boldsymbol{\pi}_n$s to create data $\boldsymbol{x}_n$. Namely, we have

$$\boldsymbol{x}_n \sim \text{Mult}(\boldsymbol{\theta}\boldsymbol{\pi}_n, L) \tag{171}$$

where L is the total number of codes, menu items, or words.

Note. When we multiply $\boldsymbol{\theta}$ $(D \times K)$ and $\boldsymbol{\pi}$ $(K \times 1)$, we get the probabilities for items in $\boldsymbol{x}$ which is $D \times 1$. We will then have the final probabilities we need for a given patient, and we enter this in as the parameters for the multinomial, with our $\boldsymbol{x}_n$ generated by the distribution. We can consider each probability in this $D \times 1$ output matrix multiplied by the length L to estimate the count of words in the corpus.

Note (Connection to unsupervised setting)**.** In unsupervised learning, our $\boldsymbol{z}_n$ produces $\boldsymbol{x}_n$ and $\boldsymbol{w}$ (global variables) also affect $\boldsymbol{x}_n$. Here, $\boldsymbol{z}_n$ corresponds to $\boldsymbol{\pi}_n$ and $\boldsymbol{w}$ corresponds to $\boldsymbol{\theta}$.

17.4 Dirichlet distributions

We need a process to create $\boldsymbol{\pi}$. In earlier lectures, we had $\boldsymbol{z}$ distributed as a Gaussian. Now we need the following:

$$0 \leq \boldsymbol{\pi}_{nk} \leq 1, \qquad \sum_k \boldsymbol{\pi}_{nk} = 1. \tag{172}$$

We need to follow these constraints so we have a probability distribution we can interpret.

Note. The sum of $\boldsymbol{\pi}_{nk}$ over all categories is 1 because the document must lie in some proportion of each of these categories.

We do not want any of these values to be negative because we are trying to define the strength of existence of attributes. We cannot have negative counts/attributes which is possible with a Gaussian distribution.

We can define the Dirichlet distribution as

$$\text{Dir}(\boldsymbol{\pi}|\alpha) \propto \prod_{k=1}^{K} \boldsymbol{\pi}_k^{\alpha_k - 1}. \tag{173}$$

Note that this is similar to a Beta distribution given by

$$\text{Beta}(p|\alpha, \beta) = p^{\alpha-1}(1-p)^{\beta-1}. \tag{174}$$

17.5 Relationship with discrete mixtures

We can think about how topic models tie into mixture models and factor analysis.

17.5.1 Mixture models

Recall that for mixture models, we have

$$\boldsymbol{z}_n \sim \text{Cat}(\boldsymbol{\pi}), \qquad \boldsymbol{x}_n = \mu_{\boldsymbol{z}_n} + \varepsilon. \tag{175}$$

where $\boldsymbol{\pi}$ is a distribution over k mixture classes.

17.5.2 Topic models

Note that

$$\boldsymbol{\pi}_n \sim \text{Dir}(\alpha), \qquad \boldsymbol{x}_n \sim \text{Mult}(\boldsymbol{\theta}\boldsymbol{\pi}_n, L). \tag{176}$$

17.5.3 Factor analysis

In factor analysis, recall that

$$\boldsymbol{z}_n \sim \mathcal{N}(0, \mathbb{1}), \qquad \boldsymbol{x}_n = A\boldsymbol{z}_n + \varepsilon. \tag{177}$$

17.5.4 Comparisons

Note that the idea is in the same way that in mixture models and factor analysis, $\boldsymbol{z}_n$ determines our $\boldsymbol{x}_n$ after incorporating global parameters. In topic models, once we have $\boldsymbol{\pi}_n$, we can create $\boldsymbol{x}_n$ after incorporating our global parameter $\boldsymbol{\theta}$.

The differences is that in topic models and factor analysis, our $\boldsymbol{\pi}$ and $\boldsymbol{z}$ are continuous whereas they are discrete.

17.6 Inference

We will now continue with topic models and move to inference.

17.6.1 Generative model

We start with the generative model:

$$p(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{\theta}) = \prod_d \left(\sum_k \boldsymbol{\pi}_k \boldsymbol{\theta}_{kd} \right)^{\boldsymbol{x}_d} \tag{178}$$

where $\sum_k \boldsymbol{\pi}_k \boldsymbol{\theta}_{kd}$ represents the probability of dimension d, $\boldsymbol{x}_d$ represents the number of times element d was observed and $(\sum_k \boldsymbol{\pi}_k \boldsymbol{\theta}_{kd})^{\boldsymbol{x}_d}$ represents the probability of seeing $\boldsymbol{x}_d$. We then take a product over d to have the probability of $\boldsymbol{x}$ overall.

We can then take the log and obtain

$$\log p(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{\theta}) = \sum_d \boldsymbol{x}_d \log \sum_k \boldsymbol{\pi}_k \boldsymbol{\theta}_{kd}. \tag{179}$$

This is tricky because there is a sum in the logarithm!

17.6.2 A new equivalent generative model

We will now assume that $\boldsymbol{\theta}$ is a global parameter where

$$\boldsymbol{\theta} \sim \text{Dir}(\beta). \tag{180}$$

Locally, for each n, we model the following:

$$\boldsymbol{\pi}_n \sim \text{Dir}(\alpha) \tag{181}$$

and for each $l \in L$ we have

$$\boldsymbol{z}_{nl} \sim \text{Cat}(\boldsymbol{\pi}_n), \qquad \boldsymbol{w}_{nl} \sim \text{Cat}(\boldsymbol{\theta}_{\boldsymbol{z}_{nl}}) \tag{182}$$

where $\boldsymbol{x}_n$ then keeps counts of $\boldsymbol{w}_{nl}$.

Under this new model, the new proposal is to generate data for a new patient. We first determine the mixture weights of the different diseases. For simplicity, suppose some oracle tells us the patient has the L_n. Once we know this, for each given code, we can ask what disease we are (the $\boldsymbol{z}_{nl} \sim \text{Cat}(\boldsymbol{\pi}_n)$). Then, once we pick the disease, we can pick the code given we know that the disease is ($\boldsymbol{w}_{nl} \sim \text{Cat}(\boldsymbol{\theta}_{\boldsymbol{z}_{nl}})$.

This is known as the *bag of words representation.*

17.6.3 Motivation

We did this because in the original setup, it was impossible to solve easily because EM algorithm would not work. Given $\boldsymbol{\theta}$, it would be hard to find $\boldsymbol{\pi}$ and given $\boldsymbol{\pi}$, it is still hard to find $\boldsymbol{\theta}$. We can introduce a new variable $\boldsymbol{z}$ that lets us to EM. The new likelihood is

$$p(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\pi},\boldsymbol{\theta}) = \prod_l \prod_d \boldsymbol{\theta}_{dk}^{\mathbb{1}_{\boldsymbol{w}_{nl}=d}\mathbb{1}_{\boldsymbol{z}_{nl}=k}} \tag{183}$$

where $\prod_l$ is a product over all words in the document, $\prod_d$ is the product over all possible word assignments, $\prod_k$ represents the product over all topics/mixtures and $\boldsymbol{\theta}_{dk}$ is the probability of a particular dimension occurring with indicators that check for the right topic. Thus

$$\log p(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\pi},\boldsymbol{\theta}) = \sum_l \sum_d \sum_K \mathbb{1}(\boldsymbol{w}_{nl}=d)\mathbb{1}(\boldsymbol{z}_{nl}=k). \tag{184}$$

Now we want to get the rest of what we need to do the EM algorithm:

$$p(\boldsymbol{z}|\boldsymbol{\pi}) = \prod_k \boldsymbol{\pi}_k^{\mathbb{1}(\boldsymbol{z}=k)}, \qquad p(\boldsymbol{\pi}) = \prod_k \boldsymbol{\pi}^{\alpha_k - 1}. \tag{185}$$

Then

$$\log p(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\pi}|\boldsymbol{\theta}) \propto \sum_n \left\{ \left[\sum_k \left(\alpha_k - 1 + \sum_l \mathbb{1}(\boldsymbol{z}_{nl}=k) \right) \log \boldsymbol{\pi}_k \right] + \sum_d \sum_k \left[\sum_l \mathbb{1}(\boldsymbol{w}_{ln}=d)\mathbb{1}(\boldsymbol{z}_{ln}=k) \right] \log \boldsymbol{\theta}_{dk} \right\}. \tag{186}$$

Note. Note that $\mathbb{1}(\boldsymbol{w}_{ln}=d)$ is known and fixed. This means that for inference, we can run EM where we take expectations across with respect to $\boldsymbol{z}$, which turns the indicators into $\boldsymbol{q}_{ln}$'s and then maximize with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$.

By rewriting the inference in this way, it lets us apply the EM algorithm.

17.6.4 Alternative derivation from lecture

We can consider the joint probability where $\boldsymbol{\pi}_n$ is a variable and $\boldsymbol{\theta}$ is a parameter. We need

$$p(\boldsymbol{x}_n,\boldsymbol{\pi}_n|\boldsymbol{\theta}) = \prod_n p(\boldsymbol{x}_n,\boldsymbol{\pi}_n|\boldsymbol{\theta}) = \prod_n p(\boldsymbol{x}_n|\boldsymbol{\pi}_n,\boldsymbol{\theta}) \underbrace{p(\boldsymbol{\pi}_n|\boldsymbol{\theta})}_{p(\boldsymbol{\pi}_n)} \tag{187}$$

Note. We are using $\boldsymbol{\pi}_n$ because in the topic model literature, we introduce a new variable $\boldsymbol{z}_n$ that is used in a *slightly different way.* In 181, however $\boldsymbol{\pi}_n$ is our $\boldsymbol{z}_n$.

Then

$$p(\boldsymbol{x}_n,\boldsymbol{\pi}_n|\boldsymbol{\theta}) = \prod_n \left(\prod_d \underbrace{\left[\sum_k \boldsymbol{\pi}_{nk}\boldsymbol{\theta}_{kd} \right]^{\boldsymbol{x}_{dn}}}_{\text{prob. of word } d \text{ in document}} \right) \underbrace{\left(\prod_k \boldsymbol{\pi}_n^{\alpha_k - 1} \right)}_{\text{prior}}. \tag{188}$$

This makes sense, but it is messy to solve because of the log in the sum. We can take a logarithm of this expression, but this won't separate.

Thus to make inference easier, we will add a new variable. We can consider a new generative model where

$$\boldsymbol{\pi}_n \sim \text{Dir}(\alpha) \tag{189}$$

and for each l in L, let

$$\boldsymbol{z}_{nl} \sim \text{Cat}(\boldsymbol{\pi}_n), \qquad \boldsymbol{w}_{nl} \sim \text{Cat}(\boldsymbol{\theta}_{\boldsymbol{z}_{nl}}) \tag{190}$$

where we first pick the topic that a word came from $\boldsymbol{z}_{nl}$ and then we pick out the actual word $\boldsymbol{w}_{nl}$.

Note. $\boldsymbol{x}_n$ corresponds to counts of $\boldsymbol{w}_{nl}$.

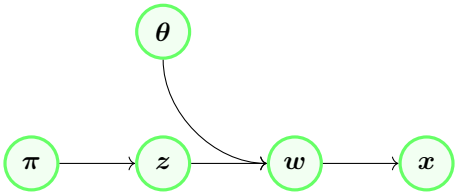
Moreover, this is equivalent because if we integrate out the $\boldsymbol{z}_{nl}$, we recover the initial formulation.

Note (Machine learning trick)**.** This trick is out of scope of the class. Adding an augmenting variable while leaving the original expression unchanged is a general machine learning trick.

We can try again with this new formulation:

$$p(\boldsymbol{x}_n, \boldsymbol{\pi}_n, \{\{\boldsymbol{z}_{nl}\}_{l=1}^{L}\}_{n=1}^{N} | \boldsymbol{\theta}) = \prod_n p(\boldsymbol{\pi}_n) \prod_l p(\boldsymbol{z}_{nl}|\boldsymbol{\pi}_n) p(\boldsymbol{w}_{nl}|\boldsymbol{z}_{nl}, \boldsymbol{\theta}) \tag{191}$$

Note. Once we know which topic we belong to, we don't need to know the $\boldsymbol{\pi}$. In the graphical model, we have



Then we can do some algebra and recover the complete data log-likelihood given by

$$\log p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\pi}|\boldsymbol{\theta}) \propto \sum_n \left\{ \left[\sum_k \left(\alpha_k - 1 + \sum_l \mathbb{1}(\boldsymbol{z}_{nl} = k) \right) \log \boldsymbol{\pi}_k \right] + \sum_d \sum_k \left[\sum_l \mathbb{1}(\boldsymbol{w}_{ln} = d) \mathbb{1}(\boldsymbol{z}_{ln} = k) \right] \log \boldsymbol{\theta}_{dk} \right\}. \tag{192}$$

Note (Observations)**.** Given $\boldsymbol{z}$, we can easily solve for $\boldsymbol{\theta}, \boldsymbol{\pi}$. Moreover, given $\boldsymbol{\pi}, \boldsymbol{\theta}$, solving for the best $\boldsymbol{z}$ is also easy. This is because $p(\boldsymbol{z}_{nl}|\boldsymbol{w}_{nl}, \boldsymbol{\pi}_n, \boldsymbol{\theta}_{\boldsymbol{w}_{nlk}} \propto \boldsymbol{\pi}_n^{\boldsymbol{z}_{nl}} \boldsymbol{\theta}_{\boldsymbol{w}_{nlk}}$. Finally, note that we can apply EM to find best parameters.

18 March 31st, 2022

Announcements

- Today's relevant readings are chapter 8
- There are two options for the practical due date
- Do some practice problems to get familiar with the material!

Last time, we finished the cube! Today, we are going to move beyond the cube. We can think of graphical models as a way of describing more complex relationships between variables than those we have seen so far in supervised and unsupervised learning.

18.1 Graphical models

Graphical models provide a structured representation of the probabilistic relationships in a problem. Graphical models help us visualize and communicate about models.

Example 18.1 (Lung cancer)**.** We can consider some observation $\boldsymbol{x}$ where one component is smoking, another is lung cancer, another is age, etc. There can be some underlying statistical structure or dependence between the components. Graphical models help us model these relationships.

We have already been informally drawing graphical models for a long time. In sueprvised learning, we considered cases where a hidden variable y generated $\boldsymbol{x}$'s (generative), or a hidden label $\boldsymbol{x}$ generated by y's (discriminative). We diagrammed the generative scenario with an arrow pointing from $y \to \boldsymbol{x}$ and the discriminative scenario with an arry from $\boldsymbol{x} \to y$.

Example 18.2 (Mixture models)**.** More recently, we have seen scenarios with more complicated diagrams with hidden z's that generated $\boldsymbol{x}$'s. Wew draw this in a plate to denote that we have a whole set of N z's generated this way. The z's and $\boldsymbol{x}$'s were affected by our parameters $\boldsymbol{\pi}, \mu, \boldsymbol{\Sigma}$. This tells us we can factor the joint distribution as

$$p(\boldsymbol{\pi}, \{\theta_k\}_{k=1}^K, \{(z_n, \boldsymbol{x}_n\}_{n=1}^N) = p(\boldsymbol{\pi}) \prod_k p(\theta_k) \prod_n p(z_n|\boldsymbol{\pi}) p(\boldsymbol{x}_n|z_n, \{\theta_k\}) \tag{193}$$

Example 18.3 (Topic models)**.** We also saw topic models, which led to more complicated diagrams.

These diagrams help encode the structure of the data, including relationships between variables.

Note. All the probability models from 181 are examples of *directed acyclic graphs* (DAGs). These are *Bayesian networks* that have no cycles.

18.1.1 Notation and rules

The notation we use in graphical models is as follows:

1. Random variables are represented by an open circle. If we observe a random variable of a given model, then we shade it in. Otherwise, it is open.
2. Deterministic parameters are represented by a tight, small dot.
3. Arrows indicate the probabilistic dependence relationship between different random variables and parameters. An arrow $X \to Y$ means Y depends on X.
4. Plates, or boxes, indicate repeated sets of variables. Often there will be a number in one of the corners indicating how many times the variable is repeated.

   Example 18.4 (Plates)**.** Consider data $\mathcal{D} = \{(x_i, y_i\}_{i=1}^N$ drawn from X, Y, we can draw a plate around X, Y with N in the corner.

18.2 Bayesian networks

A Bayesian network is a special kind of graphical model. Bayesian networks (Bayes nets) are *directed acyclic graphs* (DAGs) where nodes represent random variables and directed arrows represent dependency relationships between random variables. In a DAG, nodes have directed arrows between them and there are no cycles. We can thus simplify (and factor) joint probabilities nicely.

Bayes nets are useful for the following:

1. For inference: knowing which variables are independent helps us determine when we can use block coordinate ascent to infer their values
2. For learning: Bayes nets allow us to learn *smaller* distributions, or distributions with fewer parameters

18.2.1 d-separation

We first define the principle of local independence: every node is conditionally independent of its non-descendants given its parents.

Note. We will use $\perp$ to denote conditional independence.

When we discuss d-separation, we use the term *information flow* to describe dependencies between variables. If there is information flowing from $A \to B \implies A, B$ are conditionally dependent given the observed random variables. In contrast, A, B are *blocked* means that A, B are conditionally independent given the observed random variables.

Thus two variables A, B are d-separated, or independent, if every undirected path from $A \to B$is *blocked*. There are several ways a path can be *blocked*:

$$A \to C \to B, \qquad A \leftarrow C \leftarrow B, \qquad A \leftarrow C \to B, \qquad A \to C \leftarrow B \tag{194}$$

In the first three cases, the path is blocked if C is observed. The last case, the path is blocked if C is *not* observed.

18.3 Uniqueness and parameters

18.3.1 Uniqueness

Under a causal interpretation, A causes B and we cannot say B causes A. Under a statistical interpretation, we think about how knowing A informs us about the distribution of B. Similarly, knowing B informs us about the distribution of A.

We can write a joint distribution two ways:

$$A \longrightarrow B : p(A, B) = p(A)p(B|A), \qquad B \longrightarrow A : p(A, B) = p(B)p(A|B). \tag{195}$$

Note. Though the statistical interpretation allows multiple orderings, one ordering may require the fewest parameters and contain the most independences.

18.3.2 Parameter counting

We can think about how many parameters we need to express our nets. Let e_i denote the number of incoming edges to a node. Assuming binary variables, each node requires 2^{e_i} parameters.

Suppose we wanted a different ordering. This different ordering may admit an expression that requires *fewer* parameters.

Note (Re-ording the network)**.** We can change the topology of the net while preserving the same dependencies.

18.4 Beyond Bayes networks

There are more graphical models! Below are two other variations.

18.4.1 Undirected models

Undirected models are graphs where edges between random variables are undirected.

The joint distribution can be factored as

$$p(A, B, C, D, E) = \frac{1}{z}\phi(C, A, B)\phi(B, D)\phi(B, E) \tag{196}$$

where z is some normalization constant and ϕ a function.

These are popular in settings like image segmentation. In practice, they are hard because finding the normalization constant z that makes the joint probability a valid distribution is hard.

18.4.2 Factor graphs

In factor graphs, variables connect to their factors, which are denoted by shaded nodes. Each shaded node corresponds to a ϕ. The joint distribution of a factor graph can be written as

$$p(A, B, C, D) = \phi(A, C)\phi(A, B)\phi(B, C)\phi(C, D). \tag{197}$$

Each of the $\phi(\cdot)$ terms is created by examining each of the factor nodes and including the variables that are connected by each factor.

Factor graphs are more expressive than undirected graphs. We see that an additional factor term $\phi(B, C)$ is in this expression. This cannot have been specified in our undirected graphical model.

19 April 5th, 2022

Announcements

- Professor Finale's son tested positive for COVID. Thursday lecture will be remote.

Last time, we started adding structure to $p(\boldsymbol{x})$. The goal of probabilistic unsupervised learning is to model $p(\boldsymbol{x})$. Similarly, the goal of probabilistic *supervised* learning methods is to model $p(y|\boldsymbol{x})$.

Today, we will discuss inference.

19.1 Bayesian networks review

Last time, we defined Bayesian networks and we saw how they are useful in the probabilistic setting regardless of whether we are in the unsupervised or supervised world because it helps us describe the structure in the data.

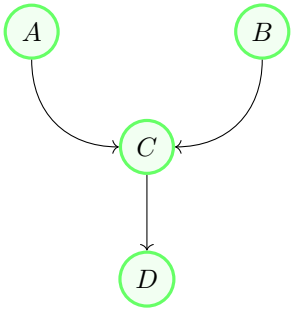
Note (Concept check)**.** In last lecture's concept check, we found that continuous cases have less parameters. This is because in the continuous case, we restricted to a very specific class (linear). In the discrete case, we consider all distributions.

Example 19.1. We can check the hypothesis that smoking causes lung cancer with Bayes nets. Note that this was actually resolved by testing smoking with animals.

19.2 Inference

19.2.1 Setup

Given the following graph:



We can ask about different conditionals and marginals, for example $p(D), p(D|A)$, etc. We can infer different things about the data. The usual procedure is to write out the joint distribution and marginalize out what we do not need.

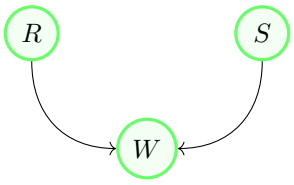
For example, we can do the following:

$$p(D) = \sum_{A,B,C} p(A,B,C,D) = \sum_{A=1}^{K}\sum_{B=1}^{K}\sum_{C=1}^{K} p(A)p(B)p(C|B,A)p(D|C) \tag{198}$$

Here, we suppose each feature has K values. This is a general property and note that this requires K^4 computational steps.

19.2.2 Specific example: Rain and sprinkler

We can try an example involving rain R, sprinkler S and wet grass W.



where $p(S) = 1/2, p(R) = 1/4$. We can consider the actual distribution of $p(W|S, R)$.

Suppose we are looking for

$$p(R|W) = \frac{p(R, W)}{p(W)}. \tag{199}$$

We can then essentially to LOTP and expand. We get that

$$p(R|W) = \frac{p(R, W, S = 0) + p(R, W, S = 1)}{p(R, W, S = 0) + p(R, W, S = 1) + p(R^c, W, S = 0) + p(R^c, W, S = 1)} = \frac{21}{51} \tag{200}$$

where $p(W, R, S) = p(W|R, S)p(R)p(S)$ because this is how we factor the joint given the Bayesian network. This is greater than our prior because the probability of raining is greater given the grass is wet.

We can do the same for the conditional probability $p(R|W, S)$ and we see that

$$p(R|W, S) = \frac{p(R, W, S)}{p(W, S)} = \frac{p(W|R, S)p(R)p(S)}{p(W|R, S)p(R)p(S) + p(W|R^c, S)p(R^c)p(S)} = \frac{11}{41} \tag{201}$$

which is close to the prior 0.25. This makes sense because we saw that sprinklers were on, which reduces the probability that it has rained because the sprinkler helped explain the wet grass.

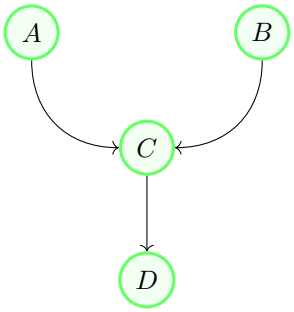
We can do this once more to find

$$p(R|W, S^c) = \frac{p(W|R, S^c)p(R)p(S^c)}{p(W|R, S^c)p(R)p(S^c) + p(W|R, S^c)p(R^c)p(S^c)} = 1. \tag{202}$$

Which is what we expected because we set $p(W^c|R^c, S^c) = 0$ which means that if it it did not rain and the sprinkler was off, the grass cannot be wet.

19.2.3 Choosing the order of elimination

We can now go back to our original diagram



Suppose we want to find $p(D)$. We noted that

$$p(D) = \sum_{A,B,C} p(A,B,C,D) = \sum_A \sum_B \sum_C p(A)p(B)p(C|A,B)p(D|C). \tag{203}$$

The question is what order we should do the summations. We can choose an ordering

$$\sum_C p(D|C) \sum_B p(B) \sum_A p(A)p(C|A,B) \tag{204}$$

which is a top-down approach. We marginalize each node.

Note (Geometric intuition)**.** The summation over A is hitting a K-dimensional cube with a K-dimensional vector. We will end with something that is a $K \times K$ sized object. We then apply a vector size K to marginalize out B and finally in the last sum, this gives a quantity size K, which is the solution.

Thus this computation is now only K^3, which is better than our original K^4 computations.

19.3 Minimum cost of inference

How do we determine the minimum cost of inference? This depends on something called *treewidth*.

Note. In general, choosing an optimal ordering of marginalization is NP-hard. However, for a *polytree*, we have a strategy.

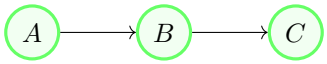
Definition 19.2 (Polytree)**.** A *polytree* is a directed acyclic graph whose underlying undirected graph is a tree, that is, it is both connected and acyclic.

19.3.1 Optimal order for polytrees

There are two steps.

Step 1. First, we prune variables that are descendants of the queried or evidenced variables.

Example 19.3 (Pruning)**.** For example in $p(B|A)$, B is the query and A is the evidence. Thus, if we have the following:



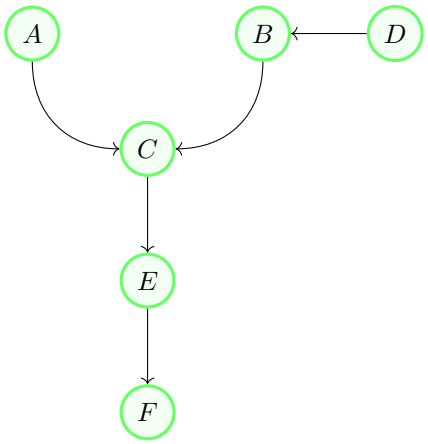
we can eliminate C. The intuition is that in the math, we do the following:

$$p(B|A=1) = \sum_C p(B,A,C) = \sum p(B|A)p(C|B) = p(B|A) \sum_C p(C|B) \tag{205}$$

where the key is that $\sum_C p(C|B) = 1$ because we do not care about C and when we take the sum, it becomes 1. Thus we can prune the descendants.

Step 2. Next, we find the leaves and work backwards. What is important is finding the leaves (ignoring the directions of the edges) and working backwards to the query.

Example 19.4 (Finding leaves)**.** Consider the following diagram:



Suppose we are looking at $p(E|D)$. First, we removed F immediately from step 1, since it is a descendant of E.

Note. We cannot remove B because even though it is a descendant of D, it is an ancestor of E. We need to focus on the variables we need to sum out, namely A, B, C.

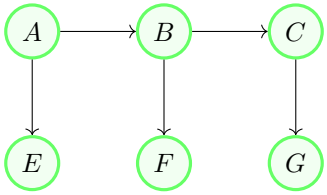
Because we work from the leaves inward, we note that we need to remove C last. We can eliminate in two orders: A, B, C or B, A, C.

These two steps will result in the most optimal order for polytrees.

19.4 Preview: Hidden Markov models

Next lecture, we will look at the *hidden Markov model*.

One example of this is the following:



We can imagine the top represents some time series of a true location we want to track and E, F, G correspond to radar measurements. When the plane is at position $\{A, B, C\}$, the radar said it was at $\{E, F, G\}$. Now, we want to know the probability of the current location of the plane given the last three sets of radar measurements.

Suppose we have discrete values and observed the radar to tell us that $E = F = G = 1$ and suppose we are looking for $p(C|E, F, G)$. Then

$$p(C|E, F, G) \propto \sum_A \sum_B p(A)p(E|A)p(B|A)p(F|B)p(C|B)p(G|C). \tag{206}$$

We can determine the order of elimination because these are also polytrees! There are no descendants to prune away. In this case, we work from the leaves towards the query, so we want to eliminate A and then B. Thus we have the ordering

$$p(G|C) \sum_B p(F|B)p(C|B) \sum_A p(A)p(E|A)p(B|A). \tag{207}$$

When we sum out A, $p(B|A)$ is $K \times K$ and after summing we get a single vector dimension K, factor $g(B)$. When we sum out B, we get another K-sized factor $g(C)$.

These factors are often called *messages*.

If we sum a different way, we get different factors that differ in both computation and memory.

19.5 Final notes

There are many algorithms for exact inference, but they are all the same ideas we covered. There is some terminology.

- Sum-product: This is equivalent to exact inference. It is also called this because our equations are sums of products.
- Forward-backwards: This is equivalent to the specific sum product idea for hidden Markov models.
- Junction-tree: This occurs more in the theory literature, and refers to specific type of tree structures like polytrees.

20 April 7th, 2022

Announcements

- Professor Doshi-Velez's son is doing ok!
- Today's relevant sections are chapter 10

Today, we discuss hidden Markov models. It is our *last day* discussing *unsupervised* learning. More specifically, we discuss a time series model.

20.0.1 Review

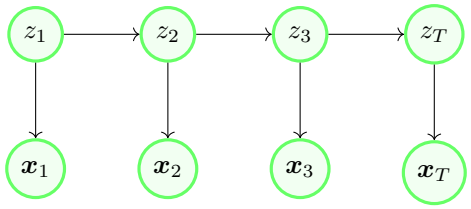
Recall that our goal in unsupervised learning is to model the data $p(\boldsymbol{x})$ because by capturing $p(\boldsymbol{x})$ in a probabilistic approach, we are capturing the *structure* of the data. In the last two lectures, we want to consider if there is indeed structure in $p(\boldsymbol{x})$ if $\boldsymbol{x}$ has multiple dimensions that have interesting conditional independence relations.

We noted that relationships can be used to perform inference more efficiently.

20.1 Time series and hidden Markov models

Hidden Markov models are a specific Bayesian network form for studying time series. The idea is that we are observing something that changes over time. We have an evolving latent state z where z_t is the value of the state at time t. The time goes from $t \in [0, T]$ where T is the last time-step we measure at.

For today, z is a discrete variable that can take one of K possible values. We do not know the true values of z but each z_t produces some measurement $\boldsymbol{x}_t$ that we can observe. In particular, the Bayesian net is given by



Note. The z's are our *local* variables that describe some hidden unobserved structure. The $\boldsymbol{x}_i$ are the observed vectors.

20.1.1 Global parameters

There are both global and local unknowns that we want to infer.

The three main *global* parameters are

- $p_0(z)$: The prior. This tells us what state we start in and gives us distribution of z_1, where we start.
- $p_T(z_{t+1}|z_t)$: The transition probabilities. The distribution of the *next* z given the current z.
- $p_\Omega(\boldsymbol{x}|z)$: The *global* emission probabilities which is the distribution of $\boldsymbol{x}$ given the current z.

The *local* unknowns are the latent states $z_1, z_2, \ldots, z_T$.

20.2 Questions we want to answer

20.2.1 Collection 1: Given globals, solve for locals

Given p_0, T, Ω, we want to find things about $z, \boldsymbol{x}$. Some specific values are

1. **Filtering.** We want to find

$$p(z_t|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t). \tag{208}$$

   This is real-time prediction. We can only use the data observed so far to make a prediction about time t.

   Example 20.1 (Missile detection)**.** We can say $\boldsymbol{x}$'s are radar measurements and z is the potential location of a missile. We want to use the data so far to estimate the current location of the missile.

2. **Smoothing.** We want to find

$$p(z_t|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T). \tag{209}$$

   This is *afterward inference*. We use data from before and after time t to make a prediction about t.

   Example 20.2 (Bird songs)**.** We can use all information from collected audio to determine if a particular bird was singing at time t.

3. **Probability of sequence.** We want

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T). \tag{210}$$

   This is the probability that a particular sequence occurred. This is useful for model selection.

   Example 20.3 (Audio continued)**.** Suppose we want to detect outliers in the audio and determine whether a section of the recording was background noise. We can use the probability of the sequence of tell if the audio was an outlier.

4. **Predict measurements.** We want to predict the next observation $\boldsymbol{x}_t$:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}). \tag{211}$$

   Example 20.4 (Hospital usage)**.** We may want to predict the hospital utilization rate tomorrow given what has happened so far.

5. **Best path.** We want to find the best latents:

$$\operatorname*{argmax}_{\boldsymbol{z}} p(z_1, \ldots, z_T|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T). \tag{212}$$

   Example 20.5 (Set of spoken words)**.** Suppose we have audio of someone speaking and we want to find the most probable set of words they have spoken that would produce the data in the recording.

Note. We have seen the first four problems! They all have the form $p(\cdot|\cdot)$ that requires various conditionals and marginals of the joint distribution $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T, z_1, \ldots, z_T)$.

20.3 Collection 2: Given $\boldsymbol{x}$'s, solve for globals

We also want to learn the globals given $\boldsymbol{x}$. Specifically, given $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$, we want to find the MLE or MAP of p_0, T, Ω.

This is important because we need to learn some value before solving problems in collection 1. The full joint probability in an HMM is

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T, z_1, \ldots, z_T) = p_0(z_1) \prod_{t=1}^{T} \underbrace{p_T(z_{t+1}|z_t)}_{A} \prod_{t=1}^{T} \underbrace{p_\Omega(\boldsymbol{x}_t|z_t)}_{B} \tag{213}$$

where A is all the terms from transition probabilities and B is all the terms from the observation probabilities.

Note. We can evaluate $p(z_1, \ldots, z_T|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ up to a constant factor by looking at the joint but if we wanted to look at something like $p(z_t|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$, we need to marginalize out all of the z's at other times.

The forward-backward algorithm (FBA) does this efficiently. It is a special case of the inference methods we covered last week.

20.4 Forward-backward algorithm: Solving questions in collection 1

Recall that collection 1 problems involve problems when p_0, T, Ω are known.

20.4.1 Intuition

The intuition we will use is that in the network, we have a time-series structure. If we wanted to know something about z_t. There are three sources of information information: from the past, from the future, and there is the present. In particular, for efficient inference, we consider information from the past and present as one factor and information from the future as another factor.

Formally, we will have α terms that are *messages* that feed information forward and β terms that are *messages* that feed information backwards.

20.4.2 The forward pass and α_t.

Let us consider

$$p(z_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t). \tag{214}$$

Note. This joint is interesting because

$$p(z_t = k|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) = \frac{p(z_t = k, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)}{\sum_j p(z_t = j, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t)} \propto p(z_t = k, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) \tag{215}$$

and if z_t is discrete, this proportionality is relatively easy to solve.

We can define the following:

$$\boxed{\alpha_t(z) \equiv p(z_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t).} \tag{216}$$

α_t is a K-dimensional vector where the ith entry is the case where $\alpha_t^{(i)} = \alpha_t(z_t = i)$, that is, this corresponds to the case $z_t = i$.

We can explicitly marginalize z_{t-1} and using our knowledge of the structure, we have

$$\begin{aligned}\alpha_t(z) = \sum_{z_{t-1}=1}^{K} p(z_{t-1}, z_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) &= \sum_{z_{t-1}=1}^{K} p(\boldsymbol{x}_t|z_t) p(z_t|z_{t-1}) p(z_{t-1}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) \\ &= \underbrace{p(\boldsymbol{x}_t|z_t)}_{\text{size } K} \sum_{z_{t-1}=1}^{K} \underbrace{p(z_t|z_{t-1})}_{K \times K \text{ matrix}} \underbrace{\alpha_{t-1}(z_{t-1})}_{\text{size } K}.\end{aligned} \tag{217}$$

Note. This sets us up for some dynamic programming solution. This is how we can efficiently calculate the α coefficients.

We can define the base case and the general case:

$$\boxed{\alpha_t(z_t) = \begin{cases} p_0(z_1)p_\Omega(\boldsymbol{x}_1|z_1), & t = 1 \\ \left[\sum_{z'} \alpha_{t-1}(z')p_T(z_t|z')\right] p_\Omega(\boldsymbol{x}_t|z_t), & t \neq 1 \end{cases}} \tag{218}$$

where z' indicates all possible values for z_{t-1}. In the recursive expression, the left term is getting the previous $\alpha_{t-1}(z')$ and accounting for the transition probability of going into the current state. The other term incorporates information about our current observation.

20.4.3 The backward pass and β_t.

β_t is similar to α but we use information from the future instead of the past. It is a K-dimensional vector where the ith entry corresponds to z_t in class i.

Note (Motivation)**.** For motivation, we consider the smoothing problem, where we need

$$p(z_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) = p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t, z_t)p(\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T|z_t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) = \underbrace{p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t, z_t)}_{\text{forward pass } \alpha} \underbrace{p(\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T|z_t)}_{\text{backward pass } \beta} \tag{219}$$

We know how to get the α_t factors. We will now discuss how to efficiently compute the second term, β_t.

We will define

$$\boxed{\beta_t(z_t = k) \equiv p(\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T|z_t = k).} \tag{220}$$

We can employ the same trick (explicitly marginalizing and using the underlying structure) as above and obtain

$$\begin{aligned} \beta_t(z_t) = \sum_{z_{t+1}=1}^{K} p(\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T, z_{t+1}|z_t) &= \sum_{z_{t+1}=1}^{K} \underbrace{p(\boldsymbol{x}_{t+1}|z_{t+1})}_{\text{observation}} \underbrace{p(\boldsymbol{x}_{t+2}, \ldots, \boldsymbol{x}_T|z_{t+1})}_{\text{future events}} \underbrace{p(z_{t+1}|z_t)}_{\text{transition probability}} \\ &= \sum_{z_{t+1}=1}^{K} p(\boldsymbol{x}_{t+1}|z_{t+1})p(z_{t+1}|z_t)\beta_{t+1}(z_{t+1}) \end{aligned} \tag{221}$$

so instead of working from the front, we work from the last event and move backwards. The convention we use is that $\beta_T(z_T) = 1$ because there is no information for $t = T$.

The recursive expressions for β are

$$\boxed{\beta_t(z_t) = \begin{cases} 1, & t = T \\ \sum_{z'} \beta_{t+1}(z')p_T(z'|z_t)p_\Omega(\boldsymbol{x}_{t+1}|z'), & t \neq T \end{cases}} \tag{222}$$

as desired.

20.4.4 Solving questions in collection 1

We now return to solving our tasks.

1. **Filtering.** Now we have
$$p(z_t|\boldsymbol{x}_1, \dots, \boldsymbol{x}_t) \propto \alpha_t(z_t). \tag{223}$$
2. **Smoothing.** Note that
$$p(z_t|\boldsymbol{x}_1, \dots, \boldsymbol{x}_T) \propto \alpha_t(z_t)\beta_t(z_t). \tag{224}$$
3. **Probability of sequence.** We want
$$p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T) = \sum_{z_T=1}^{K} \alpha_T(z_T). \tag{225}$$
Recall that $\alpha_T(z_T) = p(z_T, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$ so we simply need to take the joint distribution and marginalize out all values of T.
4. **Predict measurements.** We wanted
$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_1, \dots, \boldsymbol{x}_t) = \frac{p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_{t+1})}{p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_t)} = \frac{\sum_{z_{t+1}} \alpha_{t+1}(z_{t+1})}{\sum_{z_t} \alpha_t(z_t)}. \tag{226}$$

20.4.5 Viterbi algorithm for best path problem

The only problem we do not know how to solve is the best path problem. Recall the best path is
$$\operatorname*{argmax}_{\boldsymbol{z}} p(z_1, \dots, z_T|\boldsymbol{x}_1, \dots, \boldsymbol{x}_T). \tag{227}$$
This can be done using the *Viterbi algorithm*, which is a dynamic programming algorithm.

First we recursively define a function γ by
$$\gamma_t(z_t) = \begin{cases} p_0(z_1)p_\Omega(\boldsymbol{x}_1|z_1), & t = 1 \\ \underbrace{\left[\max_{z_{t-1}} \gamma_{t-1}p_T(z_t|z_{t-1})\right]}_{\text{best path to } z_t} p_\Omega(\boldsymbol{x}_2|z_2), & t \neq 1\ . \end{cases} \tag{228}$$
After finding all the γ's, we choose each state z_t^* in the optimal path based on the greatest $\gamma_t(z_t)$. We start with z_T^* and work backwards. We will need all $T \times K$ possible values of γ for different combinations of time steps and state values.
$$z_t^* = \operatorname*{argmax}_{z_t} \gamma_t(z_t)p_T(z_{t+1}^*|z_t). \tag{229}$$
Each $\gamma_t(z_t)$ refers to a likelihood for the most probable path for getting to z_t. z_t^* is fixed because we are working backwards, so $p_T(z_{t+1}^*|z_t)$ is like incorporating information about the likelihood of going from $z_t \to z_{t+1}^*$.

20.5 Solving questions in collection 2

We will now briefly discuss how to find p_0, T, Ω.

Note. If we had z, then solving the MLE of p_0, T, Ω is easy. We just get the empirical distribution over the z_0s to get p_0. For each z, we find how often each z' happens to find the transition matrix. *We can collect the best guesses for globals by collecting data.*

The process from above tells us how to get z given the globals.

Combining these two ideals, we can apply block coordinate ascent to iteratively solve for the global and local unknowns. This is the EM algorithm!

21 April 12th, 2022

Last class, we finished the cube. We learned about supervised and unsupervised learning to make predictions. Today, we transition to decision making.

More specifically, we will cover Markov decision processes.

Example 21.1 (Map navigation)**.** Consider finding the best route from beginning to destination. We need to model to figure out traffic information and speed of routes. We now need to make a decision about which route to take. We may have different objectives: the route that is shortest in expectation, expected arrival time, etc. To formalize these decisions, we need to define a *reward function*.

Example 21.2 (Autonomous vehicles)**.** There are many predictions when it comes to autonomous vehicles. Once we have all the data (where the road is, where the lines are, if there are signs, etc.), we need to decide how the car should drive to be safe.

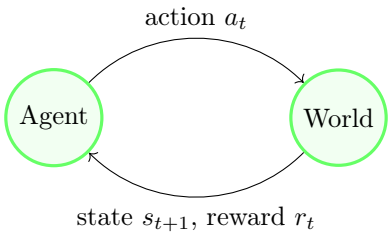
Example 21.3 (Alpha Go)**.** AlphaGo is a computer program that plays the board game Go. It was developed by DeepMind Technologies a subsidiary of Google.

21.1 Markov decision processes

We first define a few concepts in reinforcement learning. First, we have a state. This is something we can observe about the world. We then have a decision problem regarding choosing an appropriate action from the state. The action will come with a reward and the learning problem is to optimize the choice of actions given states to maximize the reward.

At the most basic level, a Markov decision process is a framework for modeling decision-making. In the model, there is one agent and the agent needs to make decisions while interacting with the world.

One representation is given below. At time t, we have



21.1.1 Notation, definitions, goal

The agent can send actions to the world, and the world will return observations $\boldsymbol{o}$. and rewards $\boldsymbol{r}$.

We let $S = \{1, \dots, |S|\}$ denote states, $A = \{1, \dots, |A|\}$ denote actions, $p(s'|s, a)$ denote the *transition model* and $\pi(a|s)$ the probability of taking an action a given we are in state s. This is the *policy*.

Moreover, we have $\boldsymbol{r}$ the rewards, t the time stamp and $\gamma \in [0, 1)$ the discount factor.

Definition 21.4 (Markov chain)**.** In a Markov chain, we have

$$p_t(s_{t+1}|s_1, s_2, \dots, s_t, a_1, a_2, \dots, a_t) = p_t(s_{t+1}|s_t, a_t). \tag{230}$$

We also assume the stationary assumption:

$$p_t(S_{t+1}|S_t, a_t) = p_t(S_{t+1}|S_t, a_t). \tag{231}$$

We note that this is the expected reward over all time where we discount additional rewards at time t by factor γ^t, which is *exponential discounting*.

21.1.2 Types of problems

The MDPs we are trying to solve are dependent on the set of variables given by $\text{MDP}(S, A, r, P)$ where the state and action spaces are discrete.

Note. The transition and reward functions can be more easily computed.

We consider the following problems:

- Planning: Input is MDP and output is optimal policy
- Reinforcement learning: Input is access to the world and outputs are actions. This is the same as learning from feedback to optimize a policy.

We want to find the best policy. We use π to denote a policy. Policies can be stochastic or describe a distribution over actions $\pi(s, a) = P(a|s)$ or deterministic and equal to one action $\pi(s) = a$.

Thus we want to choose actions to maximize

$$\max_{\pi} \mathbb{E}_{\text{world}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{232}$$

21.1.3 State definition

Definition 21.5 (State)**.** A *state* summarizes a history such that making predictions about the future given the state is equivalent to making those same predictions given the entire history.

This will be our interpretation. Some papers use approximations of states.

Thus we can now define our Markov decision process.

Definition 21.6 (Markov decision process)**.** A *Markov decision process* is a tuple

$$(S, A, R(s, a, s'), T(s'|s, a), \gamma, p_0) \tag{233}$$

where

- S are all possible states
- A are all possible actions
- $R(s, a, s')$ is the reward for doing action a in starting state s and landing in state s'
- $T(s'|s, a)$ is a matrix of transition probabilities that describe $p(s'|s, a)$, the probability of landing in state s; from action a starting in state s
- γ is the discount factor
- p_0 is some starting state

We can have different objectives:

- Infinite horizon: We want to find the policy π that maximizes

$$\max_{\pi} \mathbb{E}_{T,p_0} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{234}$$

- Finite horizon: This is the same as the above, but we want to have convergence in expectation

$$\max_{\pi} \mathbb{E}_{T,p_0} \left[\sum_{t=0}^{T} r_t \right]. \tag{235}$$

21.2 Solving using value iteration

In this lecture and the following one, we will cover three methods of solving MDPs. We will first explore value iteration. This is similar to dynamic programming because it evaluates the optimal actions based on best values it can achieve in a future time step.

21.2.1 Finite horizon planning

We define $V^*_{(t)}(s)$ as the total value from from state s under optimal policy with t steps to go.

Note. Here, t denotes the *remaining steps* and does **not** count the current time step.

Note (Principle of optimality)**.** The *principle of optimality* states that an optimal policy consists of an optimal first action following by an optimal policy from the successive state.

This brings us to our algorithm:

1. **Base case:** $V^*_{(1)}(s) = \max_a r(s,a)$
2. **Inductive case:** We define

$$V^*_{(t+1)}(s) = \max_a \left\{ r(s,a) + \sum_{s' \in S} p(s'|s,a) V^*_{(t)}(s') \right\}. \tag{236}$$

Note. We have already calculated $V^*_{(t)}(s')$ which is similar to dynamic programming.

The computational complexity is $O(T|S||A|L)$ where L is the maximum number of states reachable from any states through any action.

21.2.2 Infinite time horizon

Assume a *deterministic policy* $\pi(s) \in A$. We edfine the MDP value function

$$V^\pi(s) = \mathbb{E}_{s\sim p}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_k)) | s_0 = s \right]. \tag{237}$$

This essentially refers to the expected discounted value from policy π in state s:

$$\boxed{V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s' \in S} p(s'|s,\pi(s)) V^\pi(s').} \tag{238}$$

We define the optimal policy to be

$$\pi^* \in \operatorname*{argmax}_\pi V^\pi(s) \tag{239}$$

for all possible states. The optimal value function is defined to be

$$V^*(s) = V^{\pi^*}(s). \tag{240}$$

We will continue to expand on solving this through the Bellman equation next week.

Note (Bellman consequences)**.** For *finite* and *discrete* settings, the Bellman equation implies that policy evaluation is easy! We observe that the above equation is a linear system with $|S|$ unknowns for each $V^\pi(s)$.

We can write this in vector form. If V^π is a vector of values for all states and R^π be the vector $R(s,\pi(s))$, then

$$V^\pi = R^\pi + \gamma T^\pi V^\pi \implies \boxed{V^\pi = (I - \gamma T^\pi)^{-1} R^\pi} \tag{241}$$

where T^π is our transition matrix.

22 April 14th, 2022

Announcements

- Midterm 2 will cover unsupervised learning and our decision-making unit
- Homework 6 is out tomorrow!
- Today's relevant textbook sections are Sutton and Barto chapter 4 and sections 6.4, 6.5

Today, we continue our unit on sequential decision-making.

22.1 Review of MDPs

Recall that MDPs are defined by

$$\{S, A, r, p\} \tag{242}$$

where S is a set of possible states, A is a set of possible actions, $r(s, a)$ is the reward function, $p(s'|s, a)$ is the transition function. Our goal is to find the policy $\pi(s)$ that maximizes

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{243}$$

the expected sum of discounted rewards.

Today, we continue our discussion on planning and introduce reinforcement learning. These are both MDP problems, but are not the same problem.

22.2 Infinite horizon planning

We have an MDP value function

$$V^{\pi}(s) = \mathbb{E}_{s\sim p}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))|s_0 = s\right] \tag{244}$$

where γ is a discount factor describing how we discount rewards going to the future. Policy π is better than or equal to policy π' if $\forall s \in S$

$$V^{\pi}(s) \geq V^{\pi'}(s). \tag{245}$$

We say π^* is optimal if it is better than or equal to all other policies π.

From π^*, we can get the optimal value function: the value function associated with the optimal policy, defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s). \tag{246}$$

Our goal is to find the optimal policy. To accomplish this, we will try to calculate the optimal value function. Once we know the optimal value functions, we can extract the optimal policy from it by picking the best action at each state. We can get the optimal step at every state using

$$\pi^*(s) \in \operatorname*{argmax}_a \left\{ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^*(s') \right\}. \tag{$*$}$$

The principle we use for the calculation is the Bellman equation, or the *principal of optimality*:

$$V^*(s) = \max_a \left\{ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^*(s') \right\}. \tag{$\square$}$$

Note. This is saying that the value of a state s is the sum of the value of taking the best action on s and the value of continuing to take the optimal actions in future steps.

22.3 Value iteration

We can solve (□) iteratively as follows:

1. Initialize $V(s) = 0$ for all s
2. Repeat the following:
    1. Set
$$V'(s) = \max_a \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V(s') \right\} \tag{247}$$
    2. Replace $V(s)$ with $V'(s)$ for all s

Once we have an approximation for V^*, we can extract the corresponding approximation of the optimal policy using (∗) from the above.

Theorem 22.1. *Using the value iteration algorithm above, V will converge to the optimal value function V^*.*

22.3.1 Correctness

We will provide a sketch of why this works.

Consider a function $f : \mathbb{R}^D \to \mathbb{R}^D$ and an update step $x' \leftarrow f(x)$. We say x^* is a fixed point $\iff f(x^*) = x^*$.

We define the contraction property.

Definition 22.2 (Contraction)**.** f is a *contraction* if $\forall x \neq y$:
$$\|f(x) - f(y)\| < \|x - y\|. \tag{248}$$

Theorem 22.3. *f contraction $\implies \exists! x^* : f(x^*) = x^*$ and $x' \leftarrow f(x)$ will converge to the fixed point.*

Proof. Let f a contraction mapping. First show that a fixed point is unique. AFTSOC not unique. Then $\exists x^*, y^*$:
$$\|f(x^*) - f(y^*)\| = \|x^* - y^*\| \tag{249}$$
but this violates contraction property $\implies$ f has a unique fixed point.

We also want to show if we repeatedly set $x' \leftarrow f(x)$, we will converge to x^*. It follows from contraction that
$$\|x - x^*\| > \|f(x) - f(x^*)\| = \|f(x) - x^*\| \tag{250}$$
and thus we get closer to the fixed point if we update our x'.

The Bellman operator B is a contraction is we use norm $\|V\| = \max_s |V(s)|$ so the value iteration function has a fixed point. □

Note. The value iteration converges to V^* asymptotically. Even after the optimal policy stops changing, the value function values will continue changing.

The optimal policy can be extracted from V in a finite number of steps.

The method is similar to the finite time horizon planning problem from last week. The difference is that we do not have a set number T of limited steps.

22.4 Policy iteration

In policy iteration we start with policy π^0 and we repeated to the following:

1. Determine the value function V^π for the current policy.
2. Use the value function to improve the policy for all s using

$$\pi'(s) \leftarrow \underset{a}{\text{argmax}} \left\{ r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi(s') \right\} \tag{251}$$

3. Update next π to be π'.

Policy iteration alternates between evaluating a policy and updating the policy.

Note. In value iteration, we do not keep track of policy. We extract it from the value function at the end.

Theorem 22.4. *Policy iteration converges to optimal policy in a finite number of steps and will also produce the optimal value function.*

Note. We were not guaranteed the optimal value function in finite steps with value iteration.

Sketch. If the policy is already optimal, the update step will not change it. If *not* optimal, the update step will provide *better steps* to take. Imagine that for some s there is an action a' different from the policy recommendation $\pi(s) = a \neq a'$:

$$r(s,a') + \gamma \sum_{s}^{'} p(s'|s,a) V^\pi(s') > V^\pi(s). \tag{252}$$

This means that taking a' and following π is better than following π, so we should update π to recommend action a' at state s.

Thus it holds that each time the policy gets updated in policy iteration, $V^{k+1} > V^k$. □

22.4.1 Notes

Given π we can solve a system of equations to get V^π, where for each s we have an equation of the form

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^\pi(s') \tag{253}$$

since we have $|S|$ unknowns and $|S|$ equations. In matrix form, we have

$$V^\pi = R^\pi + \gamma T^\pi V^\pi \iff V^\pi = (I - \gamma T^\pi)^{-1} R^\pi \tag{254}$$

where T is the transition matrix and $I - \gamma T^\pi$ is a matrix with full rank.

22.4.2 Comparing algorithms

Policy iteration takes fewer iterations than value iteration to solve for π^*. If L is the maximum number of reachable states, the work for value iteration is $O(|S||A|L)$ while policy iteration takes $O(|S||A|L + |S|^3)$ per iteration.

22.5 Reinforcement learning

In planning, we had a full model of the environment, we knew the transitions and reward functions.

In RL, we want to learn from the environment when given no knowledge about r or p. This poses a new challenge — we have to balance exploring vs. exploiting to do well.

There are two main approaches.

22.5.1 Model-based approach

We try to learn a model of the environment, which allows us to predict what the next state and reward will be. The planning is used to decide how to act based on the estimated model.

If reward or transition structure changes, model-based learning is good at absorbing these changes into its model. However, model-based learning is computationally expensive.

22.5.2 Model-free approach

We don't try to learn a model and instead, we directly learn a policy or an action-value function. We refer to the action-value function as the Q-function. Model-free learning is much simpler and cheaper, but if the environment changes, we have to do more acting and exploring to learn the changes.

22.6 Value-based methods for model-free RL: Definitions

We try to learn a Q-function. We can have Q^π for a specific policy π:

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi(s') \tag{255}$$

where $r(s,a)$ is the value for taking action a now, and the second term is the expected value for the future following policy π.

Note. In the value function $V(s)$, we do not input the action. The value function automatically assumes we are following the optimal action at that step. In the Q-function, we provide both the state and action so we can have a value for an action even if it is **not optimal**.

There is also the *optimal Q-function*:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^*(s'). \tag{256}$$

Then the optimal policy comes from picking the action with the best Q-value at each state:

$$\pi^* = \operatorname*{argmax}_a Q^*(s,a). \tag{257}$$

The Bellman equation for Q^* is

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q^*(s',a') \tag{258}$$

where

$$\max_{a'} Q^*(s',a') = V^*(s') \tag{259}$$

22.7 Value-based methods for model-free RL: Algorithms

The general framework for RL algorithms is to initialize a $|S| \times |A|$ table of Q-values $Q(s,a)$ and repeated choose an action based on the Q-values, and use the data we collect from the action to update the Q-table.

In each step, the data is collected in the form s, a, r, s', a'. s is the current state, a is the action the agent took, r is the reward received, s' is the next state and a' is the action the policy recommends.

Note. We do not have access to the reward function or the transition matrix, but as the agent moves in the environment, it collects data about the reward it received.

Our agent decides how to act in an ϵ-greedy manner. This means it follows a policy that takes the optimal action (based on previous observations) with probability $1-\epsilon$ and chooses a random action to explore with probability ϵ:

$$\pi(s) = \begin{cases} \text{argmax}_a\, Q(s,a), & \text{with probability } 1-\epsilon \\ \text{random}, & \text{with probability } \epsilon \end{cases} \tag{260}$$

We will learn Q^* through *temporal difference updates*.

22.7.1 SARSA

Each time we get a new experience (s, a, r, s', a'), we make the following Q-update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \left[r + \gamma Q(s', a') - Q(s,a) \right] \tag{261}$$

where α_t is the learning rate that can vary based on how many steps have passed so far, $r + \gamma Q(s', a')$ is our one-step estimate of $Q^{\pi(s,a)}$. We take the difference between the estimate and the estimate Q-value made by our current Q-function.

Note. This is kind of like a gradient-descent update.

22.7.2 Q-learning

Each time we get a new experience (s, a, r, s'), we do the following update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right]. \tag{262}$$

Note. The difference between the two methods is that we take the Q-value for the best next step by taking the max over all possible a'. In SARSA, we update our Q-function as though the only option for the next action is a'.

Note. SARSA is known as on-policy because it estimates Q^π following the current policy when it only considers a' in the update. It estimates the optimal Q^* while following π.

Q-learning is called off-policy because the estimate might use the next action different than what we would take if we followed the current policy. Q-learning converges to Q^* as long as (s, a) are visited often enough.

22.8 Lecture notes

22.8.1 Value iteration

Recall we define the *action-value function*

$$\boxed{Q^\pi(s,a) \equiv r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\pi(s').} \tag{263}$$

Note. Intuitively, the Q-function is interpreted as a *one-step excursion*, and following our policy again after our single deviation.

We can compare $Q^\pi(s,a)$ to $V^\pi(s)$. We note $V^\pi(s) = Q^\pi(s, \pi(s))$ because V^π is the value if we follow our policy without an excursion.

We can alternatively formulate value iteration as follows:

1. Start with some arbitrary $Q_{k=0}(s,a)$. Here k is the iteration number.
2. Until convergence, set

$$Q_k(s,a) \leftarrow r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q_{k-1}(s',a') \tag{264}$$

where $r(s,a)$ is the immediate reward for the present, γ is the discount factor, $\sum_{s'} p(s'|s,a)$ is the expectation over future, and $Q_{k-1}(s',a')$ is the guess of the value of doing a' in s' *so far* and $\max_{a'}$ tells us to take the best of our choices.

Note. This does not assume we take our particular action.

Note. $Q_k(s,a)$ will converge to $Q^*(s,a)$ with the optimal policy and we can read off $\pi^*(s) = \text{argmax}_a\, Q^*(s,a)$. Also π may converge before Q_k.

22.8.2 Policy iteration

Recall we want to find the *best* policy π. We rely on the following theorem.

Theorem 22.5 (Policy improvement)**.** *If we set*

$$\pi(s) \leftarrow \operatorname*{argmax}_a Q^\pi(s,a), \tag{265}$$

we get an improved policy or no worse.

This implies the following algorithm:

1. Start with some policy π_0.
2. Evaluate or compute $Q^\pi(s,a)$ and improve π using the policy improvement theorem.
3. Repeat step 2 until convergence.

Note. In nice settings, we will converge on some optimal policy. These optima are **not unique**.

23 April 19th, 2022

Announcements

- Midterm 2 is in about one week!

Today, we discuss reinforcement learning.

23.1 Reinforcement learning

We consider situations where we do **not** have the transition function or the rewards. How do we find the optimal policy?

When we don't have T, R we have a trade-off between exploring (learning more about T, R) or exploiting (optimize with respect to T, R). We have the same goal of maximizing $\mathbb{E}\left[\sum_t \gamma^t r_t\right]$.

There are three types of algorithms:

- Build model to learn T, R and plan.
- Value-based methods, that turn into some form of value iteration
- Policy-based, optimizing $\pi(s)$ directly

23.2 Model-based learning

We want to learn a model for T, R and then optimize with respect to T, R to find π^*. We then collect some data and update T, R.

Note. Recall that we want to collect data and balance between exploration and exploitation. We used ϵ-greedy action selection!

There are two ways to sample and get the right data to build our model.

23.2.1 Optimism under uncertainty

Let $N(s, a)$ be the number of times we visit the state-action pair (s, a). Each visit is a time that the agent is in state s and chooses to perform action a. When we learn, we assume that if $N(s, a)$ is small ,then the next state has high reward. When we do this, we are being optimistic about the state-action pairs we are uncertain about. Then we plan using the optimistic model.

Note. Because the model thinks taking visiting lesser-known state-action pairs lead to high reward, we have a tendency to explore lesser-known areas.

23.2.2 Posterior sampling (Thompson sampling)

We maintain a posterior (Dirichlet) on the transition function $p(\cdot|s, a)$. We sample from the posterior to get a model that we proceed to plan with. The idea is that when we are less certain about the outcome of the transition function, there is greater likelihood of the sampling leading us to explore. When we are *more certain* about the best outcome, we are more likely to stick with the best outcome.

We maintain a distribution over models. We sample $T, R \sim p(T, R|\mathcal{D})$. We then find π^* with respect to T, R and we repeat by updating $p(T, R|\mathcal{D})$. We start by assuming everything is highly uncertain. As we gather data, the posterior will contract.

Note. Model-based approaches tend to be computationally expensive.

23.3 Model-free value-based approaches

We are no longer learning a model and planning. Instead, feedback from the world is directly used to update the agent.

Recall from last lecture about Q-functions

$$Q^\pi(s,a) = \underbrace{R(s,a)}_{\text{reward after action } a} + \underbrace{\gamma \sum_{s'} p(s'|s,a)}_{\text{expectation over where next}} \times \underbrace{V^\pi(s')}_{value of next} \tag{266}$$

In a discrete setting, we can consider the following *value-based* method.

1. We can initialize $Q(s,a)$ in a $S \times A$ table.
2. We then take an action.

   Note (ϵ-greedy)**.** One strategy to take the action is ϵ-greedy. It is given as follows:

   $$\pi(s) = \begin{cases} a, & \text{with probability } \epsilon \\ \text{random}, & \text{with probability } 1-\epsilon \end{cases} \tag{267}$$

3. We then update Q given (s,a,r,s',a'). We give two approaches below, SARSA, and Q-learning.

23.3.1 SARSA

This is *on-policy*. It learns Q-values corresponding to the agent's behavior, so what the agent learns depends on how the agent is behaving.

We update Q by

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t(s,a) \overbrace{[\underbrace{r + \gamma Q(s',a')}_{\text{one-step estimate}} - Q(s,a)]}^{\text{temporal difference}} \tag{268}$$

and α_t is our learning rate.

Note. If we *always* to $\pi_{\text{explore}}(s)$ and run SARSA, $Q(s,a)$ converges to Q^{explore}.

The intuition of SARSA is the interweaving of policy evaluation of current policy and policy optimization which is acting greedily. We will converge to π^*, Q^* at the end.

This is because we are evaluating our policy, but because we are changing what we are doing, this still converges to the optimal.

23.3.2 Q-learning

This is an *off-policy* approach. We learn Q-values Q^* corresponding the the Bellman equation

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t(s,a)[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{269}$$

Note. Our learning rate α_t is typically decayed. We want $\sum \alpha_t \to \infty$ and $\sum \alpha_t^2 \to C$.

In Q-learning, we are always taking an action a' that may not necessarily be $\pi(s)$. This is empirically faster than SARSA.

Note. Under certain conditions, Q-learn converges to Q^*. This comes from the Bellman equation and contraction property-esque ideas similar to last time.

Theorem 23.1 (Q-learning convergence)**.** $\forall s, a$ *if* $\sum_t \alpha_t(s,a) = \infty, \sum_t \alpha^2(s,a) < \infty$ *then* Q*-learn converges to* Q^* *as* $t \to \infty$*.*

Note. For SARSA convergence to Q^*, we need the following condition, that behavior is greedy in the limit. To satisfy this condition, we set $\epsilon_t(s) = \frac{1}{N_t(s,a)}$ so ϵ decreases as we visit each state-action pair more often.

23.4 Deep Q-networks

By combining Q-learning with neural networks, researchers have achieved amazing results like playing video games.

Because there are so many states in a game, a table does not work. Instead, we can parameterize $Q(s, a; \boldsymbol{w})$ and use a differentiable deep network with parameters $\boldsymbol{w}$. The network will take in a state and generate predictions for Q-values of taking each action from the state. We will use gradient descent updates to find a model that minimizes the squared error.

24 April 21st, 2022

Announcements

- Midterm Tuesday!
- There will be office hours on Sunday
- Homework 6 is due Friday
- Practical is due the following Friday

Recall from last time, we discussed reinforcement learning and some value-based approaches like SARSA and Q-learning.