CS 181 Notes

Elvin Lo

Spring 2024

2024.3.15

Preface

These notes follow *Undergraduate Fundamentals of Machine Learning*, the text accompanying CS 181 at Harvard College.

Contents

2 Regression

2.1 Defining the problem

Definition 2.1. Regression
A class of techniques that seeks to make predictions about unknown continuous target variables given observed input variables.

2.2 Solution options

Regression algorithms include KNN, neural networks, RFs, gradient boosted trees, and of course linear regression.

Definition 2.2. Non-parametric regression with KNN
Linear regression may be solved with KNN by simply averaging the target values of the K nearest neighbhors. Similarly, we might take a weighted average of target values with kernel regression.

Note that a non-parametric model is most likely to overfit; non-parametric means infinite-dimensional parameters, which means a more flexible model.

2.3 Introduction to linear regression

Definition 2.3. Linear regression
Suppose we have an input $\mathbf{x} \in \mathbb{R}^D$ and a continuous target $y \in \mathbb{R}$. Linear regression determines weights $w_i \in \mathbb{R}$ that combine the values of x_i to produce y, generally written as

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D.$$

The intercept w_0 is called the *bias*, and it accounts for data with non-zero mean. Intuitively, the linear regression problem is finding the best fitting line.

2.4 Basic Setup

Remark 2.4. Merging of bias
We introduce the bias trick, a common notational trick to make the bias term w_0 easier to handle. By introducing another variable x_0 that is always 1 for every data point, we have

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D.$$

2.5 Finding the best fitting line

Definition 2.5. Objective function, aka loss
A function that measures the "goodness" of a model. We can optimize this function to identify the best possible model for our data. We shall notate by $\mathcal{L}(\mathbf{w})$ the loss incurred for an entire data set by the model $\mathbf{w}$, and by $\mathcal{L}_i(\mathbf{w})$ the loss incurred for the i^{th} data point.

Definition 2.6. Residual
The residual is the difference between the target (y) and predicted value that a model produces:

$$\text{residual} = \text{target} - \text{prediction} = y - f(\mathbf{x}, \mathbf{w}) = y - \mathbf{w}^\top \mathbf{x}.$$

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

Definition 2.7. L1 and L2 losses
In linear regression, we most commonly have *L1 and L2 losses*, the sum of the absolute values and the sum of the squares of all the residuals, respectively. Solutions to $\mathbf{w}$ minimizing these have different properties, e.g.,

- L2 loss produces only a single solution to $\mathbf{w}$, while L1 loss can potentially have many equivalent solutions.
- L2 loss is not robust to outliers, but L1 loss produces unstable solutions (i.e., small changes in our data set may induce large changes in our solution).

Unlike L1, the L2 loss function is quadratic and so continuously differentiable, making optimization convenient.

2.6 Linear regression algorithms

Linear regression and calculating the optimal weights $\mathbf{w}$ may be understood through several perspectives.

Example 2.8. Linear regression with least squares loss
We may solve for the optimal weights $\mathbf{w}$ analytically, using a least squares loss. Let $N \times D$ matrix $\mathbf{X}$ be the design matrix, where row n contains the features $\mathbf{x}_n^\top$. Denote by $N \times 1$ vector $\mathbf{Y}$ the corresponding the target values, and define an L2 loss (scaled for computational convenience):

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y_n - \mathbf{w}^\top \mathbf{x}_n \right)^2.$$

To derive optimal $\mathbf{w}$, we first take the gradient w.r.t. $\mathbf{w}$,

$$\nabla \mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} \left(y_n - \mathbf{w}^\top \mathbf{x}_n \right) (-\mathbf{x}_n).$$

Here it is often convenient to rewrite our summations as matrix operations with $\mathbf{X}$ and $\mathbf{y}$. To solve for $\mathbf{w}$ such that $\nabla\mathcal{L}(\mathbf{w}) = 0$, we may write

$$\sum_{n=1}^{N} y_n \mathbf{x}_n - \sum_{n=1}^{N} \mathbf{x}_n \left(\mathbf{x}_n^\top \mathbf{w}\right) = \mathbf{X}^\top \mathbf{y} - \mathbf{X}^\top \mathbf{X}\mathbf{w} = 0,$$

yielding solution

$$\mathbf{w}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y},$$

well-defined if $\mathbf{X}$ has full column rank (features are not colinear) such that $\mathbf{X}^\top\mathbf{X}$ is positive definite and the inverse exists. Note the quantity $\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top$ is the (left) Moore-Penrose pseudoinverse.

Example 2.9. Linear regression from a generative view

We can also consider that our data was generated with Gaussian noise, and solve for $\mathbf{w}$ maximizing the likelihood of our dataset. Consider $D = \{(\mathbf{x}_n, y_n)\}$ and imagine our data was generated according to the following process:

$$y_n \sim \mathcal{N}\left(\mathbf{w}^\top \mathbf{x}_n, \beta^{-1}\right),$$

where we denote the variance as the inverse of hte precision β, since the precision is sometimes nicer to work with. Equivalently,

$$p\left(y_n \mid \mathbf{x}_n, \mathbf{w}, \beta\right) = \mathcal{N}\left(\mathbf{w}^\top \mathbf{x}_n, \beta^{-1}\right).$$

The likelihood of our data set is given by:

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}\left(\mathbf{w}^\top \mathbf{x}_n, \beta^{-1}\right)$$

To calculate the MLE, we take the gradient of the log-likelihood

$$\begin{aligned}
\ln p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \beta) &= \sum_{n=1}^{N} \ln \mathcal{N}\left(\mathbf{w}^\top \mathbf{x}_n, \beta^{-1}\right) \\
&= \sum_{n=1}^{N} \ln \frac{1}{\sqrt{2\pi\beta^{-1}}} e^{-\left(y_n - \mathbf{w}^\top \mathbf{x}_n\right)^2 / 2\beta^{-1}} \\
&= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \frac{\beta}{2} \sum_{n=1}^{N} \left(y_n - \mathbf{w}^\top \mathbf{x}_n\right)^2, \\
\frac{\partial}{\partial \mathbf{w}} \ln p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \beta) &= -\beta \sum_{n=1}^{N} \left(y_n - \mathbf{w}^\top \mathbf{x}_n\right)\left(-\mathbf{x}_n\right),
\end{aligned}$$

and solve for $\mathbf{w}$, yielding

$$\sum_{n=1}^{N} y_n \mathbf{x}_n - \sum_{n=1}^{N} \left(\mathbf{w}^\top \mathbf{x}_n\right) \mathbf{x}_n = 0.$$

This is exactly the same form as before:

$$\mathbf{w}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}.$$

so we see that minimizing a least squares loss is equivalent to maximizing the probability under the assumption of a linear model with Gaussian noise.

As another remark, linear regression may be interpreted as a projection of our targets onto the column space of our inputs, see Section 2.6.3.

2.7 Model Flexibility

Definition 2.10. Basis function
A basis function $\phi(\cdot)$ is a transformation applied to an input data point $\mathbf{x}$ to move our data into a different input basis/domain. For example, consider our original data point:

$$\mathbf{x} = \left(x^{(1)}, x^{(2)}\right)'$$

We may choose our basis function $\phi(\mathbf{x})$ such that our transformed data point in its new basis is:

$$\phi(\mathbf{x}) = \left(x^{(1)}, x^{(1)^2}, x^{(2)}, \sin\left(x^{(2)}\right)\right)'$$

Sometimes we will also describe our input data points as $\boldsymbol{\phi} = \left(\phi^{(1)}, \phi^{(2)}, \ldots, \phi^{(D)}\right)'$. Choosing the appropriate basis is often difficult; domain specific knowledge might help, but more often than not we won't have this expert knowledge.

Definition 2.11. Generalization
Generalization is the ability of a model to perform well on new data points outside of the training set.

Definition 2.12. Regularization
Regularization is applying penalties to parameters of a model. A convoluted line that matches the noise of our training set exactly isn't going to generalize well, so we sometimes penalize the total size of our weights $\mathbf{w}$ such that we favor simple regression lines that take advantage of only the most important basis functions. There is a tradeoff between how aggressively we regularize our weights and how tightly our solution fits to our data.

Example 2.13. Regularizing least squares loss
In general, we introduce a regularization parameter λ and write

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \mathbf{w}^\top \boldsymbol{\phi}_n\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_h^h.$$

As a first example, with $h = 2$ we have L2 norm regularization or ridge regression:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \mathbf{w}^\top \boldsymbol{\phi}_n\right)^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}.$$

Note in the limit $\lim_{\lambda\to\infty}\mathcal{L}(\mathbf{w})$, we will drive all weights to 0. Now let's examine a few types of regression:

- **Ridge Regression**: By using the L2 norm and thus squaring, ridge regression punishes any individual weight from growing too large, providing us with solutions that are generally moderate.
- **Lasso**: By using $h = 1$, lasso will drive some parameters w_i to zero if they aren't as informative. Lasso helps recover a sparser solution that lets us throw out some of our basis functions.
- **Elastic Net**: Uses a linear combination of the previous two regularization terms to get a middle ground.

Proposition 2.14. From Bayesian perspective, regularization is adding a prior
In Bayesian linear regression, we may regularize our weight parameters by specifying a prior distribution over $\mathbf{w}$.

In the Bayesian framework, we have a distribution over $\mathbf{w}$, intuitively meaning we average over different models specified by different values of $\mathbf{w}$. Then fitting our model on data corresponds to updating our distribution with Bayes', and in particular, the posterior distribution is proportional to the posterior likelihood times the prior. Thus specifying our prior will make our posterior tend toward particular values of $\mathbf{w}$, equivalently regularizing our weights $\mathbf{w}$. See details in Derivation 2.7.1.

2.8 Choosing Between Models

Model selection is important because of the bias-variance tradeoff.

Definition 2.15. Bias-Variance Tradeoff
When constructing machine learning models, we have a choice somewhere on a spectrum between two extremes: fitting exactly to our training data (high variance) or not varying in response to our training data at all (high bias). Formally, consider a model $f(\cdot)$ predicting y given $\mathbf{x}$. We have the MSE

$$\text{MSE} = \mathrm{E}_{\mathbf{D},y|\mathbf{x}}\left[(y - f_{\mathbf{D}}(\mathbf{x}))^2\right],$$

where $f_{\mathbf{D}}$ denotes the fitting of our model f on the dataset $\mathbf{D}$. The expectation is taken with respect to both our data set $\mathbf{D}$ (variation in our modeling due to our training data) and our conditional distribution $y \mid \mathbf{x}$ (noise in our training data). We may decompose the MSE into

$$\begin{aligned}\text{MSE} &= \mathrm{E}_{y|\mathbf{x}}\left[(y - \bar{y})^2\right] + (\bar{y} - \bar{f}(\mathbf{x}))^2 + \mathrm{E}_{\mathbf{D}}\left[\left(\bar{f}(\mathbf{x}) - f_{\mathbf{D}}(\mathbf{x})\right)^2\right] \\ &= \text{noise}(\mathbf{x}) + \text{bias}(f(\mathbf{x}))^2 + \text{variance}(f(\mathbf{x})),\end{aligned}$$

where $\bar{y} = \mathrm{E}_{y|\mathbf{x}}[y]$ is the true conditional mean and $\bar{f}(\cdot) = \mathrm{E}_{\mathbf{D}}\left[f_{\mathbf{D}}(\cdot)\right]$, the prediction mean, is the expectation of our model function taken with respect to our random data set.

Proof. See Derivation 2.8.1. □

Definition 2.16. Overfitting and underfitting
Overfitting describes a convoluted model that is able to predict every point in our data set perfectly but which doesn't generalize well to new data points. Underfitting describes a model that doesn't respond to variation in our data. This tradeoff is why regularization is important.

Definition 2.17. K-fold cross-validation
We tune hyperparameters with cross-validation: we set some portion of a data set aside for validation, and use the rest for training. For example, we might perform cross validation for many different values of our regularization parameter λ, and choose the value minimizing the validation error.

In K-fold cross-validation, we perform cross-validation K times, allocating $\frac{1}{\mathbf{K}}$ of the data for validation at each iteration. The K-fold part helps evaluate the expected performance of a model independent of the training data.

Definition 2.18. Bayesian Model Averaging
We can also handle model selection using a Bayesian approach, accounting for our uncertainty about the true model by averaging over the possible candidate models, weighting each model by our prior certainty that it is the one producing our data. Formally, if we have M models, we can write the likelihood of observing our data set $\mathbf{X}$ as follows:

$$p(\mathbf{X}) = \sum_{m=1}^{M} p(\mathbf{X} \mid m)p(m)$$

where $p(m)$ is our prior certainty for a given model and $p(\mathbf{X} \mid m)$ is the likelihood of our data set given that model. The elegance of this approach is that we don't have to pick any particular model, instead choosing to marginalize out our uncertainty.

2.9 Linear Regression Extras

3 Classification

3.1 Defining the Problem

Definition 3.1. Classification
A set of problems that seeks to make predictions about unobserved target classes given observed input variables.

3.2 Solution Options

Definition 3.2. Generalized linear models
We will discuss three generalized linear models to produce a class prediction; each uses some linear combination of input variables. They are:

- Discriminant functions
- Probabilistic discriminative models (e.g. logistic regression)
- Probabilistic generative models

3.3 Discriminant Functions

Our goal is to linearly separate the input space into sections belonging to different target classes.

Definition 3.3. Discriminant functions
As with linear regression, discriminant functions $h(\mathbf{x}, \mathbf{w})$ seek to find a weighted combination of our input variables to make a prediction about the target class:

$$h(\mathbf{x}, \mathbf{w}) = w^{(0)}x^{(0)} + w^{(1)}x^{(1)} + \ldots + w^{(D)}x^{(D)}$$

where we are using the bias trick of appending $x^{(0)} = 1$ to all of our data points.

Example 3.4. Binary classification with discriminant functions
In the simple binary classification case with classes 1 and -1, we fit a discriminant function $h(\mathbf{x}, \mathbf{w})$ predicting

$$\begin{cases} +1 & \text{if } h(\mathbf{x}, \mathbf{w}) \geq 0 \\ -1 & \text{if } h(\mathbf{x}, \mathbf{w}) < 0, \end{cases}$$

and the decision boundary will be where $h(\mathbf{x}, \mathbf{w}) = 0$.

Remark 3.5. Multiple classes
Now consider the case that we have $K > 2$ classes $C_1, C_2, \ldots, C_K$ to choose between. We use K different linear classifiers $h_k(\mathbf{x}, \mathbf{w}_k)$, and then assign new data points to the class C_k for which $h_k(\mathbf{x}, \mathbf{w}_k) > h_j(\mathbf{x}, \mathbf{w}_j)$ for all $j \neq k$. Then, similar to the two-class case, the decision boundaries are described by the surface along which $h_k(\mathbf{x}, \mathbf{w}_k) = h_j(\mathbf{x}, \mathbf{w}_j)$.

There are also some naive approaches, but they lead to ambiguous regions:

- **One-versus-all approach**: Use K different discriminant functions that each determine whether or not a given input is in that class C_k. But several discriminator functions could claim that a data point is a part of their class.
- **One-versus-one approach**: Use $\binom{K}{2}$ discriminant functions that each determine whether a given point is more likely to be in class C_j or class C_k. This again gives us ambiguous regions.

3.4 Numerical Parameter Optimization and Gradient Descent

Definition 3.6. Gradient descent
Gradient descent is often used to fit complex model parameters. Notationally, we have

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla \mathcal{L}\left(\mathbf{w}^{(t)}\right)$$

where $\mathbf{w}^{(t)}$ is the state of the parameters $\mathbf{w}$ at time $t, \mathcal{L}\left(\mathbf{w}^{(t)}\right)$ is the gradient of our objective function, and $\eta > 0$ is the learning rate. The initial parameter values $\mathbf{w}^{(0)}$ are often initialized randomly. To compute the gradient at each step, we can use either

- batch gradient descent, computing the gradient using the entire data set;
- stochastic gradient descent (SGD), using a subset of the data (sometimes just a single data point).

SGD is typically more popular because (1) the computation time is often significantly smaller, and (2) we are less likely to get stuck in local minima while running SGD because a point in the parameter space that is a local minima for the entire data set combined is much less likely to be a local minima for each data point individually. Finally, SGD lends itself to being used for training online models (meaning models built on data points that are arriving at regular intervals) as the entirety of the data does not need to be present in order to train.

3.5 Objectives for Decision Boundaries

Definition 3.7. One-hot encoding
To encode class information, we often use one-hot encoding. The class of a given data point is described by a vector with K options, with a 1 in the position corresponding to C_k and 0s elsewhere. Note that classes aren't usually 0-indexed.

Definition 3.8. Hinge loss, ReLU
First we define ReLU, the rectified linear activation unit,

$$\text{ReLU}(z) = \max\{0, z\}.$$

This lets us incur error when we are wrong $z > 0$ and none when we are right $z < 0$. Againing using ± 1 to denote our two classes, we write

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{N} \text{ReLU}\left(-h\left(\mathbf{x}_i, \mathbf{w}\right) y_i\right) \\ &= -\sum_{y_i \neq \hat{y}_i}^{N} h\left(\mathbf{x}_i, \mathbf{w}\right) y_i \\ &= -\sum_{y_i \neq \hat{y}_i}^{N} \mathbf{w}^\top \mathbf{x}_i y_i,\end{aligned}$$

where $\hat{y}_i$ is the class prediction and y_i is the true class value. We have take the gradient

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{y_i \neq \hat{y}_i}^{N} \mathbf{x}_i y_i,$$

which allows us to optimize using SGD. The combination of hinge loss and SGD with discriminant functions to solve classification is called the *perceptron algorithm*. Overall, the benefits of hinge loss are:

- differentiability, which lets us optimize our weights (unlike 0/1 loss),
- penalizes more heavily data points that are more poorly misclassified (unlike 0/1 loss),
- doesn't penalize any correctly classified data points (unlike basic linear classification).

Now let us consider some alternatives to hinge loss, and discuss why they are poor.

Definition 3.9. 0/1 Loss
With the 0/1 loss function, we incur a loss of 1 if our model misclassifies a point, and no loss if our model classifies it correctly. While intuitive, the 0/1 loss is not differentiable or convex, and so has no closed form solution. It also gives no sense of how good a given prediction was.

Definition 3.10. Least squares loss for classification
While least squares gives us an analytic solution for our discriminant function, it penalizes data points that are 'too good', meaning they fall too far on the correct side of the decision boundary. Furthermore, it is not robust to outliers; the decision boundary significantly changes with the addition of just a few outlier data points. Nonetheless, we give the derivation.

We can again apply the least squares loss from linear regression in this context to find the set of weights $\mathbf{w}$. Each class C_k gets its own linear function with a different set of weights $\mathbf{w}_k$:

$$h_k(\mathbf{x}, \mathbf{w}_k) = \mathbf{w}_k^\top \mathbf{x}.$$

We can combine the set of weights for each class into a matrix $\mathbf{W}$, which gives us our linear classifier:

$$h(\mathbf{x}, \mathbf{W}) = \mathbf{W}^\top \mathbf{x}$$

where each row in the transposed weight matrix $\mathbf{W}^\top$ corresponds to the linear function of an individual class, and matrix $\mathbf{W}$ is $D \times K$. Assuming a data set of input data points $\mathbf{X}$ and one-hot encoded target vectors $\mathbf{Y}$ (where $\mathbf{Y}$ is $N \times K$), the optimal solution for $\mathbf{W}$ is

$$\mathbf{W}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y}.$$

3.6 Probabilistic Methods

3.6.1 Probabilistic Discriminative Modeling

Definition 3.11. Probabilistic Discriminative Modeling
Probabilistic modeling tries to directly model the conditional class distribution $p(y^* \mid \mathbf{x}^*)$ in order to make classification predictions. To do this, we perform discriminative training, directly optimizing the parameters of a conditional distribution.

Example 3.12. Logistic regression with sigmoid for binary classification
To do probabilistic discriminative modeling, we use the sigmoid function σ to compress the real line into the unit interval,

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

This transforms the output of our generalized linear models (by default real numbers) into probabilities. For example, in the binary classification example, the class probabilities of $\mathbf{x}^*$ are

$$p(y^* = C_1 \mid \mathbf{x}^*) = \sigma\left(\mathbf{w}^\top \mathbf{x}^*\right)$$
$$p(y^* = C_2 \mid \mathbf{x}^*) = 1 - p(y^* = C_1 \mid \mathbf{x}^*).$$

Then we can define the loss function as the negative log-likelihood of observing our data, called the logistic loss (or more generally cross-entropy loss): given data set $\{\mathbf{x}_i, y_i\}$, the likelihood of our weights is

$$p\left(\{y_i\}_{i=1}^N \mid \mathbf{w}\right) = \prod_{i=1}^N \hat{y}_i^{y_i} \{1 - \hat{y}_i\}^{1-y_i}$$

where $\hat{y}_i = p(y_i = C_1 \mid \mathbf{x}_i) = \sigma\left(\mathbf{w}^\top \mathbf{x}_i\right)$, and so we want to optimize

$$\mathrm{E}(\mathbf{w}) = -\ln p(\{y_i\} \mid \mathbf{w}) = -\sum_{i=1}^N \{y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)\}.$$

While a closed form solution to the MLE does not exist here due to the nonlinearity of the logistic sigmoid function, we can still optimize the parameters $\mathbf{w}$ using gradient descent.

Definition 3.13. Multi-class logistic regression with softmax
For squashing a vector of activations in probabilities, the multi-class generalization of the sigmoid function is the softmax:

$$\mathrm{softmax}_k(\mathbf{z}) = \frac{\exp(z_k)}{\sum_{i=1}^K \exp(z_i)}, \text{ for all } k.$$

Multi-class logistic regression applies softmax and assigns new data points to the class with the highest probability.

3.6.2 Probabilistic Generative Models

Definition 3.14. Probabilistic Generative Models
Here we model the joint distribution of the class y^* and the input data point $\mathbf{x}^*$ together as

$$p(y^*, \mathbf{x}^*).$$

In probabilistic generative modeling, we view that a data point is produced by first selecting a class y^* from a categorical class prior $p(y^*)$, and then generating the data point $\mathbf{x}^*$ from the class-conditional distribution $p(\mathbf{x}^* \mid y^*)$, which we must choose. In other words, we model the process by which the data was generated.

Note that the generative approach lets us exploit any domain knowledge we have about how the data was generated. It also lets us create more data by sampling from the joint distribution.

Example 3.15. Gaussian probabilistic generative modeling for classification
Given a new data point $\mathbf{x}^*$, we pick the class C_k that maximizes the conditional density

$$p(y^* = C_k \mid \mathbf{x}^*) \propto p(\mathbf{x}^* \mid y^* = C_k)\, p(y^* = C_k).$$

Our class prior distribution $p(y)$ is always categorical, so given some dataset $\mathbf{D}$, then the MLE of each class probability $p(y = C_k) = \pi_k$ is the proportion of the dataset belonging to C_k.

Now we must choose the shape of the class-conditional distribution $p(\mathbf{x} \mid y)$ of features. Suppose our class-conditional distributions are multivariate Gaussian with different means but shared covariance. To fit our model on $\mathbf{D}$, we must then calculate the MLE of these parameters $\boldsymbol{\mu}_k$ and $\mathbf{\Sigma}$. To do so, we optimize each parameter individually by differentiating the likelihood with respect to the parameter of interest and considering all others as constants. For the full derivation, see Homework 2 Problem 2 for the multi-class case (or Derivation 3.6.1 for the simplified Ebinary case).

Definition 3.16. Naive Bayes
In the Naive Bayes generative model, we make the simplification that our features are conditionally independent on class, i.e., each feature x_i of the data points $\mathbf{x}$ are independent given class $y = C_k$. Then the class-conditional distribution is

$$p\left(\mathbf{x} \mid y = C_k\right) = \prod_{i=1}^{D} p\left(x_i \mid y = C_k\right).$$

The model of $p\left(x_i \mid y = C_k\right)$ is left to our choice.

4 Neural Networks

4.1 Motivation

Neural networks simultaneously solve for our model parameters and the best basis transformations. They are universal function approximators, meaning that with a large enough network, it is possible to approximate any function. However, this flexibility also means that (1) NNs take a lot of computation to train due to the size of the effective model space, and (2) NNs can severely overfit if we are not careful.

As such, there is need to apply NNs if a problem can be solved effectively with simpler techniques. However, more complex problems make it too hard to engineer features for simple regression or classification techniques.

4.2 Neural Network Basics and Terminology

The feed-forward neural network is the most basic setup for a neural network.

4.3 Neural Network Basics and Terminology

Definition 4.1. Feed-forward network
To transform an input $\mathbf{x}$ into $\mathbf{y}$, we use a series of connected layers of nodes. The nodes in the hidden layers, those in between the input and output layers, correspond to activations.

Between each layer, there are connections, each with their own weight. The activation value at a given node is the weighted sum of the connected nodes preceding it (plus some extra bias term defined for that layer). That activation is then transformed by some non-linear function and passed forward. In doing so, the basis transformations made by the neural network are updated along as we update the weights.

Formally, consider a fully-connected neural network with M nodes in its first hidden layer. The activation at node j in the first hidden layer is

$$a_j^{(1)} = \sum_{d=1}^{D} w_{jd}^{(1)} x_d + w_{j0}^{(1)},$$

which is then transformed by some function h,

$$z_j^{(1)} = h\left(a_j\right),$$

and then passed forward. The activation at node j' of the second layer is

$$a_{j'}^{(2)} = \sum_{j=1}^{M} w_{j'm}^{(2)} z_j^{(1)} + w_{j'0}^{(2)},$$

and so on.

4.4 Network Training

Example 4.2. Common neural network loss functions
Our choice of objective will depend on the type of problem and the properties we desire. Some common choices are

- For linear regression, least squares loss:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(y\left(\mathbf{x}_n, \mathbf{w}\right) - y_n\right)^2,$$

  If the regression problem has multiple outputs, then the loss would sum over these different target values.

- For binary classification, produced by a single sigmoid output activation unit, then negated log-likelihood (or cross-entropy) is the typical loss function:

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left(y_n \ln \hat{y}_n + (1 - y_n)\left(\ln\left(1 - \hat{y}_n\right)\right)\right).$$

  For multiclass classification, produced by a softmax function in the output activation layer, we similarly have

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{kn} \ln\left(\frac{\exp\left(a_k(\mathbf{x}, \mathbf{w})\right)}{\sum_{j=1}^{K}\exp\left(a_j(\mathbf{x}, \mathbf{w})\right)}\right).$$

Definition 4.3. Backpropagation
To compute the gradient of our objective function with respect to our weights, we use the chain rule to backpropagate and pass errors backwards. In practice, libraries like PyTorch create a computational graph during the forward pass to efficiently backpropogate gradients.

4.5 Choosing a Network Structure

Remark 4.4. Addressing NN overfitting
Our input and output layer dimensions depend simply on our feature dimensions and the output dimensions required for our problem. The hidden layer dimensions are of our choice, but the more hidden layers and hidden nodes we have, the more we will overfit. We have a few methods to address this:

- **Cross validation:** we train our model with differing numbers of nodes and structures, use cross validation to choose the best model by selecting the model that performs best on the validation set.
- **Regularization:** we can introduce some regularization to our objective functions, such as a simple quadratic regularizer of form $\frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$.
- **Data augmentation:** we could increase the size and diversity of our training data by applying transformations to the initial data set (e.g., for an image dataset, change something like the brightness or density of images)
- Simply use a smaller network

4.6 Specialized Forms of Neural Networks

Simple neural networks, as universal function approximators, are useful for any task. However, we have developed specialized neural networks that better handle certain data types and use cases. Here we give a high level view of some forms.

Definition 4.5. Convolutional Neural Networks (CNNs)
CNNs are most often used for image data, though their underlying principles apply in other domains as well. CNNs extract smaller local features from images via a sliding window; intuitively, this is a matrix kernel that moves over every subsection of an image, producing a summary of those subsections that feed into the next layer in our network. We do this over the entire image, and with several different sliding windows. This (1) lets us summarize a feature of interest, such as some characteristic in an image, and (2) is location invariant, letting us identify features anywhere in an image.

Definition 4.6. Recurrent Neural Networks (RNNs)
RNNs add backward passing of activations into their network structure to improve predictions on data where there is some temporal dependence on what came previously. In this sense, our network is stateful because it's remembering what came before. This is helpful in NLP applications such as next word prediction, where the preceding words are crucial to predicting the next word.

Definition 4.7. Bayesian Neural Networks (BNNs)
Until now, our training process has used maximum likelihood/posterior estimation. BNNs introduce a distribution over the model parameters, which are marginalized in order to make predictions. The rationale for utilizing a BNN is the same as any Bayesian techniques in generaly, particularly the incorporation of prior information.

5 Support Vector Machines

5.1 Motivation

SVMs are useful for both classification and regression, and they are part of the family of margin methods. Appealingly, SVMs can be solved as convex optimization problems. Here we will discuss SVMs for classification.

Definition 5.1. Margin
Margin is the distance of the nearest data point from the separating hyperplane of an SVM model. Larger margins often lead to more generalizable models.

5.2 Hard Margin Classifier for Linearly Separable Data

Example 5.2. Hard margin SVM formulation for binary classificaion
First let us formulate the optimization problem under the hard margin constraint, i.e., under the assumption that our data is linearly separable. We classify our points into classes ± 1 depending on the sign of our discriminant function $h(\mathbf{x}^*)$, where

$$h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$

is some simply linear model. Our hyperplane is given by

$$\mathbf{w}^\top \mathbf{x} + w_0 = 0.$$

Note that $\mathbf{w}$ is orthogonal to this hyperplane, for if $\mathbf{x_1}, \mathbf{x_2}$ on our hyperplane, then

$$\mathbf{w}^\top (\mathbf{x_1} - \mathbf{x_2}) = -w_0 + w_0 = 0.$$

Thus, we may calculate the signed distance between any $\mathbf{x}$ and our hyperplane by the orthogonal decomposition

$$\mathbf{x} = \mathbf{x}_p + d\frac{\mathbf{w}}{\|\mathbf{w}\|_2},$$

from which we derive

$$\mathbf{w}^\top \mathbf{x} = \mathbf{w}^\top \mathbf{x}_p + d\frac{\mathbf{w}^\top \mathbf{w}}{\|\mathbf{w}\|_2} = -w_0 + d\|\mathbf{w}\|_2 \quad \Longrightarrow \quad d = \frac{\mathbf{w}^\top \mathbf{x} + w_0}{\|\mathbf{w}\|_2}.$$

To make the distance unsigned for a correctly classified point, we multiply by y_n. Then defining the margin for our data set to be the margin of the closest point, we wish to find the parameters $\mathbf{w}, w_0$ maximizing this least margin,

$$\max_{\mathbf{w}, w_0} \frac{1}{\|\mathbf{w}\|_2} \left[\min_n y_n \left(\mathbf{w}^\top \mathbf{x}_n + w_0 \right) \right].$$

Because scaling $\mathbf{w}, w_0$ by any constant α does not change our hyperplane, we may enforce the condition

$$y_n \left(\mathbf{w}^\top \mathbf{x}_n + w_0 \right) \geq 1, \forall n,$$

which simplifies our problem to

$$\max_{\mathbf{w},w_0} \frac{1}{\|\mathbf{w}\|_2} \quad \text{such that} \quad y_n\left(\mathbf{w}^\top \mathbf{x}_n + w_0\right) \geq 1 \; \forall n$$

or equivalently,

$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|_2^2 \quad \text{such that} \quad y_n\left(\mathbf{w}^\top \mathbf{x}_n + w_0\right) \geq 1 \; \forall n,$$

where we scale by 1/2 for convenience.

5.3 Soft Margin Classifier

Example 5.3. Soft margin SVM
The soft margin SVM allows for some data points to be within the margin boundary or on the incorrect side of the hyperplane. To relax the hard margin constraint, we introduce slack variables $\xi_n \geq 0$ given by

$$\xi_n = \begin{cases} = 0 & \text{if } \mathbf{x}_n \text{ is correctly classified,} \\ \in (0,1] & \text{if } \mathbf{x}_n \text{ is correctly classified but inside the margin region,} \\ > 1 & \text{if } \mathbf{x}_n \text{ is incorrectly classified.} \end{cases}$$

While we still penalize points on the wrong side of the margin boundary, it is no longer forbidden. This gives us the soft-margin training problem:

$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{n=1}^{N} \xi_n$$

such that

$$y_n\left(\mathbf{w}^\top \mathbf{x}_n + w_0\right) \geq 1 - \xi_n \; \forall n \qquad \text{and} \qquad \xi_n \geq 0 \; \forall n.$$

Here, the regularization parameter C determines how heavily we penalize violations of the hard margin constraints; large C yields less regularization and follows the data more closely (less misclassifications).

5.4 Conversion to Dual Form

Example 5.4. Hard margin dual formulation
Writing our problem objective and constraints as

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w},$$
$$g_n(\mathbf{w}, w_0) = -y_n\left(\mathbf{w}^\top \mathbf{x}_n + w_0\right) + 1 \leq 0 \; \forall n,$$

our problem becomes

$$\min_{\mathbf{w}} f(\mathbf{w}) \quad \text{s.t.} \quad g_n(\mathbf{w}, w_0) \leq 0 \; \forall n.$$

To solve this, we optimize the Lagrangian function

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = f(\mathbf{w}) + \sum_n \alpha_n g_n(\mathbf{w})$$

while introducing a subproblem on $\boldsymbol{\alpha}$ to encode the inequality form of our constraints,

$$\max_{\boldsymbol{\alpha}} L(\mathbf{w}, w_0, \boldsymbol{\alpha}) \quad \text{s.t.} \quad \alpha_n \geq 0 \; \forall n.$$

This encodes our hard margin constraint because if $\mathbf{w}$ violates any of our constraints, then the subproblem on $\boldsymbol{\alpha}$ becomes unbounded, with α_n on the corresponding constraints driven arbitrarily large. If all constraints are met, then $g_n(\mathbf{w}) < 0$ and thus $\alpha_n = 0$ for all n, yielding $\alpha_n g_n(\mathbf{w}) = 0$, as desired. Overall, we have the problem

$$\min_{\mathbf{w}} \left[\max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\mathbf{w}, w_0, \boldsymbol{\alpha}) \right] = \min_{\mathbf{w}, w_0} \left[\max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} \frac{1}{2} \mathbf{w}^\top \mathbf{w} - \sum_n \alpha_n \left(y_n \left(\mathbf{w}^\top \mathbf{x}_n + w_0 \right) - 1 \right) \right].$$

Because this problem meets sufficient conditions of strong duality, this is equivalent to

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} \left[\min_{\mathbf{w}, w_0} L\left(\mathbf{w}, \boldsymbol{\alpha}, w_0\right) \right].$$

To solve this, we begin with the inner minimization problem, solving for $\mathbf{w}$ and w_0 in terms of $\boldsymbol{\alpha}$,

$$\begin{aligned}
\nabla L\left(\mathbf{w}, \boldsymbol{\alpha}, w_0\right) = \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = 0 \quad &\implies \quad \mathbf{w}^* = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n, \\
\frac{\partial}{\partial w_0} L\left(\mathbf{w}, \boldsymbol{\alpha}, w_0\right) = -\sum_{n=1}^{N} \alpha_n y_n = 0 \quad &\implies \quad \sum_{n=1}^{N} \alpha_n y_n = 0.
\end{aligned}$$

Note that we do not actually get any value for w_0, but another constraint on $\boldsymbol{\alpha}$. Substituting $\mathbf{w}^*$, we get

$$\begin{aligned}
L\left(\mathbf{w}, \boldsymbol{\alpha}, w_0\right) &= \frac{1}{2} \mathbf{w}^\top \mathbf{w} - \mathbf{w}^\top \sum_n \alpha_n y_n \mathbf{x}_n - w_0 \sum_n \alpha_n y_n + \sum_n \alpha_n \\
&= -\frac{1}{2} \mathbf{w}^\top \mathbf{w} + \sum_n \alpha_n \\
&= \sum_n \alpha_n - \frac{1}{2} \left(\sum_n \alpha_n y_n \mathbf{x}_n \right)^\top \left(\sum_{n'} \alpha_{n'} y_{n'} \mathbf{x}_{n'} \right)
\end{aligned}$$

Now our problem is entirely in terms of $\boldsymbol{\alpha}$,

$$\max_{\boldsymbol{\alpha}} \left[\sum_n \alpha_n - \frac{1}{2} \sum_{n,n'} \alpha_n \alpha_{n'} y_n y_{n'} \mathbf{x}_n^\top \mathbf{x}_{n'} \right] \quad \text{such that} \quad \sum_n \alpha_n y_n = 0, \alpha_n \geq 0 \; \forall n,$$

i.e., we have a quadratic programming problem which we have techniques to solve.

Example 5.5. Making predictions with SVMs, support vectors
After solving our quadratic programming problem for the optimal multipliers $\boldsymbol{\alpha}$, most of the α_i will be zero. The data points corresponding to non-zero α_i are the *support vectors*, and they are the only data points informing our decision boundary.

- In the hard-margin formulation, our support vectors must be data points on the margin boundary.
- In the soft-margin formulation, they will be data points either on the margin boundary, inside the margins, or misclassified.

To make predictions, we substitute our expression for the optimal weights $\mathbf{w}^*$, yielding

$$h(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^\top \mathbf{x} + w_0.$$

To solve for w_0, we use any point $\mathbf{x}$ on the margin boundary, recognizing that we must have $y_n \left(\mathbf{w}^\top \mathbf{x}_n + w_0\right) = 1$. Choosing a point on the margin boundary is easy:

- For the hard-margin formulation, this is any example for which $\alpha_n > 0$.
- For the soft-margin formulation, it can be shown that points on the margin boundary are those with $C > \alpha_n > 0$.

Also note that support vectors have margin $1/\|\mathbf{w}\|_2$.

Example 5.6. Soft margin dual formulation
There is a similar dual form for the soft margin SVM training problem:

$$\max_{\boldsymbol{\alpha}} \left[\sum_n \alpha_n - \frac{1}{2} \sum_{n,n'} \alpha_n \alpha_{n'} y_n y_{n'} \mathbf{x}_n^\top \mathbf{x}_{n'} \right] \quad \text{such that} \quad \sum_n \alpha_n y_n = 0, C \geq \alpha_n \geq 0 \ \forall n.$$

Definition 5.7. Kernel trick
The kernel trick reduces the computational cost of high-dimensional basis transformations.

6 Clustering

6.1 Motivation

Clustering is our first foray into unsupervised techniques, whose goal is to uncover structure in a potentially non-labeled data set. Here we explore two common techniques: K-Means Clustering and Hierarchical Agglomerative Clustering.

In clustering specifically, our goal is to group data points that are similar. There are many reasons we might do this. For organizational purposes, it's convenient to have different classes of data. It can be easier for a human to sift through data if it's loosely categorized beforehand. It may be a preprocessing step for an inference method; for example, by creating additional features for a supervised technique. It can help identify which features make our data points most distinct from one another. It might even provide some idea of how many distinct data types we have in our set.

Of course, we must choose some metric of distance between data points $\mathbf{x}$ and $\mathbf{x}'$. For D-dimensional Euclidean space, we often use L2 distance,

$$\left\|\mathbf{x} - \mathbf{x}'\right\|_{L2} = \sqrt{\sum_{d=1}^{D} (\mathbf{x}_d - \mathbf{x}_d{}')^2},$$

though other metrics are required for other data types such as data with discrete features.

6.2 K-Means Clustering

First let us introduce the general prodecure of K-means and its usual optimization algorithm of Lloyd's Algorithms (using L2 distance).

Definition 6.1. K-means
Our goal is to assign data points to a fixed number of clusters, iteratively updating our cluster locations and assignments based on some distance metric. We proceed as follows:

1. Initialize cluster centers by randomly selecting points in our data set.
2. Using a distance metric of your choosing, assign each data point to the closest cluster.
3. Update the cluster centers based on your assignments and distance metric. (In the common case of L2 distance, we use Lloyd's algorithm, updating the centers by averaging the data points in each cluster.)
4. Repeat steps 1 and 2 until convergence.

Example 6.2. Lloyd's Algorithm
Lloyd's algorithm optimizes cluster assignments via coordinate descent. At each iteration, we define the loss

$$\mathcal{L}\left(\mathbf{X}, \{\boldsymbol{\mu}\}_{c=1}^{C}, \{\mathbf{r}\}_{n=1}^{N}\right) = \sum_{n=1}^{N}\sum_{c=1}^{C} r_{nc} \left\|\mathbf{x}_n - \boldsymbol{\mu}_c\right\|_2^2,$$

where data $\mathbf{X}$ is $N \times D$, cluster centers $\{\boldsymbol{\mu}\}_{c=1}^{C}$ is $C \times D$, and responsibility vector matrix $\{\mathbf{r}\}_{n=1}^{N}$ is $N \times C$. The responsibility vectors are one-hot encoded vectors denoting the cluster to which we assign each point, and of course we initially choose our responsibility vectors to minimize each data point's distance from its cluster center:

$$r_{nc} = \begin{cases} 1 & \text{if } c = \arg\min_{c'} \|\mathbf{x}_n - \boldsymbol{\mu}_{c'}\| \\ 0 & \text{otherwise} \end{cases}.$$

We wish to minimize our loss by updating our cluster centers $\boldsymbol{\mu}_c$, so we compute

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_c} = -2 \sum_{n=1}^{N} r_{nc} (\mathbf{x}_n - \boldsymbol{\mu}_c) \qquad \Longrightarrow \qquad \boldsymbol{\mu}_c = \frac{\sum_{n=1}^{N} r_{nc}\mathbf{x}_n}{\sum_{n=1}^{N} r_{nc}}.$$

Intuitively, this is the average of all data points assigned to the cluster center $\boldsymbol{\mu}_c$.We iteratively update our responsibility vectors and cluster centers until we have converged to a stable set of cluster centers and responsibility vectors.

Lloyd's is guaranteed to converge only to a locally optimal solution. Finding the globally optimal assignments is NP-hard, so we generally execute Lloyd's several times with different random initializations of cluster centers, and simply select the best assignment. Also note that we must standardize our data in a preprocessing step to avoid scale mismatch between features, which generally means standardizing each feature to mean 0 and standard deviation 1.

Now let us make some remarks on how to determine the proper number of clusters and some modified K-means procedures.

Remark 6.3. Number of Clusters
Loss strictly decreases with more clusters, but certainly a tradeoff, for having either a single cluster or N clusters is obviously useless. To determine a good number of clusters, we generally perform K-Means with a varying C, plot the number of clusters against the loss, and choose C at the 'knee' of our plot, where there is a slight bend in the curve and the loss begins decreasing more slowly.

Remark 6.4. K-Means++
K-Means++ assumes our cluster centers will typically be spread out when we've reached convergence, and so presents a better initialization algorithm. We choose the first cluster center by randomly selecting a data point, and then for all subsequent cluster centers, we select points in our data set with probability proportional to the squared distance from their nearest cluster center.

Remark 6.5. K-Medoids
Previously, we computed loss by averaging the contributions of each data point. In some cases like categorical features, this averaging step doesn't make sense, so we use K-Medoids and instead update the new cluster center to be the data point assigned to that cluster which is most like the others.

6.3 Hierarchical Agglomerative Clustering

Definition 6.6. Hierarchical agglomerative clustering
In HAC, we group data from the bottom up instead of assigning it to a fixed number of clusters. Each data point begins as its own cluster, and then we iteratively merge clusters together using some distance metric. The iterative merging also constructs a relationship tree over our data set, called dendrograms, whose leaves are individual data points and whose trunk is the trivial cluster containing the entire data set. Overall, our steps are

1. Start with N clusters for each data point.
2. Measure the distance between clusters using some inter-cluster distance metric.
3. Merge the two closest clusters together, recording the distance between these two merged clusters through the height of the dendrogram where they merge.
4. Iterate step 2 until we're left with only a single cluster.

The dendrogram provides layers of clustering; if we desire clusters which are k units apart, we may simply examine all the clusters that exist below that cut point on the dendrogram.

Example 6.7. Linkage criterion
Our linkage criteria are methods to define inter-cluster distance $d_{C,C'}$ between a cluster pair C and C'. Note that after defining a linkage criterion, the merging of HAC is deterministic.

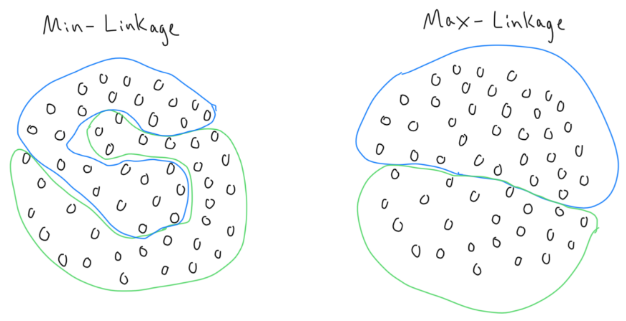
- **Min-Linkage Criteria:** tends to produce 'stringy' clusters because we are most inclined to extend existing clusters by grabbing whichever data points are closest,

$$d_{C,C'} = \min_{k,k'} \|\mathbf{x}_k - \mathbf{x}_{k'}\|$$

  where $\mathbf{x}_k$ are data points in cluster C and $\mathbf{x}_{k'}$ are data points in cluster C'.

- **Max-Linkage Criterion:** tends to produce more compact clusters,

$$d_{C,C'} = \max_{k,k'} ||\mathbf{x}_k - \mathbf{x}_{k'}||$$



- **Average-Linkage Criterion:** we average the pairwise distance between each point in each cluster,

$$d_{C,C'} = \frac{1}{KK'} \sum_{k=1}^{K} \sum_{k'=1}^{K'} \|\mathbf{x}_k - \mathbf{x}_{k'}\|$$

- **Centroid-Linkage Criterion:** use the distance between the centroid of each cluster,

$$d_{C,C'} = \left\| \frac{1}{K} \sum_{k=1}^{K} \mathbf{x}_k - \frac{1}{K'} \sum_{k'=1}^{K'} \mathbf{x}_{k'} \right\|$$

Remark 6.8. K-Means vs HAC
We discuss some differences in these clustering techniques:

- There is HAC is deterministic while K-Means incurs randomness and so needs to be run multiple times to ensure good results.
- In K-means, we must specify the number of clusters up front, using something like the knee-method to decide on the number of clusters. On the other hand, HAC does not require fixing the number of clusters, though it is later necessary to decide a max inter-cluster distance to present final clustering results.
- HAC provides more interactivity than K-Means by letting us choose arbitrary cut points and visualizing a relationship tree in the dendrogram itself.

7 Dimensionality Reduction

Previously, we made built more expressive models in supervised learning tasks by projecting our data into higher dimensions using basis functions. Here, we instead focus on reducing the dimensionality of our data (in an unsupervised way) through PCA. PCA is useful for visualization purposes, removing redundant information, or making a data set more computationally manageable. PCA is a good tool for data exploration, particularly when digging into an unfamiliar data set for the first time.

7.1 Motivation

7.2 Applications

7.3 Principal Component Analysis