Machine Learning

Muchang Bahng

Spring 2024

Contents

Machine learning in the 1980s have been focused on developing a rigorous theory of learning algorithms, and the field has been dominated by statisticians. They strived to develop the theoretical foundation of algorithms that can be implemented and applied to real-world data. With the advent of deep learning, the theory behind state-of-the-art algorithms, mostly black-box models, has slowed down, but their applications have exploded. It is now a field of trying out a bunch of things and sticking to what works.

These set of notes focuses on trying to provide the theoretical foundations of classical, interpretable machine learning algorithms, while my deep learning notes skim over the theory to focus on model architectures and applications (which is plenty to write about on its own). I've spent a good amount of time trying to create a map of machine learning, but after rewriting these notes multiple times. I've come to the conclusion that it is impossible to create a nice chronological timeline. Like math, you keep on revisiting the same topics over and over again, but at a higher level, and it's not as simple to organize everything into, say parametric vs nonparametric[1], supervised vs unsupervised[2], or discriminative vs generative models.[3] Therefore, I've settled (for now) on the following structure. Before you begin, the direct prerequisites are my notes in probability theory, functional analysis, frequenist and Bayesian statistics, and information theory. If you are new to machine learning, go over my notes on Stanford CS229, which simply covers basic algorithms and their implementation.

First, we will establish the theoretical foundations of ML by introducing the most abstract and general formulation of learning: statistical learning theory. In here, we will talk about function classes, what it means for a model to *learn*, what *overfitting* means in a mathematical sense, and some basic probability theorems that give us enough confidence that we can achieve good theoretical results.

Then we move on to supervised learning, which provides review from statistics and gives us ideas to build upon when talking about unsupervised learning.

1. Just like in every other class, we begin with low-dimensional linear regression, introducing it both in the frequentist and Bayesian perspectives. We expand a bit into modified versions of these models, through weighted least squares and different loss functions.
2. Then we consider the need for regularization in high-dimensional linear regression, where the covariates may be much lager than the samples. This leads us to introduce both lasso and ridge as convex approximations of forward stepwise regression. We interpret them in both the frequentist and Bayesian ways.
3. Now that we have covered linear regression, we can move onto *nonparametric regression.*
4. We complete regression and move to *linear classification*, with logistic/softmax regression, SVMs, and DLA, followed by *nonparametric classification*, such as decision trees.
5. It turns out that all these models are a specific instance of a more general model called *generalized linear models (GLMs).* We develop the theory behind this model in general.
6. We conclude supervised learning by talking about *ensemble methods*, where we can take any combination of the models that we have learned so far to create new models that may have better performance.

Now we move onto unsupervised learning, which consists of every other problem dealing with a dataset without labels. The simplest algorithms are direct-clustering (K-means), direct density estimation (kernel density estimation), or direct dimensionality-reduction (PCA, UMAP). I call them *direct* because they interact with the data directly, as opposed to *latent variable* models that attempts to model the data by adding hidden random variables.

1. Direct clustering and density estimation models cover most algorithms you would learn in a introduction to machine learning course. Besides simple clustering models and kernel density estimation, there is

[1]K nearest neighbors is a nonparameteric model given that the data is not fixed. When the data is fixed, then our function search space is finite.

[2]There are semi-supervised or weakly supervised models, and models like autoencoders use a supervised algorithm without any labels.

[3]Using Bayes rule, we can always reduce generative models into discriminative models.

not a lot to learn here. However, I do focus on the high-dimensional case where we may need some tricks for computational efficiency.

2. Direct dimensionality reduction models tend to have more variants, and I go through the most widely used algorithms.
3. Now we start getting into latent variable models. Just as linear regression is the simplest supervised regression model, linear factor models are the unsupervised analogue, which attempts to model the latent and the visible variables through a linear relationship.
4. We expand on this into nonlinear latent variable models, such as latent variables living in a discrete space. The most popular is Gaussian mixture models, which uses a multinomial latent variable representing which "cluster" a sample lives in.
5. Finally we talk about a generalization of these models by modeling probability distributions with a *graph*, or a *Bayesian network*. This allows for high-dimensional distributions to be represented with much fewer parameters.

At this point we are done, and all that is left is to talk about some meta-training techniques, such as cross validation, information criterion, and practical methods for real-world problems.

For clarification, $\mathcal{D}$ can represent different things depending on the problem:

1. In a density estimation problem, where we have a single dataset $\mathbf{X}$, $\mathcal{D} = \mathbf{X}$ since this data tells us information about which distribution it could come from.
2. In a regression problem, $\mathcal{D} = \mathbf{Y}$, that is, $\mathcal{D}$ will always be the output data, not the input data $\mathbf{X}$. We can think of the input data $\mathbf{X}$ as always being fixed, and it is upon observation of the *outputs* $\mathbf{Y}$ on these inputs that gives us information.

In both the frequentist and Bayesian settings, the likelihood $p(\mathcal{D} \mid \mathbf{w})$ plays a central role. In the frequentist setting, the process is divided into two steps:

1. We optimize $\mathbf{w}$ with some **estimator**, with a popular one being the **maximum likelihood estimator**. A popular estimator is **maximum likelihood**, which seeks to maximize $p(\mathcal{D} \mid \mathbf{w})$ w.r.t. $\mathbf{w}$.
2. We optimize $\mathbf{w}$ with some **estimator**, with a popular one being the **maximum likelihood estimator**. A popular estimator is **maximum likelihood**, which seeks to maximize $p(\mathcal{D} \mid \mathbf{w})$ w.r.t. $\mathbf{w}$.
3. We fix the optimized $\mathbf{w}^*$ and error bars on this estimate are obtained by considering the distribution of possible datasets $\mathcal{D}$. One approach is **bootstrapping**, which goes as follows. Given our original dataset $\mathbf{X} = \{x^{(1)}, \ldots, x^{(N)}\}$, we can create a new dataset $\mathbf{X}'$ by sampling N points at random from $\mathbf{X}$, with replacement, so that some points in $\mathbf{X}$ may be replicated in $\mathbf{X}'$, whereas other points in $\mathbf{X}$ may be absent in $\mathbf{X}'$. This process is repeated L times to generate L different datasets. Then, we can look at the variability of prediction between the different bootstrap data sets.

In a Bayesian setting, there is only a single dataset $\mathcal{D}$ and the uncertainity in the parameters is expressed through a probability distribution over $\mathbf{w}$. It also includes prior knowledge naturally in the form of prior distributions.

1 Low Dimensional Linear Regression

In introductory courses, we start with linear predictors since it is easy to understand. We still start with linear predictors because in high-dimensional machine learning, even linear prediction can be hard as we will see. Low dimensional linear regression is what statisticians worked in back in the early days, where data was generally low dimensional.[4] Generally, we had $d < n$, but these days, we are in the regime where $d > n$. For example, in genetic data, you could have a sample of $n = 100$ people but each of them have genetic sequences at $d = 10^6$. When the dimensions become high, the original methods of linear regression tend to break down, which is why I separate low and high dimensional linear regression. The line tends to be fuzzy between these two regimes, but we will not worry about strictly defining that now.

In here, we start with **multiple linear regression**, which assumes that we have several covariates and one response. If we extend this to multiple responses (i.e. a response vector), this is called **multivariate linear regression**. The simple case for one response is called **simple linear regression**, and we will mention some nice formulas and intuition that come out from working with this.

Definition 1.1 (Linear Regression Model)

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$, where $x^{(i)} \in \mathbb{R}^d$ with $x_1 = 1$ (which is what we do in practice to include an intercept term) and $y^{(i)} \in \mathbb{R}$, the multiple linear regression model is

$$y = \beta^T x + \epsilon \tag{1}$$

with the following assumptions:

1. *Weak exogeneity*: the covariates are observed without error.
2. *Linearity*: the mean of the variate is a linear combination of the parameters and the covariates.
3. *Gaussian errors*: the errors are Gaussian.[a]
4. *Homoscedasticity*: the errors (the observations of Y) have constant variance.
5. *Independence of errors*: The errors are uncorrelated.
6. *No multicollinearity*: more properly, the lack of perfect multicollinearity. Assume that the covariates aren't perfectly correlated.[b]

[a]We can relax this assumption when we get into generalized linear models, and in most cases we assume some closed form of the error for computational convenience, like when computing the maximum likelihood.

[b]This is the assumption that breaks down in high dimensional linear regression.

In order to check multicollinearity, we compute the correlation matrix.

Definition 1.2 (Correlation Matrix)

The correlation matrix of random variables $X_1, \ldots, X_d$ is

$$\mathbf{C}_{ij} = \mathrm{Corr}(X_i, X_j) = \frac{\mathrm{Cov}(X_i, X_j)}{\sigma_{X_i}\sigma_{X_j}}$$

given that $\sigma_{X_i}\sigma_{X_j} > 0$. Clearly, the diagonal entries are 1, but if there are entries that are very close to 1, then we have multicollinearity.

Assume that two variables are perfectly correlated. Then, there would be pairs of parameters that are indistinguishable if moved in a certain linear combination. This means that the variance of $\hat{\boldsymbol{\beta}}$ will be very ill conditioned, and you would get a huge standard error in some direction of the β_i's. We can fix this by making sure that the data is not redundant and manually removing them, standardizing the variables, making a change of basis to remove the correlation, or just leaving the model as it is.

If these assumptions don't hold,

[4]Quoting Larry Wasserman, even 5 dimensions was considered high and 10 was considered massive.

1. *Weak exogeneity*: the sensitivity of the model can be tested to the assumption of weak exogeneity by doing bootstrap sampling for the covariates and seeing how the sampling affects the parameter estimates. Covariates measured with error used to be a difficult problem to solve, as they required errors-in-variables models, which have very complicated likelihoods. In addition, there is no universal fitting library to deal with these. But nowadays, with the availability of Markov Chain Monte Carlo (MCMC) estimation through probabilistic programming languages, it is a lot easier to deal with these using Bayesian hierarchical models (or multilevel models, or Bayesian graphical models—these have many names).

2. *Linearity*: the linear regression model only assumes linearity in the parameters, not the covariates. Therefore you could build a regression using non-linear transformations of the covariates, for instance,

$$Y = X_1\beta_1 + X_1^2\beta_2 + \log(X_1)\beta_3 \tag{2}$$

   If you need to further relax the assumption, you are better off using non-linear modelling.

3. *Constant variance*: the simplest fix is to do a variance-stabilising transformation on the data. Assuming a constant coefficient of variation rather than a constant mean could also work. Some estimation libraries (such as the `glm` package in R) allow specifying the variance as a function of the mean.

4. *Independence of errors*: this is dangerous because in the financial world things are usually highly correlated in times of crisis. The most important thing is to understand how risky this assumption is for your setting. If necessary, add a correlation structure to your model, or do a multivariate regression. Both of these require significant resources to estimate parameters, not only in terms of computational power but also in the amount of data required.

5. *No multicollinearity*: If the covariates are correlated, they can still be used in the regression, but numerical problems might occur depending on how the fitting algorithms invert the matrices involved. The t-tests that the regression produces can no longer be trusted. All the covariates must be included regardless of what their significance tests say. A big problem with multicollinearity, however, is overfitting. Depending on how bad the situation is, the parameter values might have huge uncertainties around them, and if you fit the model using new data their values might change significantly.[5] Multicollinearity is a favourite topic of discussion for quant interviewers, and they usually have strong opinions about how it should be handled. The model's intended use will determine how sensitive it is to ignoring the error distribution. In many cases, fitting a line using least-squares estimation is equivalent to assuming errors have a normal distribution. If the real distribution has heavier tails, like the t-distribution, how risky will it make decisions based on your outputs? One way to address this is to use a technique like robust-regression. Another way is to think about the dynamics behind the problem and which distribution would be best suited to model them—as opposed to just fitting a curve through a set of points.

1.1 Ordinary Least Squares

If we use a squared loss function, this is called **ordinary least squares**. It is a well known fact that the true regressor that minimizes this loss is

$$f^*(x) = \mathbb{E}[Y \mid X = x] \tag{3}$$

which is the conditional expectation of Y given X. This is the true regressor function, which is the best approximation of Y over the σ-algebra generated by X. This may or may not be linear.

[5] I suggest reading this Wikipedia article on multicollinearity, as it contains useful information: `https://en.wikipedia.org/wiki/Multicollinearity`

Theorem 1.1 (Least Squares Solution For Linear Regression)

Given the design matrix $\mathbf{X}$, we can present the linear model in vectorized form:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2 \mathbf{I}) \tag{4}$$

The solution that minimizes the squared loss is

$$\begin{aligned} \boldsymbol{\beta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \in \mathbb{R}^d \\ \mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \hat{\sigma}^2(\mathbf{X}^T\mathbf{X})^{-1} \in \mathbb{R}^{d\times d} \end{aligned}$$

Proof.

The errors can be written as $\boldsymbol{\epsilon} = \mathbf{Y} - \mathbf{X}\boldsymbol{\beta}$, and you have the following total sum of squared errors:

$$S(\boldsymbol{\beta}) = \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})$$

We want to find the value of $\boldsymbol{\beta}$ that minimizes the sum of squared errors. In order to do this, remember the following matrix derivative rules when differentiating with respect to vector $\mathbf{x}$.

1. $\mathbf{x}^T\mathbf{A} \mapsto \mathbf{A}$
2. $\mathbf{x}^T\mathbf{A}\mathbf{x} \mapsto 2\mathbf{A}\mathbf{x}$

Now this should be easy.

$$\begin{aligned} S(\boldsymbol{\beta}) &= \mathbf{Y}^T\mathbf{Y} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\ &= \mathbf{Y}^T\mathbf{Y} - 2\mathbf{Y}^T\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\ \frac{\partial}{\partial\boldsymbol{\beta}}S(\boldsymbol{\beta}) &= -2\mathbf{X}^T\mathbf{Y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \end{aligned}$$

and setting it to $\mathbf{0}$ gives

$$2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{Y} = 0 \implies \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{Y}$$

and the variance of $\boldsymbol{\beta}$, by using the fact that $\mathrm{Var}[\mathbf{A}\mathbf{X}] = \mathbf{A}\,\mathrm{Var}[X]\mathbf{A}^T$, is

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\ \sigma^2\mathbf{I}\ \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}(\mathbf{X}'\mathbf{X})(\mathbf{X}'\mathbf{X})^{-1} = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}$$

But we don't know the true σ^2, so we estimate it with $\hat{\sigma}^2$ by taking the variance of the residuals. Therefore, we have

$$\begin{aligned} \boldsymbol{\beta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \in \mathbb{R}^d \\ \mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \hat{\sigma}^2(\mathbf{X}^T\mathbf{X})^{-1} \in \mathbb{R}^{d\times d} \end{aligned}$$

Example 1.1 (Copying Data)

What happens if you copy your data in OLS? In this case, our MLE estimate becomes

$$\begin{aligned} &\left(\begin{pmatrix} X \\ X \end{pmatrix}^T \begin{pmatrix} X \\ X \end{pmatrix}\right)^{-1} \begin{pmatrix} X \\ X \end{pmatrix}^T \begin{pmatrix} Y \\ Y \end{pmatrix} = \\ &\qquad = (X^TX + X^TX)^{-1}(X^TY + X^TY) = (2X^TX)^{-1}2X^TY = \hat{\beta} \end{aligned}$$

and our estimate is unaffected. However, the variance shrinks by a factor of 2 to

$$\frac{\sigma^2}{2}(\mathbf{X}^T\mathbf{X})^{-1} \tag{5}$$

A consequence of that is that confidence intervals will shrink with a factor of $1/\sqrt{2}$. The reason is that we have calculated as if we still had iid data, which is untrue. The pair of doubled values are obviously dependent and have a correlation of 1.

Another way to solve the solution is through likelihood estimation.

Theorem 1.2 (Maximum Likelihood Estimation of Linear Regression)

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, our likelihood is

$$L(\theta; \mathcal{D}) = \prod_{i=1}^N p(y^{(i)} \mid x^{(i)}; \theta) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \right)$$

We can take its negative log, remove additive constants, and scale accordingly to get

$$\begin{aligned} \ell(\theta) &= -\frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi) + \frac{1}{2\sigma^2} \sum_{i=1}^N \big(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\big)^2 \\ &= \frac{1}{2} \sum_{i=1}^N \big(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\big)^2 \end{aligned}$$

which then corresponds to minimizing the sum of squares error function.

Theorem 1.3 (Gradient Descent for Linear Regression)

Taking the gradient of this log likelihood w.r.t. θ gives

$$\nabla_\theta \ell(\theta) = \sum_{i=1}^N (y^{(i)} - \theta^T x^{(i)}) x^{(i)}$$

and running gradient descent over a minibatch $M \subset \mathcal{D}$ gives

$$\begin{aligned} \theta &= \theta - \eta \nabla_\theta \ell(\theta) \\ &= \theta - \eta \sum_{(x,y) \in M} (y - \theta^T x) x \end{aligned}$$

This is guaranteed to converge since $\ell(\theta)$, as the sum of convex functions, is also convex.
Note that since we can solve this in closed form, by setting the gradient to 0, we have

$$0 = \sum_{n=1}^N y^{(n)} \boldsymbol{\phi}(\mathbf{x}^{(n)})^T - \mathbf{w}^T \left(\sum_{n=1}^N \boldsymbol{\phi}(\mathbf{x}^{(n)}) \boldsymbol{\phi}(\mathbf{x}^{(n)})^T \right)$$

which is equivalent to solving the least squares equation

$$\mathbf{w}_{ML} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{Y}$$

Note that if we write out the bias term out explicitly, we can see that it just accounts for the translation (difference) between the average of the outputs $\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n$ and the average of the basis functions $\bar{\phi_j} = \frac{1}{N} \sum_{n=1}^N \phi_j(\mathbf{x}^{(n)})$.

$$w_0 = \bar{y} - \sum_{j=1}^{M-1} w_j \bar{\phi_j}$$

We can also maximize the log likelihood w.r.t. σ^2, which gives the MLE

$$\sigma^2_{ML} = \frac{1}{N} \sum_{n=1}^N \big(y^{(n)} - \mathbf{w}^T_{ML} \phi(\mathbf{x}^{(n)})\big)^2$$

Code 1.1 (MWE for OLS Linear Regression in scikit-learn)

Here is a minimal working example of performing linear regression with scikit-learn. Note that the input data must be of shape (n, d).

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
y = np.dot(X, np.array([1, 2])) + 3

model = LinearRegression()
model.fit(X, y)
print(X)
print(y)
print(model.score(X, y))
print(model.intercept_)
print(model.coef_)
print(model.predict(np.array([[3, 5]])))
```

```
[[1 1]
 [1 2]
 [2 2]
 [2 3]]
[ 6  8  9 11]
1.0
3.0000000000000018
[1. 2.]
[16.]
.
.
.
.
.
```

1.1.1 Bias Variance Decomposition

We can use what we have learned to prove a very useful result for the mean squared loss.

Theorem 1.4 (Pythagorean's Theorem)

The expected square loss over the joint measure $\mathbb{P}_{X\times Y}$ can be decomposed as

$$\mathbb{E}_{X\times Y}[(Y - h(X))^2] = \mathbb{E}_{X\times Y}[\big(Y - \mathbb{E}[Y \mid X]\big)^2] + \mathbb{E}_X[\big(\mathbb{E}[Y \mid X] - h(X)\big)^2] \tag{6}$$

That is, the squared loss decomposes into the squared loss of $\mathbb{E}[Y \mid X]$ and $g(X)$, which is the intrinsic misspecification of the model, plus the squared difference of Y with its best approximation $\mathbb{E}[Y \mid X]$, which is the intrinsic noise inherent in Y beyond the σ-algebra of X.

Proof.

We can write

$$\begin{aligned}
\mathbb{E}_{X\times Y}[L] &= \mathbb{E}_{X\times Y}\big[\big(Y - g(X)\big)^2\big] \\
&= \mathbb{E}_{X\times Y}\big[\big((Y - \mathbb{E}[Y \mid X]) + (\mathbb{E}[Y \mid X] - g(X))\big)^2\big] \\
&= \mathbb{E}_{X\times Y}[\big(Y - \mathbb{E}[Y \mid X]\big)^2] + \mathbb{E}_{X\times Y}[\{Y - \mathbb{E}[Y \mid X]\}\{\mathbb{E}[Y \mid X] - g(X)\}] \\
&\qquad + \mathbb{E}_X[\big(\mathbb{E}[Y \mid X] - g(X)\big)^2] \\
&= \mathbb{E}_{X\times Y}[\big(Y - \mathbb{E}[Y \mid X]\big)^2] + \mathbb{E}_X[\big(\mathbb{E}[Y \mid X] - g(X)\big)^2]
\end{aligned}$$

where the middle term cancels out due to the tower property.

We also proved a second fact: Since $\mathbb{E}[\big(\mathbb{E}[Y \mid X] - g(X)\big)^2]$ is the misspecification of the model, we cannot change this (positive) constant, so $\mathbb{E}\big[\big(Y - g(X)\big)^2\big] \geq \mathbb{E}[(Y - \mathbb{E}[Y \mid X])^2]$, with equality achieved when we perfectly fit g as $\mathbb{E}[Y \mid X]$ (i.e. the model is well-specified). Therefore, denoting $\mathcal{F}$ as the set of all $\sigma(X)$-measurable functions, then the minimum of the loss is attained when

$$\operatorname*{argmin}_{g \in \mathcal{F}} \mathbb{E}[L] = \operatorname*{argmin}_{g \in \mathcal{F}} \mathbb{E}\big[\big(Y - g(X)\big)^2\big] = \mathbb{E}[Y \mid X] \tag{7}$$

Even though this example is specific for the mean squared loss, this same decomposition, along with the bias variance decomposition, exists for other losses. It just happens so that the derivations are simple for the MSE, which is why this is introduced first. However, the derivations for other losses are much more messy, and sometimes may not hold rigorously. However, the general intuition that more complex models tend to overfit still hold true.

Now if we approximate $\mathbb{E}[Y \mid X]$ with our parameterized hypothesis $h_{\boldsymbol{\theta}}$, then from a Bayesian perspective the uncertainty in our model is expressed through a poseterior distribution over $\boldsymbol{\theta}$. A frequentist treatment, however, involves making a point estimate of $\boldsymbol{\theta}$ based on the dataset $\mathcal{D}$ and tries instead to interpret the uncertainty of this estimate through the following thought experiment: Suppose we had a large number of datasets each of size N and each drawn independently from the joint distribution $X \times Y$. For any given dataset $\mathcal{D}$, we can run our learning algorithm and obtain our best fit function $h^*_{\boldsymbol{\theta};\mathcal{D}}(\mathbf{x})$. Different datasets from the ensemble will give different functions and consequently different values of the squared loss. The performance of a particular learning algorithm is then assessed by taking the average over this ensemble of datasets, which we define $\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(\mathbf{x})] = \mathbb{E}_{(X \times Y)^N}[h_{\boldsymbol{\theta};\mathcal{D}}(\mathbf{x})]$. We are really taking an expectation over all datasets, meaning that the N points in each $\mathcal{D}$ must be sampled from $(X \times Y)^N$.

Consider the term $\big(\mathbb{E}[Y \mid X] - h_{\boldsymbol{\theta}}(X)\big)^2$ above, which models the discrepancy in our optimized hypothesis and the best approximation. Now, over all datasets $\mathcal{D}$, there will be a function $h_{\boldsymbol{\theta};\mathcal{D}}$, and averaged over all datasets $\mathcal{D}$ is $\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}]$. So, the random variable below (of $\mathcal{D}$ and X) representing the stochastic difference between our optimized function $h_{\boldsymbol{\theta};\mathcal{D}}(X)$ and our best approximation $\mathbb{E}[Y \mid X]$ can be decomposed into

$$\begin{aligned}
\big(\mathbb{E}[Y \mid X] - h_{\boldsymbol{\theta}:\mathcal{D}}(X)\big)^2 &= \big[\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]\big) + \big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta}:\mathcal{D}}(X)\big)\big]^2 \\
&= \big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]\big)^2 + \big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta}:\mathcal{D}}(X)\big)^2 \\
&\quad + 2\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]\big)\big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta}:\mathcal{D}}(X)\big) \\
&= \big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]\big)^2 + \big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta}:\mathcal{D}}(X)\big)^2
\end{aligned}$$

Averaging over all datasets $\mathcal{D}$ causes the middle term to vanish and gives us the expected squared difference between the two random variables, now of X.

Theorem 1.5 (Bias Variance Decomposition)

Therefore, we can write out the expected square difference between $h_{\boldsymbol{\theta}}$ and $\mathbb{E}[Y \mid X]$ as the sum of two terms.

$$\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}[Y \mid X] - h_{\boldsymbol{\theta}}(X)\big)^2\big] = \underbrace{\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]\big)^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta};\mathcal{D}}(X)\big)^2\big]}_{\text{variance}} \tag{8}$$

Let us observe what these terms mean:

1. The **bias** $\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]$ is a random variable of X that measures the difference in how the average prediction of our hypothesis function $\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]$ differs from the actual prediction $\mathbb{E}[Y \mid X]$.
2. The **variance** $\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta};\mathcal{D}}(X)\big)^2\big]$ is a random variable of X that measures the variability of each hypothesis function $h_{\boldsymbol{\theta}}(X)$ about its mean over the ensemble $\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]$.

Therefore, we can substitute this back into our Pythagoras decomposition, where we must now take the expected bias and the expected variance. Therefore, we get

$$\text{Expected Loss} = (\text{Expected Bias})^2 + \text{Expected Variance} + \text{Noise} \tag{9}$$

where

$$\begin{aligned}
(\text{Bias})^2 &= \mathbb{E}_X\left[\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)]\big)^2\right] \\
\text{Variance} &= \mathbb{E}_X\left[\mathbb{E}_{\mathcal{D}}\left[\big(\mathbb{E}_{\mathcal{D}}[h_{\boldsymbol{\theta};\mathcal{D}}(X)] - h_{\boldsymbol{\theta};\mathcal{D}}(X)\big)^2\right]\right] \\
\text{Noise} &= \mathbb{E}_{X\times Y}\left[\big(Y - \mathbb{E}[Y \mid X]\big)^2\right]
\end{aligned}$$

1.1.2 Convergence Bounds

Let's get a deeper understanding on linear regression by examining the convergence of the empirical risk minimizer to the true risk minimizer. We can develop a naive bound using basic concentration of measure.

Theorem 1.6 (Exponential Bound)

Let $\mathcal{P}$ be the set of all distributions for $X \times Y$ supported on a compact set. There exists constants c_1, c_2 s.t. that the following is true. For any $\epsilon > 0$,

$$\sup_{\mathbb{P}\in\mathcal{P}} \mathbb{P}^n\big(r(\hat{\beta}_n) > r(\beta_*(\mathbb{P}) + 2\epsilon)\big) \leq c_1 e^{-nc_2\epsilon^2} \tag{10}$$

Hence

$$r(\hat{\beta}_n) - r(\beta_*) = O_{\mathbb{P}}\left(\sqrt{\frac{1}{n}}\right) \tag{11}$$

Proof.

However, this is not a very tight bound, and we can do better. Though the proof is very long and will be omitted.

Theorem 1.7 (Gyorfi, Kohler, Krzyzak, Walk, 2002 [GKKW02])

Let $\sigma^2 = \sup_x \text{Var}[Y \mid X = x] < \infty$. Assume that all random variables are bounded by $L < \infty$. Then

$$\mathbb{E}\int |\hat{\beta}^T x - m(x)|^2 \, d\mathbb{P}(x) \leq 8 \inf_{\beta} \int |\beta^T x - m(x)|^2 \, d\mathbb{P}(x) + \frac{Cd(\log(n)+1)}{n} \tag{12}$$

You can see that the bound contains a term of the form

$$\frac{d\log(n)}{n} \tag{13}$$

and under the low dimensional case, d is small and bound is good. However, as d becomes large, then we don't have as good of theoretical guarantees.

Theorem 1.8 (Central Limit Theorem of OLS)

We have

$$\sqrt{n}(\hat{\beta} - \beta) \xrightarrow{d} N(0, \Gamma) \tag{14}$$

where

$$\Gamma = \Sigma^{-1}\mathbb{E}\big[(Y - X^T\beta)^2 XX^T\big]\Sigma^{-1} \tag{15}$$

The covariance matrix Γ can be consistently estimated by

$$\hat{\Gamma} = \hat{\Sigma}^{-1} \hat{M} \hat{\Sigma}^{-1} \tag{16}$$

where

$$\hat{M}(j,k) = \frac{1}{n} \sum_{i=1}^n X_i(j) X_i(k) \hat{\epsilon}_i^2 \tag{17}$$

and $\hat{\epsilon}_i = Y_i - \hat{\beta}^T X_i$.

1.2 Simple Linear Regression

The simple linear regression is the special case of the linear regression with only one covariate.[6]

$$y = \alpha + x\beta \tag{18}$$

which is just a straight line fit. Interviewers like this model for its aesthetically pleasing theoretical properties. A few of them are described here, beginning with parameter estimation. For n pairs of (x_i, y_i),

$$y_i = \alpha + \beta x_i + \epsilon_i \tag{19}$$

To minimize the sum of squared errors

$$\sum_i \epsilon_i^2 = \sum_i (y_i - \alpha - \beta x_i)^2 \tag{20}$$

Taking the partial derivatives w.r.t. α and β and setting them equal to 0 gives

$$\begin{aligned} \sum_i (y_i - \hat{\alpha} - \hat{\beta} x_i) &= 0 \\ \sum_i (y_i - \hat{\alpha} - \hat{\beta} x_i) x_i &= 0 \end{aligned}$$

From just the first equation, we can write

$$n\bar{y} = n\hat{\alpha} + n\hat{\beta}\bar{x} \implies y = \hat{\alpha} + \hat{\beta}\bar{x} \implies \hat{\alpha} = \bar{y} - \hat{\beta}\bar{x} \tag{21}$$

The second equation gives

$$\sum_i x_i y_i = \hat{\alpha} n \bar{x} + \hat{\beta} \sum_i x_i^2 \tag{22}$$

and substituting what we derived gives

$$\begin{aligned} \sum_i x_i y_i &= (\bar{y} - \hat{\beta}\bar{x}) n \bar{x} + \hat{\beta} \sum_i x_i^2 \\ &= n\bar{x}\bar{y} + \hat{\beta}\bigg(\Big(\sum_i x_i^2 \Big) - n\bar{x}^2 \bigg) \end{aligned}$$

and so we have

$$\hat{\beta} = \frac{\big(\sum_i x_i y_i\big) - n\bar{x}\bar{y}}{\big(\sum x_i^2\big) - n\bar{x}^2} = \frac{\sum_i x_i y_i - \bar{x} y_i}{\sum x_i^2 - \bar{x} x_i} = \frac{\sum_i (x_i - \bar{x}) y_i}{\sum_i (x_i - \bar{x}) x_i} \tag{23}$$

Now we can use the identity

$$\sum_i (x_i - \bar{x})(y_i - \bar{y}) = \sum_i y_i (x_i - \bar{x}) = \sum_i x_i (y_i - \bar{y})$$

[6] I've included a separate section on this since this was especially important for quant interviews.

to substitute both the numerator and denominator of the equation to

$$\hat{\beta} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = \frac{\mathrm{cov}(x, y)}{\mathrm{var}(x)} = \rho_{xy} \frac{s_y}{s_x}$$

where ρ_{xy} is the correlation between x and y, and the variance and covariance represent the sample variance and covariance (indicated in lower case letters). Therefore, the correlation coefficient ρ_{xy} is precisely equal to the slope of the best fit line when x and y have been standardized first, i.e. $s_x = s_y = 1$.

Example 1.2 (Switching Variables)

Say that we are fitting Y onto X in a simple regression setting with MLE β_1, and now we wish to fit X onto Y. How will the MLE slope change? We can see that

$$\beta_1 = \rho \frac{s_y}{s_x}, \quad \beta_2 = \rho \frac{s_x}{s_y}$$

and so

$$\beta_2 = \rho^2 \frac{1}{\rho} \frac{s_x}{s_y} = \rho^2 \frac{1}{\beta_1} = \beta_1 \frac{\mathrm{var}(x)}{\mathrm{var}(y)}$$

The reason for this is because regression lines don't necessarily correspond to one-to-one to a casual relationship. Rather, they relate more directly to a conditional probability or best prediction.

The **coefficient of determination** R^2 is a measure tells you how well your line fits the data. When you have your y_i's, their deviation around its mean is captured by the sample variance $s_y^2 = \sum_i (y_i - \bar{y})^2$. When we fit our line, we want the deviation of y_i around our predicted values $\hat{y}_i$, i.e. our sum of squared loss $\sum_i (y_i - \hat{y}_i)^2$, to be lower. Therefore, we can define

$$R^2 = 1 - \frac{\text{MSELoss}}{\mathrm{var}(y)} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

In simple linear regression, we have

$$R^2 = \rho_{yx}^2$$

An R^2 of 0 means that the model does not improve prediction over the mean model and 1 indicates perfect prediction. However, a drawback of R^2 is that it can increase if we add predictors to the regression model, leading to a possible overfitting.

Theorem 1.9 ()

The residual sum of squares (RSS) is equal to the a proportion of the variance of the y_i's.

$$\text{RSS} = \sum (y_i - \hat{y}_i)^2 = (1 - \rho^2) \sum (y_i - \bar{y})^2 \tag{24}$$

1.3 Weighted Least Squares

1.4 Mean Absolute Error

1.5 Significance Tests

This is not as emphasized in the machine learning literature, but it is useful to know from a statistical point of view.[7]

[7]This is also asked in quant interviews.

1.5.1 T Test

Given some multilinear regression problem where we must estimate $\boldsymbol{\beta} \in \mathbb{R}^{D+1}$ (D coefficients and 1 bias), we must determine whether there is actually a linear relationship between the x and y variables in our dataset $\mathcal{D}$. Say that we have a sample of N points $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$. Then, for each ensemble of datasets $\mathcal{D}$ that we sample from the distribution $(X \times Y)^N$, we will have some estimator $\boldsymbol{\beta}$ for each of them. This will create a sampling distribution of $\boldsymbol{\beta}$'s where we can construct our significance test on.

So what should our sampling distribution of $\hat{\boldsymbol{\beta}}$ be? It is clearly normal since it is just a transformation of the normally distributed Y: $\hat{\boldsymbol{\beta}} \sim N(\boldsymbol{\beta}, \sigma^2 (X^T X)^{-1})$. Therefore, only considering one element β_i here,

$$\frac{\hat{\beta}_i - \beta_i}{\sigma \sqrt{(X^T X)^{-1}_{ii}}} \sim N(0, 1)$$

But the problem is that we don't know the true σ^2, and we are estimating it with $\hat{\sigma}^2$. If we knew the true σ^2 then this would be a normal, but because of this estimate, our normalizing factor is also random. It turns out that the residual sum of squares (RSS) for a multiple linear regression

$$\sum_i (y_i - x_i^T \beta)^2$$

follows a χ^2_{n-d} distribution. Additionally from the χ^2 distribution of RSS we have

$$\frac{(n-d)\hat{\sigma}^2}{\sigma^2} \sim \chi^2_{n-d}$$

where we define $\hat{\sigma}^2 = \frac{\text{RSS}}{n-d}$ which is an unbiased estimator for σ^2. Now there is a theorem that says that if you divide a $N(0,1)$ distribution by a χ^2_k / k distribution (with k degrees of freedom), then it gives you a t-distribution with the same degrees of freedom. Therefore, we divide

$$\frac{\frac{\hat{\beta}_i - \beta_i}{\sqrt{(X^T X)^{-1}_{ii}}}}{\hat{\sigma}} = \frac{\sigma \sim N(0,1)}{\sigma \chi^2_{n-d}/(n-d)} = \frac{\sim N(0,1)}{\chi^2_{n-d}/(n-d)} = t_{n-d}$$

where the standard error of the distribution is

$$\text{SE}(\hat{\beta}_i) = \sigma_{\hat{\beta}_i} = \sigma \sqrt{(X^T X)^{-1}_{ii}}$$

In ordinary linear regression, we have the null hypothesis $h_0 : \beta_i = 0$ and the alternative $h_a : \beta_i \neq 0$ for a two sided test or $h_a : \beta_i > 0$ for a one sided test. Given a certain significance level, we compute the critical values of the t-distribution at that level and compare it with the test statistic

$$t = \frac{\hat{\beta} - 0}{\text{SE}(\hat{\beta})}$$

Now given our β, how do we find the standard error of it? Well this is just the variance of our estimator $\boldsymbol{\beta}$, which is $\hat{\sigma}^2 (\mathbf{X}^T \mathbf{X})^{-1}$, where $\hat{\sigma}^2$ is estimated by taking the variance of the residuals ϵ_i. When there is a single variable, the model reduces to

$$y = \beta_0 + \beta_1 x + \epsilon$$

and

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \qquad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

and so

$$(\mathbf{X}'\mathbf{X})^{-1} = \frac{1}{n\sum x_i^2 - (\sum x_i)^2} \begin{pmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & n \end{pmatrix}$$

and substituting this in gives

$$\sqrt{\widehat{\text{Var}}(\hat{\beta}_1)} = \sqrt{[\hat{\sigma}^2(\mathbf{X}'\mathbf{X})^{-1}]_{22}} = \sqrt{\frac{\hat{\sigma}^2}{\sum x_i^2 - (\sum x_i)^2}} = \sqrt{\frac{\hat{\sigma}^2}{\sum (x_i - \bar{x}_i)^2}}$$

Example 1.3 ()

Given a dataset

```
Hours Studied for Exam 20 16 20 18 17 16 15 17 15 16 15 17 16 17 14
Grade on Exam 89 72 93 84 81 75 70 82 69 83 80 83 81 84 76
```

The hypotheses are $h_0 : \beta = 0$ and $h_a : \beta \neq 0$, and the degrees of freedom for the t-test is $df = N - (D+1) = 13$, where $N = 15$ is the number of datapoints and $D = 1$ is the number of coefficients (plus the 1 bias term). The critical values is ± 2.160, which can be found by taking the inverse CDF of the t-distribution evaluated at 0.975.
Now we calculate the t score. We have our estimate $\beta_1 = 3.216, \beta_0 = 26.742$, and so we calculate

$$\hat{\sigma}^2 = \frac{1}{15}\sum_{i=1}^{15} \big(y_i - (3.216x_i + 26.742)\big) = 13.426$$

$$\sum_i (x_i - \hat{x}_i)^2 = 41.6$$

and therefore, we can compute

$$t = \frac{\beta_1}{\sqrt{\hat{\sigma}^2 / \sum_i (x_i - \hat{x}_i)^2}} = \frac{3.216}{\sqrt{13.426/41.6}} = 5.661$$

and therefore, this is way further than our critical value of 2.16, meaning that we reject the null hypothesis.

Note that when multicolinearity is present, then $\sum_i (x_i - \hat{x}_i)^2$ will be very small causing the denominator to blow up, and therefore you cannot place too much emphasis on the interpretation of these statistics. While it is hard to see for the single linear regression case, we know that some eigenvalue of $(\mathbf{X}^T\mathbf{X})^{-1}$ will blow up, causing the diagonal entries $(\mathbf{X}^T\mathbf{X})^{-1}_{ii}$ to be very small. When we calculate the standard error by dividing by this small value, the error blows up.

Theorem 1.10 ()

We can compute this t-statistic w.r.t. just the sample size n and the correlation coefficient ρ as such.

$$t = \frac{\hat{\beta} - 0}{\text{SE}(\hat{\beta})}$$

and the denominator is simply

$$\text{SE}(\hat{\beta}) = \sqrt{\frac{\frac{1}{n-1}\sum (y_i - \hat{y})^2}{\sum (x_i - \bar{x})^2}} \implies t = \frac{\hat{\beta}\sqrt{\sum (x_i - \bar{x})^2}\sqrt{n-1}}{\sqrt{\sum (y_i - \hat{y})^2}} = \frac{\hat{\beta}\sqrt{\sum (x_i - \bar{x})^2}\sqrt{n-1}}{\sqrt{(1-\rho^2)}\sqrt{\sum (y_i - \bar{y})^2}}$$
$$= \frac{\rho}{\sqrt{1-\rho^2}}\sqrt{n-1}$$

where the residual sum of squares on the top can be substituted according to our theorem. Therefore

$$t = \frac{\rho}{\sqrt{1-\rho^2}}\sqrt{n-1} \tag{25}$$

1.5.2 F Test

Given that you have n data points that have been fit on a linear model, the F-statistic is based on the ratio of two variances.

1.6 Bayesian Linear Regression

2 High Dimensional Linear Regression

Now supposed that $d > n$, then the first problem is that we can no longer use least squares since $X^T X$ is no longer invertible and the same problem happens with maximum likelihood. This is known as the **high dimensional** or **large p, small n** problem. The most straightforward way is simply to reduce the covariates to a dimension smaller than n. This can be done with three ways.

1. We perform PCA on the X and use the first k principal components where $k < n$.
2. We cluster the covariates based on their correlation. We can use one feature from each cluster or take the average of the covariates within each cluster.
3. We can screen the variables by choosing the k features that have the largest correlation with Y.

Once this is done, we are back in the low dimensional regime and can use least squares. Essentially, this is a way to find a good subset of the covariates, which can be formalized by the following. Let S be a subset of $[d]$ and let $X_S = (X_j : j \in S)$. If the size of S is not too large, we can regress Y on X_S instead of X.

Definition 2.1 (Best Subset Regression)

Fix $k < d$ and let $\mathcal{S}_k$ denote all subsets of size k. For a given $S \in \mathcal{S}_k$, let β_S be the best linear predictor for the subset S. We want to find the best subset S that minimizes the loss

$$\mathbb{E}[(Y - \beta_S^T X_S)^2] \tag{26}$$

which is equivalent to finding

$$\underset{\beta}{\operatorname{argmin}} \, \mathbb{E}[(Y - \beta^T X)^2] \text{ subject to } ||\beta||_0 \leq k \tag{27}$$

where $||\beta||_0$ is the number of non-zero entries in β.

There will be a bias variance tradeoff. As k increases, the bias decreases but the variance increases. We can approximate the risk with the training error, but the minimization is over all subset of size k, and so this is NP-hard. Therefore, best subset regression is infeasible, but we can approximate best subset regression in two different ways.

1. A greedy approximation leads to *forward stepwise regression*.
2. A convex relaxation of the problem leads to the *LASSO* regression.

It turns out that the theoretical guarantees and computational time for both are the same, but the Lasso is much more popular. It may be due to a cleaner form or that it's easier to study, but who knows.

A completely separate way is to use all the covariates, but instead of least squares, we shrink the coefficients towards 0. This is called *ridge regression* and is an example of a *shrinkage model*.

2.1 Ridge Regression

Ridge regression is used both in the high dimensional case or when our function space is too large/complex, which leads to overfitting. In the overfitting case, we have seen that either decreasing our function space or getting more training data helps. Another popular way is to add a **regularizing term** to the error function in order to discourage the coefficients from reaching large values, effectively limiting the variance over $\mathcal{D}$.

Definition 2.2 (Ridge Regression)

In **ridge regression**, we minimize

$$L(\beta) = ||Y - X\beta||^2 + \lambda ||\beta||^2 \tag{28}$$

where we penalize according to the L2 norm of the coefficients.

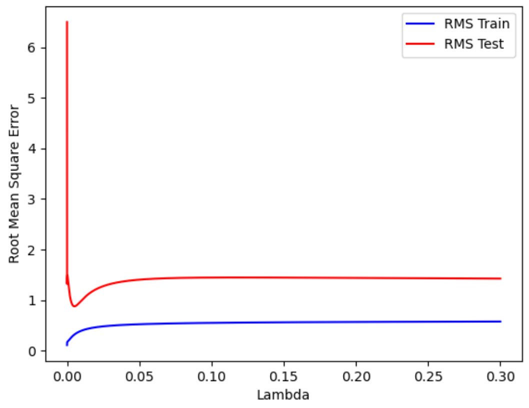


Figure 1: Even with a slight increase in the regularization term λ, the RMS error on the testing set heavily decreases.

Theorem 2.1 (Least Squares Solution for Ridge Regression)

The minimizer of the ridge loss is

$$\hat{\beta} = (X^T X + \lambda I)^{-1} X^T Y \tag{29}$$

Proof.

TBD

Theorem 2.2 (Bias Variance Decomposition of Ridge Regression)

TBD

From a computational point of view, we can see that by adding the λI term, it *dampens* the matrix so that it does become invertible (or well conditioned), allowing us to find a solution. The higher the λ term, the higher the damping effect. The next theorem compares the performance of the best ridge regression estimator to the best linear predictor.

Theorem 2.3 (Hsu, Kakade, Zhang, 2014 [HKZ14])

Suppose that $||X_i|| \leq r$ and let $\beta^T x$ be the best linear approximation to $m(x)$. Then, with probability at least $1 - 4e^{-t}$, we have

$$r(\hat{\beta}) - r(\beta) \leq \left(1 + O\left(\frac{1 + r^2/\lambda}{n}\right)\right) \frac{\lambda ||\beta||^2}{2} + \frac{\sigma^2}{n} \frac{\text{Tr}(\Sigma)}{2\lambda} \tag{30}$$

We can see that the λ term exists in the numerator on $\frac{\lambda||\beta||^2}{2}$ and in the denominator on $\frac{\text{Tr}(\Sigma)}{2\lambda}$. This is the bias variance tradeoff. The first term is the bias term, which is the penalty for not being able to fit the data as well. The second term is the variance term, which is the penalty for having a more complex model. So our optimal λ in the theoretical sense would be the one that minimizes the sum of these two terms. In practice, it's not this clean since we have unknown quantities in the formula, but just like how we did cross validation over the model complexity, we can also do cross validation over the λ. The decomposition above just gives you a theoretical feeling of how these things trade off.

Code 2.1 (MWS of Ridge Regression in scikit-learn)

```python
import numpy as np
from sklearn.linear_model import Ridge

X = np.random.randn(10, 5)
y = np.random.randn(10)
# regularization parameter
model = Ridge(alpha=1.0)
model.fit(X, y)
print(model.score(X, y))
print(model.intercept_)
print(model.coef_)
print(model.predict(np.array([[1, 2, 3, 4, 5]])))
```

```
0.8605535024325397
-0.28291076492665157
[-0.10400521 -0.7587073
    -0.05116735  1.16236649
    -0.0401323 ]
[2.39097184]
.
.
.
.
.
.
```

Question 2.1 (To Do)

Bayesian interpretation of ridge regression.

2.2 Forward Stepwise Regression

Forward stepwise regression is a greedy algorithm that starts with an empty set of covariates and adds the covariate that most improves the fit. It avoids the NP-hardness of the best subset regression by adding covariates one by one.

Definition 2.3 (Greedy Forward Stepwise Regression)

Given your data $\mathcal{D}$, let's first standardize it to have mean 0 and variance 1.[a] You start off with a set $\mathcal{Q} = \{\}$ and choose the number of parameters K.

1. With each covariate $X = (X_1, \ldots, X_n)$, we compute the correlation between it and the Y, which reduces to the inner product (since we standardized).
$$\rho_j = \langle Y, X_{:,j} \rangle = \frac{1}{n} \sum_{i=1}^n Y_i X_{ji} \tag{31}$$
2. Then, we take the covariate index that has the highest empirical correlation with Y, add it to $\mathcal{Q}$ and regress Y only on this covariate.
$$q_1 = \operatorname*{argmax}_j \rho_j, \quad \mathcal{Q} = \{q_1\}, \quad \hat{\beta}_{q_1} = \operatorname*{argmin}_\beta \frac{1}{n} ||Y - X_{:,q_1}\beta||^2 \tag{32}$$
3. Then you repeat the process. You take the residual values $r = Y - X_{:,q_1}\hat{\beta}_{q_1} \in \mathbb{R}^n$ compute the correlation between r and the remaining covariates, and pick our the maximum covariate index q_2. Then, you *repeat the regression from start* with these two covariates
$$q_2 = \operatorname*{argmax}_j \langle r, X_{:;j} \rangle, \quad \mathcal{Q} = \{q_1, q_2\}, \quad \hat{\beta}_{q_1,q_2} = \operatorname*{argmin}_\beta \frac{1}{n} ||Y - X_{:,[q_1,q_2]}\beta||^2 \tag{33}$$
Note that you're not going to get the same coefficient for $\hat{\beta}_{q_1}$ as before since you're doing two variable regression.

4. You get out the residual values $r = Y - X_{:,[q_1,q_2]}\hat{\beta}_{q_1,q_2} \in \mathbb{R}^n$ and keep repeating this process until you have K covariates in $\mathcal{Q}$.

[a] This may or may not be a good idea, since the variance of each covariate can tell you a lot about the importance of the covariate.

Again, there is a bias variance tradeoff in choosing the number of covariates K, but through cross-validation, we can find the optimal K. It is also easy to add constraints, e.g. if we wanted to place a restriction that two adjacent covariates can't be chosen, we can easily add this to the algorithm.

Theorem 2.4 (Rate of Convergence for Stepwise Regression)

Let $\hat{f}_K$ be the optimal regressor we get from K covariates in stepwise regression. Then, we have something like

$$||f - \hat{f}||^2 \leq c||f - f_K||^2 + \frac{\log n}{\sqrt{n}} \tag{34}$$

2.2.1 Stagewise Regression

Stagewise regression is a variant of forward stepwise regression where we add the covariate that most improves the fit, but we only take a small step in that direction. This is useful when we have a lot of covariates and we don't want to overfit.

2.3 Lasso Regression

The Lasso approximates the best subset regression by using a convex relaxation. In particular, the norm $||\beta||_0$ is not convex, but the L1 norm $||\beta||_1$ is. Therefore, we want relax our constraint equation as such:

$$\operatorname*{argmin}_{||\beta||_0 \leq L} r(\beta) \mapsto \operatorname*{argmin}_{||\beta||_1 \leq L} r(\beta) \tag{35}$$

This gives us a convex problem, which we can then solve. In fact, it turns out that optimizing the risk given the L1 restriction on the norm is equivalent to minimizing the risk plus a L1 penalty, as this is the Lagrangian form of the original equation (this is in convex optimization). Therefore, there exists a pair (L, λ) for which the two problems are equivalent

$$\operatorname*{argmin}_{||\beta||_1 \leq L} r(\beta) = \operatorname*{argmin}_{\beta} r(\beta) + \lambda||\beta||_1 \tag{36}$$

Definition 2.4 (LASSO Regression)

In **lasso regression**, we minimize the loss defined

$$\hat{R}(\beta) = \frac{1}{n}\sum_{i=1}^n (y^{(i)} - \beta^T x^{(i)})^2 + \lambda||\beta||_1 \tag{37}$$

where we penalize according to the L1 norm of the coefficients.

A question arises: Why use the L1 norm? The motivation behind this is that we want to model the L0 norm as much as possible but at the same time we want it to be convex. This turns out to be precisely the L1 norm. Unfortunately, there is no closed form solution for this estimator, but in convex optimization, we can prove that this estimator is sparse. That is, for large enough λ, many of the components of $\hat{\beta}$ are 0. The classical intuition for this is the figure below, where the equipotential lines have "corners." In fact for any $0 < p < 1$, there are also corners, but the problem with using these p-norms is that they are not convex.

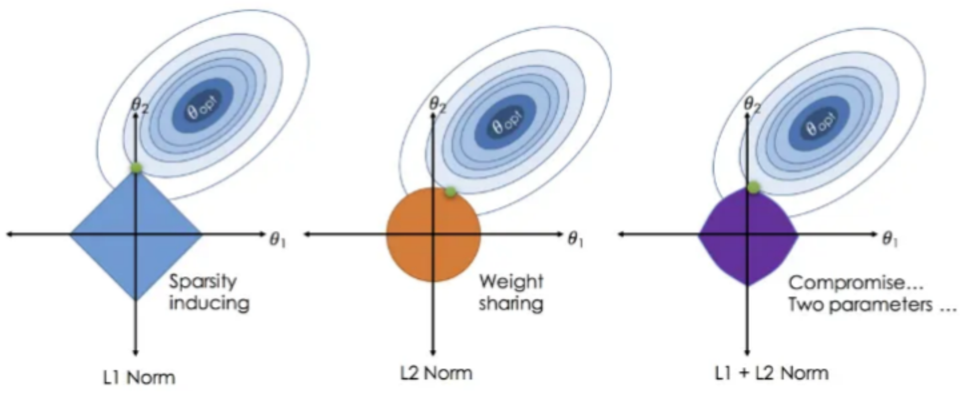


Figure 2: The ridge regularizer draws equipotential circles in our parameter space. The lasso draws a diamond, which tends to give a sparser solution since the loss is most likely to "touch" the corners of the contour plots of the regularizer. The elastic net is a linear combination of the ridge and lasso regularizers.

To motivate this even further, let us take the two vectors

$$a = \left(\frac{1}{\sqrt{d}}, \ldots, \frac{1}{\sqrt{d}} \right) \qquad b = (1, 0, \ldots, 0) \tag{38}$$

Then the L0, L1, and L2 norms of a are $d, \sqrt{d}, 1$ and those of b are $1, 1, 1$. We want to choose a norm that capture the sparsity of b and distinguishes it from b., The L0 norm clearly does this, but the L2 norm does not. The L1 norm is a good compromise between the two.

This now raises the question of how to determine a suitable regularization parameter λ. The next theorem shows a nice concentration property of the Lasso for bounded covariates.

Theorem 2.5 (Concentration of Lasso)

Given (X, Y), assume that $|Y| \leq B$ and $\max_j |X_j| \leq B$. Let

$$\beta^* = \operatorname*{argmin}_{||\beta||_1 \leq L} r(\beta) \tag{39}$$

be the best sparse linear predictor in the L1 sense, where $r(\beta) = \mathbb{E}[(Y - \beta^T X)^2]$. Let our lasso estimator be

$$\hat{\beta} = \operatorname*{argmin}_{||\beta||_1 \leq L} \hat{r}(\beta) = \operatorname*{argmin}_{||\beta||_1 \leq L} \frac{1}{n} \sum_{i=1}^n (Y_i - \beta^T X_i)^2 \tag{40}$$

which minimizes the empirical risk. Then, with probability at least $1 - \delta$,

$$r(\hat{\beta}) \leq r(\beta^*) + \sqrt{\frac{16(L+1)^4 B^2}{n} \log \left(\frac{\sqrt{2}d}{\sqrt{\delta}} \right)} \tag{41}$$

Proof.

Code 2.2 (MWS of Lasso Regression in scikit-learn)

```python
from sklearn.linear_model import Lasso

X = np.random.randn(10, 5)
y = np.random.randn(10)
# regularization parameter
model = Lasso(alpha=1e-1)
model.fit(X, y)
print(model.score(X, y))
print(model.intercept_)
print(model.coef_)
print(model.predict(np.array([[1, 2, 3, 4, 5]])))
```

```
0.47590269719236045
-0.8861298412689853
[0.         0.10767647
    0.24172197 0.7427863  0.
          ]
[3.02553422]
.
.
.
.
.
```

2.3.1 Soft Thresholding and Proximal Gradient Descent

2.4 Bayesian Regularization with Priors

We will now demonstrate how having a normal $\alpha\mathbf{I}$ prior around the origin in a Bayesian setting is equivalent to having a ridge penalty of $\lambda = \sigma^2/\alpha^2$ in a frequentist setting. If we have a Gaussian prior of form

$$p(\mathbf{w} \mid \alpha^2) = N(\mathbf{w} \mid \mathbf{0}, \alpha^2 \mathbf{I}) = \bigg(\frac{1}{2\pi\alpha^2}\bigg)^{M/2} \exp\bigg(-\frac{1}{2\alpha^2}||\mathbf{w}||_2^2\bigg)$$

We can use Bayes rule to compute

$$\begin{aligned} p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}, \alpha^2, \sigma^2) &\propto p(\mathbf{Y} \mid \mathbf{w}, \mathbf{X}, \alpha^2, \sigma^2)\, p(\mathbf{w} \mid \mathbf{X}, \alpha^2, \sigma^2) \\ &= \bigg[\prod_{n=1}^N p(y^{(n)} \mid \mathbf{w}, \mathbf{x}^{(n)}, \alpha^2, \sigma^2)\bigg] p(\mathbf{w} \mid \mathbf{X}, \alpha^2, \sigma^2) \\ &= \bigg[\prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\bigg(-\frac{(y^{(n)} - h_{\mathbf{w}}(x^{(n)}))^2}{2\sigma^2}\bigg)\bigg] \cdot \bigg(\frac{1}{2\pi\alpha^2}\bigg)^{M/2} \exp\bigg(-\frac{1}{2\alpha^2}||\mathbf{w}||_2^2\bigg) \end{aligned}$$

and taking the negative logarithm gives us

$$\ell(\mathbf{w}) = \frac{1}{2\sigma^2}\sum_{n=1}^N \big(y^{(n)} - h_{\mathbf{w}}(\mathbf{x}^{(n)})\big)^2 + \frac{N}{2}\ln\sigma^2 + \frac{N}{2}\ln(2\pi) - \frac{M}{2}\ln(2\pi\alpha^2) + \frac{1}{2\alpha^2}||\mathbf{w}||_2^2$$

taking out the constant terms relative to $\mathbf{w}$ and multiplying by $2\sigma^2$ (which doesn't affect optima) gives us the ridge penalized error with a penalty term of $\lambda = \sigma^2/\alpha^2$.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^N \big(y^{(n)} - h_{\mathbf{w}}(\mathbf{x}^{(n)})\big)^2 + \frac{\sigma^2}{\alpha^2}||\mathbf{w}||_2^2$$

But minimizing this still gives a point estimate of $\mathbf{w}$, which is not the full Bayesian treatment. In a Bayesian setting, we are given the training data $(\mathbf{X}, \mathbf{Y})$ along with a new test point $\mathbf{x}'$ and want to evaluate the predictive distribution $p(y \mid \mathbf{x}', \mathbf{X}, \mathbf{Y})$. We can do this by integrating over $\mathbf{w}$.

$$\begin{aligned} p(y \mid \mathbf{x}', \mathbf{X}, \mathbf{Y}) &= \int p(y \mid \mathbf{x}', \mathbf{w}, \mathbf{X}, \mathbf{Y})\, p(\mathbf{w} \mid \mathbf{x}', \mathbf{X}, \mathbf{Y})\, d\mathbf{w} \\ &= \int p(y \mid \mathbf{x}', \mathbf{w})\, p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})\, d\mathbf{w} \end{aligned}$$

where we have omitted the irrelevant variables, along with α^2 and σ^2 to simplify notation. By substituting the posterior $p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})$ with a normalized version of our calculation above and by noting that

$$p(y \mid \mathbf{x}', \mathbf{w}) = N(y \mid h_{\mathbf{w}}(\mathbf{x}'), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\bigg(- \frac{\big(y - h_{\mathbf{w}}(\mathbf{x}')\big)^2}{2\sigma^2} \bigg)$$

Now this integral may or may not have a closed form, but if we consider the polynomial regression with the hypothesis function of form

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{M-1} x^{M-1}$$

then this integral turns out to have a closed form solution given by

$$p(y \mid \mathbf{x}', \mathbf{X}, \mathbf{Y}) = N\big(y \mid m(x'), s^2(x')\big)$$

where

$$\begin{aligned} m(x') &= \frac{1}{\sigma^2} \boldsymbol{\phi}(x')^T \mathbf{S} \bigg(\sum_{n=1}^N \boldsymbol{\phi}(x^{(n)}) y^{(n)} \bigg) \\ s^2(x') &= \sigma^2 + \boldsymbol{\phi}(x')^T \mathbf{S} \boldsymbol{\phi}(x') \\ \mathbf{S}^{-1} &= \alpha^{-2} \mathbf{I} + \frac{1}{\sigma^2} \sum_{n=1}^N \boldsymbol{\phi}(x^{(n)}) \boldsymbol{\phi}(x')^T \end{aligned}$$

and $\boldsymbol{\phi}(x)$ is the vector of functions $\phi_i(x) = x^i$ from $i = 0, \ldots, M-1$.

3 Nonparametric Regression

3.1 K Nearest Neighbors Regression

When we want to do nonparametric regression, i.e. when dealing with nonlinear functions, we can construct a function that uses local averaging of its nearby points.

Example 3.1 (Local Averaging)

Say that we want to fit some function through a series of datapoints in simple regression (one covariate). Then, what we can do is take some sliding window and our vale of the function at a point x is the average of all values in the window $[x - \delta, x + \delta]$.

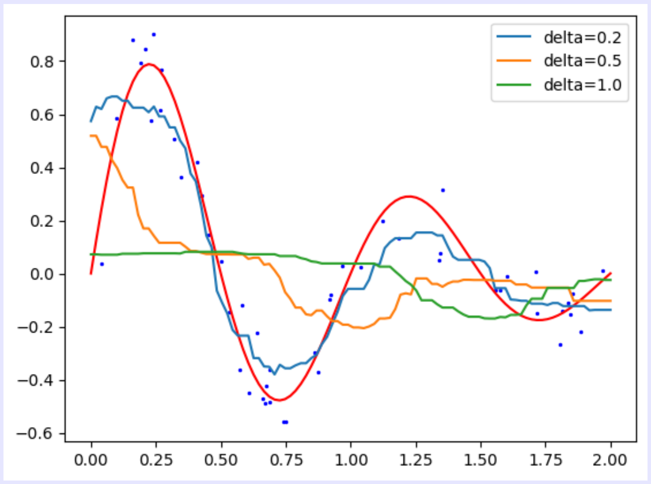


Figure 3: K means smoother

Code 3.1 (MWS of K Nearest Neighbor Regression in scikit-learn)

Local averaging is implemented as the K nearest neighbor regressor in scikit learn. It is slightly different in the way that it doesn't use the points within a certain δ away but rather the K nearest points. Either way, a minimal working example of this is

```python
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=2)
neigh.fit(X, y)
print(neigh.predict([[1.5]]))
```

Note that since $\hat{f}$ is a combination of step functions, this makes it discontinuous at points.

3.2 Kernel Regression and Linear Smoothers

K nearest neighbor regression puts equal weights on both near and far points, as long as they are in the window. This may not be ideal, so a simple modification is to *weigh* these points according to their distance from the middle x. We can do this with a kernel, as the name suggests. Now this is not the same thing as a Mercer kernel in RKHS, so to distinguish that I will call it a *local averaging kernel.*

Definition 3.1 (Local Averaging Kernel)

A **kernel** is any smooth, symmetric, and non-negative function $K : \mathbb{R} \to \mathbb{R}$.

Definition 3.2 (Kernel Regression)

Given some datapoints, X, our fitted regressor is of form

$$\hat{f}(X) = \frac{\sum_i Y_i K\left(\frac{||X - X_i||}{h}\right)}{\sum_i K\left(\frac{||X - X_i||}{h}\right)} \tag{42}$$

where h is the **bandwidth** and the denominator is made sure so that the coefficients sum to 1. To get a clearer picture, we are really taking the weighted average of the Y_i's.

$$\hat{f}(X) = \sum_i Y_i \ell_i(X) \text{ where } \sum_i \ell_i(X) = 1 \tag{43}$$

Denoting $Y = (Y_1, \ldots, Y_n) \in \mathbb{R}^n$ and the vector $f(X) = (f(X_1), \ldots, f(X_n))$, if we can write the kernel function as $\hat{Y} = \hat{f}(X) = SY$, which in matrix form, is

$$\begin{bmatrix} \hat{Y}_1 \\ \vdots \\ \hat{Y}_n \end{bmatrix} = \begin{bmatrix} \hat{f}(X_1) \\ \vdots \\ \hat{f}(X_n) \end{bmatrix} = \begin{bmatrix} \ell_1(X_1) & \cdots & \ell_n(X_1) \\ \vdots & \ddots & \vdots \\ \ell_1(X_n) & \cdots & \ell_n(X_n) \end{bmatrix} \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \tag{44}$$

then we say that we have a **linear smoother**, with stochastic matrix S being our **smoothing matrix**.

The reason we'd like to have the weights to sum to 1 is that if we had data that was constant (i.e. all Y_i's are the same), then the fitted function should be constant at that value as well. Furthermore, the theme of linearity is important and will be recurring. The kernel estimator is defined for all X, but it's important to see its behavior at the training points X_i. The estimator $\hat{Y} = \hat{f}(X)$ is a linear combination of the Y_i's, and the coefficients $\ell_i(X_j)$ depend on the values of X_j. Therefore, we have $\hat{Y} = SY$, which is very similar to the equation $\hat{Y} = HY$ in linear regression, where H is the hat matrix that projects Y onto the column space of X. Nonparametric regression has the same form, but rather than being a projection, it is a linear smoothing matrix. Therefore, this theme unifies both linear regression and nonparametric regression. Linear smoothers, spline smoother, Gaussian processes, are all just different choices of the smoothing matrix S. However, not all nonparametric estimators are linear smoothers, as we will see later.

Here are some popular kernels.

Definition 3.3 (Gaussian Kernel)

The **Gaussian kernel** is defined as

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{45}$$

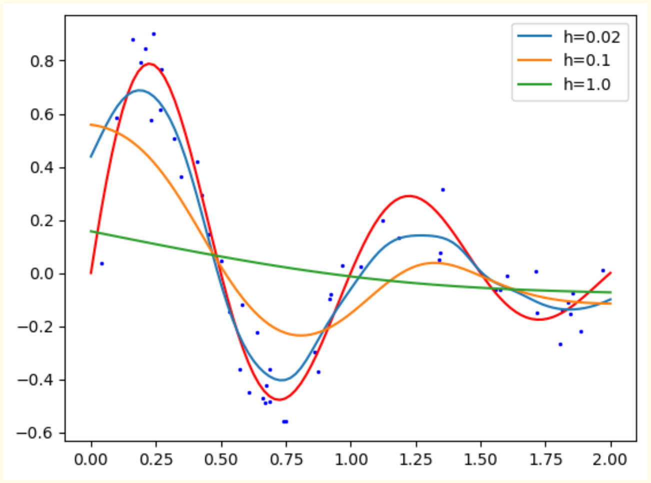


Figure 4: Gaussian kernel.

Definition 3.4 (Box-Car Kernel)

The **Box-Car kernel** is defined as

$$K(x) = \frac{1}{2}\mathbb{1}(|x| \leq 1) \tag{46}$$

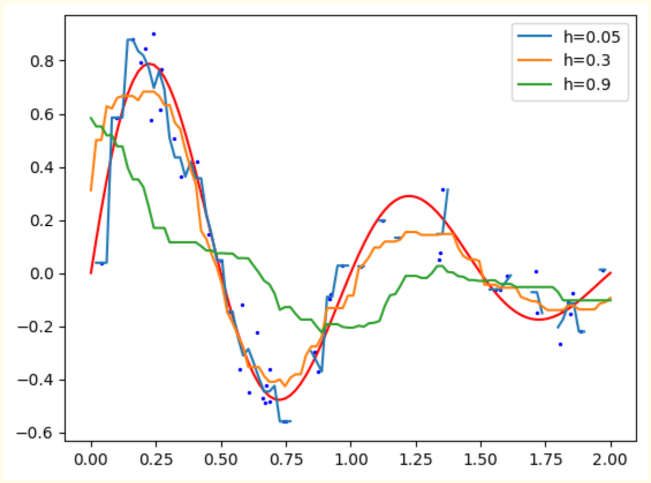


Figure 5: Boxcar kernel.

Definition 3.5 (Epanechnikov Kernel)

The **Epanechnikov kernel** is defined as

$$K(x) = \frac{3}{4}(1 - x^2)\mathbb{1}(|x| \leq 1) \tag{47}$$

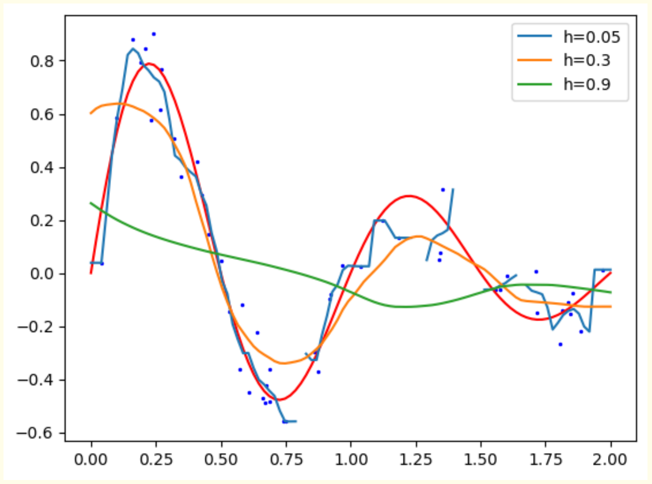


Figure 6: Epanechnikov kernel.

It turns out that from a theoretical point of view, the choice of the kernel doesn't really matter. What really matters is the bandwidth h since that is what determines the bias variance tradeoff. To see why, if $h = 0$, then it will simply interpolate the points and variance is extremely high, and if $h = \infty$, then the fitted function will be constant at $\bar{Y}$, leading to high bias. The following theorem formalizes this.

Theorem 3.1 (Bias Variance Tradeoff of Kernel Regression)

Suppose that $d = 1$ and that m'' is bounded. Also suppose that X has a nonzero, differentiable density p and that the support is unbounded. Then, the risk is

$$\begin{aligned} R_n = \frac{h_n^4}{4} \bigg(\int x^2 K(x) \bigg)^2 \int \bigg(m''(x) + 2m'(x) \frac{p'(x)}{p(x)} \bigg)^2 \, dx \qquad (48) \\ + \frac{\sigma^2 \int K^2(x)\, dx}{n h_n} \int \frac{dx}{p(x)} + o\bigg(\frac{1}{n h_n} \bigg) + o(h_n^4) \qquad (49) \end{aligned}$$

The first term is the squared bias and the second term is the variance.

Proof.

We first denote

$$\hat{f}(X) = \frac{\frac{1}{nh} \sum_{i=1}^n K\Big(\frac{X - X_i}{h}\Big) Y_i}{\frac{1}{nh} \sum_{i=1}^n K\Big(\frac{X - X_i}{h}\Big)} \qquad (50)$$

where the denominator is the kernel density estimator $\hat{p}(X)$. Therefore, we rewrite

$$\begin{aligned} \hat{f}(x) - f(x) &= \frac{\hat{a}(x)}{\hat{p}(x)} - f(x) && (51) \\ &= \bigg(\frac{\hat{a}(x)}{\hat{p}(x)} - f(x) \bigg) \Bigg(\frac{\hat{p}(x)}{p(x) + 1 - \frac{\hat{p}(x)}{p(x)}} \Bigg) && (52) \\ &= \frac{\hat{a}(x) - f(x)\hat{p}(x)}{p(x)} + \frac{(\hat{f}(x) - f(x))(p(x) - \hat{p}(x))}{p(x)} && (53) \end{aligned}$$

as $n \to \infty$ both $\hat{f}(x) - f(x)$ and $p(x) - \hat{p}(x)$ going to 0, and since they're multiplied in the second term, it will go to 0 very fast. So the dominant term is the first term, and we can write the above as approximately

$$\hat{f}(x) - f(x) \approx \frac{\hat{a}(x) - f(x)\hat{p}(x)}{p(x)} \tag{54}$$

TBD continued. Wasserman lecture 6, 10:00.

From the theorem above, we can see that if the bandwidth is small, then h^4 is small and the bias decreases. However, there is a h term in the denominator of the variance term, which also trades it off. We can furthermore see that the bias is sensitive to $p'/p(x)$. This means that if the density is steep, then the bias will be high. This is known as *design bias*, which is an underlying weakness in smoothing kernel regression. Another problem that is not contained in the theorem is the *boundary bias*, which states that if you're near the boundary of the distribution (i.e. near the boundary of its support), then the bias also explodes. This happens to be very nasty especially in high dimensions where most of the data tends to be near the boundary. It turns out that this can be easily fixed with local polynomial regression, which gets rid of this term in the bias without any cost to variance. This means that this is unnecessary bias.

Then you can apply regularization on this to get kernel ridge regression.

Code 3.2 (MWS of Kernel Ridge Regression in scikit learn)

```python
from sklearn.kernel_ridge import KernelRidge
import numpy as np
n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
krr = KernelRidge(alpha=1.0)
krr.fit(X, y)
```

3.3 Local Polynomial Regression

Now another way to think about the kernel estimator is as such. Suppose that you're doing linear regression on a bunch of points and you want to choose a c that minimizes the loss.

$$\sum_i (Y_i - c)^2 \tag{55}$$

You would just pick $c = \hat{Y}$. But if you are given some sort of locality condition, that the value of c should depend more on the values closer to it, you're really now minimizing

$$\sum_i (Y_i - c(x))^2 K\left(\frac{X_i - x}{h}\right) \tag{56}$$

Minimizing this by setting the derivative equal to 0 and solving gives us the kernel estimator. Therefore you're fitting some sort of local constant at a point X. But why fit a local constant, when you can fit a local line or polynomial? This is the idea behind local polynomial regression.

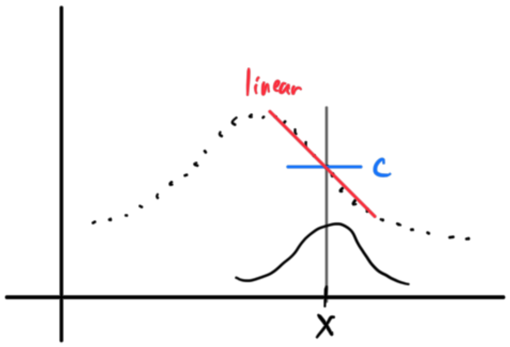


Figure 7: Rather than using a local constant, we can use a local linear estimator.

Therefore, we can minimize the modified loss.

Definition 3.6 (Local Polynomial Estimator)

The **local polynomial estimator** is a local linear kernel smoother that estimates the function $\hat{f}$ that minimizes the following loss.

$$\operatorname*{argmin}_{\beta} \sum_i K\bigg(\frac{X_i - x}{h}\bigg)\big(Y_i - (\beta_0(x) - \beta_1(x)(x - X_i) + \ldots + \beta_k(x)(x - X_i)^k)\big) \tag{57}$$

So we can fit a line

$$f(\mu) \approx \hat{\beta}_0(x) + \hat{\beta}_1(x)(\mu - x) \tag{58}$$

and simply remove the intercept term to get the local linear estimator.

$$\hat{f}(x) = \hat{\beta}_0(x) \tag{59}$$

Note that this is not the same as taking the constant estimate. We are extracting the fitted intercept term and so $\hat{\beta}_0(x) \neq c(x)$.

Theorem 3.2 (Weighted Least Squares)

The solution to the local linear estimator is similar to the weighted least squares solution.

$$\hat{\beta}(x) = \begin{pmatrix} \hat{\beta}_0(x) \\ \hat{\beta}_1(x) \end{pmatrix} = (X^T W X)^{-1} X^T W Y \tag{60}$$

where

$$X = \begin{pmatrix} 1 & X_1 - x \\ \vdots & \vdots \\ 1 & X_n - x \end{pmatrix} \qquad W = \begin{pmatrix} K\Big(\frac{X_1 - x}{h}\Big) & 0 & \cdots & 0 \\ 0 & K\Big(\frac{X_2 - x}{h}\Big) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K\Big(\frac{X_n - x}{h}\Big) \end{pmatrix} \tag{61}$$

Computationally, it's similar to kernel regression and it gets rid of both the boundary and design bias.

3.4 Regularized: Spline Smoothing

This is not local, but it's a linear smoother.

3.5 Regularized: RKHS Regression

This is not local, but it's a linear smoother.

3.6 Additive Models

In the most general case, we want to create nonparametric regression functions of the form

$$Y = f(x_1, \ldots, x_d) + \epsilon \tag{62}$$

We've done this for one dimensional case, but we can extend this to multiple dimensions through additive models of the form

$$Y = \sum_j f_j(x_j) + \epsilon \tag{63}$$

This gives us very interpretable models where we can clearly see the effect of each covariate on Y. Clearly, this is not as flexible as the previous model since they can't capture dependencies, but we can create sub-dependency functions and replace the form above to something like

$$Y = \sum_{i,j} f_{i,j}(x_i, x_j) + \epsilon \tag{64}$$

giving us more flexible models.

3.7 Nonlinear Smoothers, Trend Filtering

Tough example of the Dobbler function (like topologists sine curve). It's a pretty good fit but it's not too good since it's using a linear smoother (homogeneous). So we might need to fit it with nonlinear smoothers.

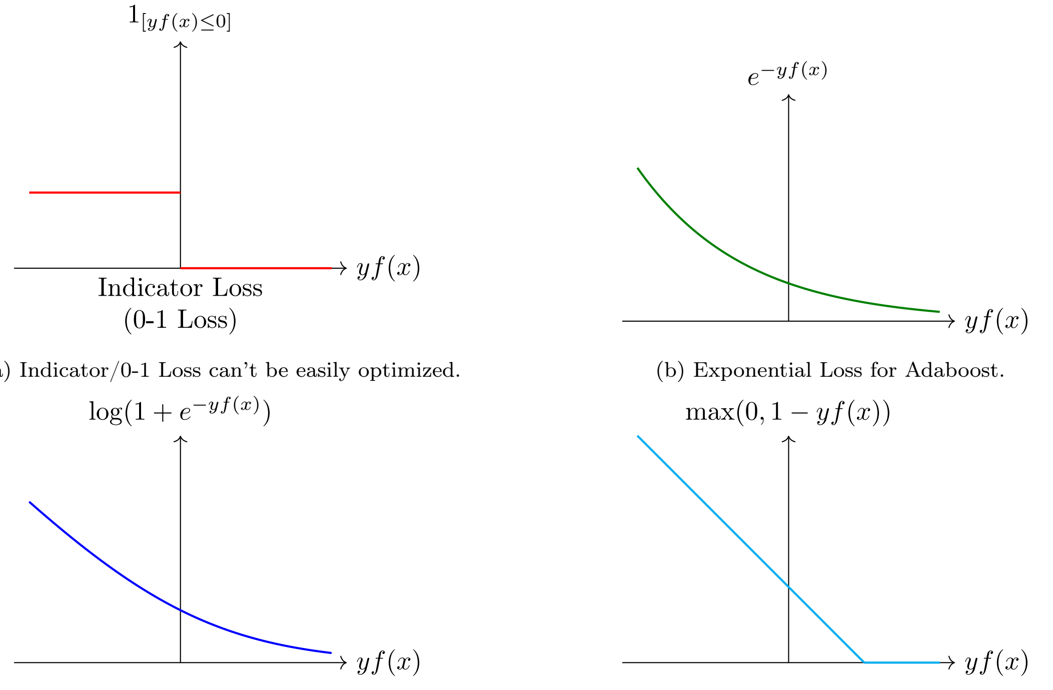
3.8 High Dimensional Nonparametric Regression

3.9 Regression Trees

4 Linear Classification

4.1 Empirical Risk Minimizer

You literally just try to build a hyperplane to minimize the number of misclassifications, but this is not really differentiable and is hard. It's just a stepwise function. Therefore, you use a **surrogate loss function** to approximate the 0-1 loss function. The logistic uses some function, and the SVM uses the smallest convex function to approximate the 0-1 loss function. Here are some examples:



(a) Indicator/0-1 Loss can't be easily optimized.

(b) Exponential Loss for Adaboost.

(c) Log Loss for logistic regression.

(d) Hinge Loss for support vector machines.

Figure 8: Common loss functions used in classification

4.2 Perceptron

Definition 4.1 (Perceptron Model and Loss)

The simplest binary classification model is the **perceptron algorithm**. It is a discriminative parametric model that assigns

$$f_w(x) = \begin{cases} 1 & \text{if } w^T x + b \geq 0 \\ -1 & \text{if } w^T x + b < 0 \end{cases} \tag{65}$$

where we have chosen to label class $C_1 = 1$ and $C_2 = -1$. Note that unlike linear regression (and logistic regression, as we will see later), the perceptron is not a probabilistic model. It is a **discriminant function**, which just gives point estimates of the classes, not their respective probabilities. Like logistic regression, however, it is a linear model, meaning that the decision boundary it creates is always a linear (affine) hyperplane.

To construct the surrogate loss function, we would want a loss that penalizes not only if there is a misclassification, but how *far* that misclassified point is from the boundary. Therefore, if y and $\hat{y} = f_w(x)$ have the same sign, i.e. if $yf_w(x) > 0$, then the penalty should be 0, and if it is < 0, then the penalty should be proportional to the orthogonal distance of the misclassified point to the boundary, which is represented by

$-wTxy$ (where the negative sign makes this cost term positive).

Definition 4.2 (Surrogate Loss for Perceptron)

Therefore, our cost functions would take all the points and penalize all the terms by 0 if they are correctly classified and by $-\mathbf{w}^T \boldsymbol{\phi}^{(n)} y^{(n)}$ if incorrectly classified.

$$L(y, \hat{y}) = \sum_{n=1} [-\mathbf{w}^T \boldsymbol{\phi}^{(n)} y^{(n)}]_+ \text{ where } [f(\mathbf{x})]_+ := \begin{cases} f(\mathbf{x}) & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{else} \end{cases} \tag{66}$$

Note that this is a piecewise linear function and convex.

Code 4.1 (Perceptron in scikit-learn)

Let's implement this in scikit-learn, using two pipelines with different data standardization techniques to see the differences in the perceptron boundary.

```python
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import QuantileTransformer, StandardScaler

pipe1 = Pipeline([
    ("scale", StandardScaler()),
    ("model", Perceptron())
])

pipe2 = Pipeline([
    ("scale", QuantileTransformer(n_quantiles=100)),
    ("model", Perceptron())
])
```

Figure 9

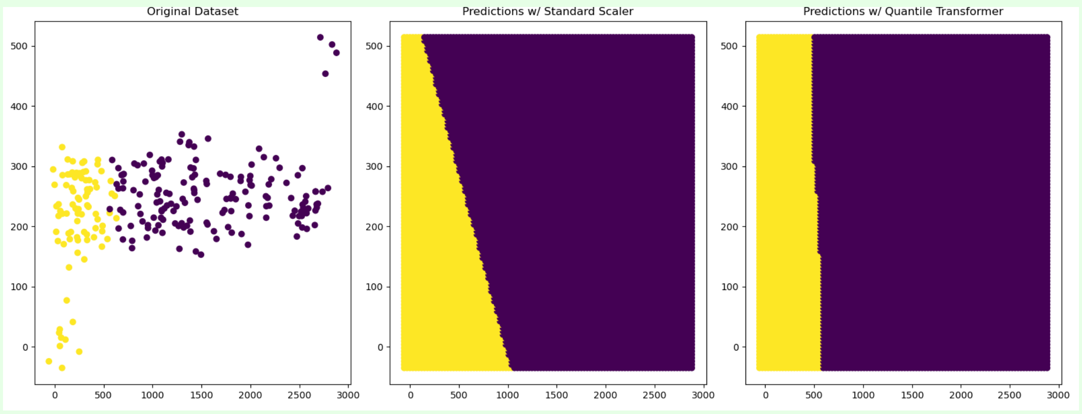


Figure 10: Perceptron Trained on Different Standardized Data

4.3 Logistic and Softmax Regression

We can upgrade from a discriminant function to a discriminative probabilistic model with **logistic regression**. In practice, we usually deal with **probabilistic models** where rather than giving a point estimate $\hat{y}$, we attempt to model the *distribution* $\mathbb{P}_{Y|X=\hat{x}}$. Even though in the end, we will just output the mean μ of this conditional distribution, modeling the distribution allows us to quantify uncertainty in our measurements.

Definition 4.3 (Logistic Regression)

The **logistic regression** model is a linear model of the form

$$f_w(x) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}}, \text{ where } \sigma(x) \coloneqq \frac{1}{1+e^x} \tag{67}$$

It is different from linear regression in two ways:

1. In linear regression, we assumed that the targets are linearly dependent with the covariates as $y = w^T x + b$. However, this means that the hypothesis f_w is unbounded. Since we have two classes (say with labels 0 and 1), we must have some sort of *link function* σ that takes the real numbers and compresses it into the domain $[0, 1]$. Technically, we can choose any continuous, monotinically increasing function from $\mathbb{R}$ to $(0, 1)$. However, the following property of the sigmoid makes derivation of gradients very nice.

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{68}$$

2. Once this is compressed, we assume that the residual distribution is a Bernoulli.

Definition 4.4 (Binary Cross Entropy Loss as Surrogate Loss for Logistic Regression)

The surrogate loss for logistic regression is the **binary cross entropy loss**, which is defined as

$$L(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{69}$$

One important observation to make is that notice that the output of our hypothesis is used as a parameter to define our residual distribution.

1. In linear regression, the f_w was used as the *mean* μ of a Gaussian.
2. In logistic regression, the f_w is used also as the mean p of a Bernoulli.

The reason we want this sigmoid is so that we make the domains of the means of the residuals match the range of the outputs of our model. It's simply a manner of convenience, and in fact we could have really chose any function that maps $\mathbb{R}$ to $(0, 1)$.

Some questions may arise, such as "why isn't the variance parameter of the Gaussian considered in the linear model?" or "what about other residual distributions that have multiple parameters?" This is all answered by generalized linear models, which uses the output of a linear model as a *natural parameter* of the canonical exponential family of residual distributions.

Unfortunately, there is no closed form solution for logistic regression like the least squares solution in linear regression. Therefore, we can only resort to maximum likelihood estimation.

Theorem 4.1 (Maximum Likelihood Estimation for Logistic)

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, our likelihood is

$$L(\theta; \mathcal{D}) = \prod_{i=1}^n p(y^{(i)} \mid x^{(i)}; \theta) = \prod_{i=1}^N \big(h_\theta(x^{(i)})\big)^{y^{(i)}} \big(1 - h_\theta(x^{(i)})\big)^{1-y^{(i)}} \tag{70}$$

We can equivalently minimize its negative log likelihood, giving us the **binary cross entropy** loss function

$$\ell(\theta) = -\log L(\theta) \tag{71}$$

$$= -\sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \tag{72}$$

Now taking the gradient for just a single sample $(x^{(i)}, y^{(i)})$ gives

$$\frac{\partial \ell}{\partial \theta} = \left(\frac{y^{(i)}}{\sigma(\theta^T x^{(i)})} - \frac{1 - y^{(i)}}{1 - \sigma(\theta^T x^{(i)})} \right) \frac{\partial}{\partial \theta} \sigma(\theta^T x^{(i)}) \tag{73}$$

$$= \frac{\sigma(\theta^T x^{(i)}) - y^{(i)}}{\sigma(\theta^T x^{(i)}) \big(1 - \sigma(\theta^T x^{(i)})\big)} \sigma(\theta^T x^{(i)}) \big(1 - \sigma(\theta^T x^{(i)})\big) x^{(i)} \tag{74}$$

$$= \big(h_\theta(x^{(i)}) - y^{(i)}\big) x \tag{75}$$

and summing it over some minibatch $M \subset \mathcal{D}$ gives

$$\nabla_\theta \ell_M = \sum_{(x,y) \in M} (y - h_\theta(x)) x \tag{76}$$

Therefore, the stochastic gradient descent algorithm is

$$\theta = \theta - \eta \nabla_\theta \ell(\theta) \tag{77}$$

$$= \theta - \eta \sum_{(x,y) \in M} (y - h_\theta(x)) x \tag{78}$$

We would like to extend this to the multiclass case.

Definition 4.5 (Softmax Function)

The softmax function is defined

$$o(\mathbf{x}) = \frac{e^{\mathbf{x}}}{||e^{\mathbf{x}}||} = \frac{1}{\sum_j e^{x_j}} \begin{pmatrix} e^{x_1} \\ \vdots \\ e^{x_D} \end{pmatrix} \tag{79}$$

What makes the softmax so popular is that the total derivative turns out to simplify functions a lot. The total derivative of the softmax can be derived as such.

Lemma 4.1 (Derivative of Softmax)

The derivative of the softmax is

$$Do(\mathbf{x}) = \text{diag}(o(\mathbf{x})) - o(\mathbf{x}) \otimes o(\mathbf{x}) \tag{80}$$

where $\otimes$ is the outer product. That is, let y_i be the output of the softmax. Then, for the 4×4 softmax function, we have

$$Do(\mathbf{x}) = \begin{pmatrix} y_1(1-y_1) & -y_1 y_2 & -y_1 y_3 & -y_1 y_4 \\ -y_2 y_1 & y_2(1-y_2) & -y_2 y_3 & -y_2 y_4 \\ -y_3 y_1 & y_3 y_3 & y_3(1-y_3) & -y_3 y_4 \\ -y_4 y_1 & -y_4 y_2 & -y_4 y_3 & y_4(1-y_4) \end{pmatrix} \tag{81}$$

Proof.

We will provide a way that allows us not to use quotient rule. Given that we are taking the partial derivative of y_i with respect to x_j, we can use the log of it to get

$$\frac{\partial}{\partial x_j} \log(y_i) = \frac{1}{y_i} \frac{\partial y_i}{\partial x_j} \implies \frac{\partial y_i}{\partial x_j} = y_i \frac{\partial}{\partial x_j} \log(y_i)$$

Now the partial of the log term is

$$\log y_i = \log \left(\frac{e^{x_i}}{\sum_l e^{x_l}} = x_i - \log \left(\sum_l e^{x_l} \right) \tag{82}$$

$$\frac{\partial}{\partial x_j} \log(y_i) = \frac{\partial x_i}{\partial x_j} - \frac{\partial}{\partial x_j} \log \left(\sum_l e^{x_l} \right) \tag{83}$$

$$= 1_{i=j} - \frac{1}{\sum_l e^{x_l}} e^{x_j} \tag{84}$$

and plugging this back in gives

$$\frac{\partial y_i}{\partial x_j} = y_i (1_{i=j} - y_j) \tag{85}$$

It also turns out that the sigmoid is a specific case of the softmax. That is, given softmax for 2 classes, we have

$$o \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \frac{1}{e^{x_1} + e^{x_2}} \begin{pmatrix} e^{x_1} \\ e^{x_2} \end{pmatrix} \tag{86}$$

So, the probability of being in class 1 is

$$\frac{e^{x_1}}{e^{x_1} + e^{x_2}} = \frac{1}{1 + e^{x_2 - x_1}} \tag{87}$$

and the logistic sigmoid is just a special case of the softmax function that avoids using redundant parameters. We actually end up overparameterizing the softmax because the probabilities must add up to one.

Definition 4.6 (Softmax Regression Model)

The softmax regression of K classes assumes a model of the form

$$h_\theta(x) = o(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{88}$$

where $\mathbf{W} \in \mathbb{R}^{K \times D}, \mathbf{b} \in \mathbb{R}^D$. Again, we have a linear map followed by some link function (the softmax) which allows us to nonlinearly map our unbounded linear outputs to some domain that can be easily parameterized by a probability distribution. In this case, our residual distribution is a **multinomial distribution**

$$y \sim \text{Multinomial}\big(h_\mathbf{w}(\mathbf{x})\big) = \text{Multinomial}\big([h_\mathbf{w}(\mathbf{x})]_1, \ldots, [h_\mathbf{w}(\mathbf{x})]_K\big) \tag{89}$$

Definition 4.7 (Multiclass Cross Entropy Loss as Surrogate Loss for Softmax)

The surrogate loss for softmax regression is the **multiclass cross entropy loss**, which is defined as

$$L(\theta; \mathcal{D}) = -\sum_{i=1}^N \sum_{k=1}^K y_k^{(i)} \log \left(h_\theta(\mathbf{x}^{(i)}) \right)_k \tag{90}$$

Theorem 4.2 (Maximum Likelihood Estimation for Softmax)

Since a closed form solution is not available for logistic regression, it is clearly not available for softmax. Therefore, we one hot encode our target variables as $\mathbf{y}^{(i)}$ and write our likelihood as

$$L(\theta; \mathcal{D}) = \prod_{i=1}^N \prod_{k=1}^K p(C_k \mid \mathbf{x}^{(i)})^{\mathbf{y}_k^{(i)}} = \prod_{i=1}^N \prod_{k=1}^K \big(\mathbf{h}_{\mathbf{W}}(\mathbf{x}^{(i)})\big)_k^{\mathbf{y}_k^{(i)}} \tag{91}$$

Taking the negative logarithm gives us the **cross entropy** loss function

$$\ell(\boldsymbol{\theta}) = -\sum_{i=1}^N \sum_{k=1}^K y_k^{(i)} \log \big(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\big)_k = -\sum_{i=1}^N \mathbf{y}^{(i)} \dot{\log}\big(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}\big) \tag{92}$$

where $\cdot$ is the dot product. The gradient of this function may seem daunting, but it turns out to be very cute. Let us take a single sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, drop the index i, and write

$$\begin{aligned} \mathbf{x} &\mapsto \mathbf{W}\mathbf{x} + \mathbf{b} = \mathbf{z} \\ \hat{\mathbf{y}} = \mathbf{a} &= o(\mathbf{z}) \\ L &= -\mathbf{y} \cdot \log(\mathbf{a}) = -\sum_{k=1}^K y_k \log(a_k) \end{aligned}$$

We must compute

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \boldsymbol{\theta}} \tag{93}$$

We can compute $\partial L / \partial \mathbf{z}$ as such, using our derivations for the softmax derivative above. We compute element wise.

$$\begin{aligned} \frac{\partial L}{\partial z_j} &= -\sum_{k=1}^K y_k \frac{\partial}{\partial z_j} \log(a_k) \\ &= -\sum_{k=1}^K y_k \frac{1}{a_k} \frac{\partial a_k}{\partial z_j} \\ &= -\sum_{k=1}^K \frac{y_k}{a_k} a_k (1_{\{k=j\}} - a_j) \\ &= -\sum_{k=1}^K y_k (1_{\{k=j\}} - a_j) \\ &= \bigg(\sum_{k=1}^K y_k a_j \bigg) - y_j \\ &= a_j \bigg(\sum_{k=1}^K y_k \bigg) - y_j \\ &= a_j - y_j \end{aligned}$$

and combining these gives

$$\frac{\partial L}{\partial \mathbf{z}} = (\mathbf{a} - \mathbf{y})^T \tag{94}$$

Now, computing $\partial \mathbf{z} / \partial \mathbf{W}$ gives us a 3-tensor, which is not ideal to work with. However, let us just compute this with respect to the elements again. We have

$$z_k = \sum_{d=1}^{D} W_{kd} x_d + b_k$$
$$\frac{\partial z_k}{\partial W_{ij}} = \sum_{d=1}^{D} x_d \frac{\partial}{\partial W_{ij}} W_{kd}$$

where

$$\frac{\partial}{\partial W_{ij}} W_{kd} = \begin{cases} 1 & \text{if } i = k, j = d \\ 0 & \text{else} \end{cases} \tag{95}$$

Therefore, since d is iterating through all elements, the sum will only be nonzero if $k = i$. That is, $\frac{\partial z_k}{\partial W_{ij}} = x_j$ if $k = i$ and 0 if else. Therefore,

$$\frac{\partial \boldsymbol{z}}{\partial W_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_j \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{th element}$$

Now computing

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial W_{ij}} = (\mathbf{a} - \mathbf{y}) \frac{\partial \mathbf{z}}{\partial W_{ij}} = \sum_{k=1}^{K} (a_k - y_k) \frac{\partial z_k}{\partial W_{ij}} = (a_i - y_i) x_j \tag{96}$$

To get $\partial L / \partial W_{ij}$ we want a matrix whose entry (i, j) is $(a_i - y_i) x_j$. This is simply the outer product as shown below. For the bias term, $\partial \mathbf{z} / \partial \mathbf{b}$ is simply the identity matrix.

$$\frac{\partial L}{\partial \mathbf{W}} = (\mathbf{a} - \mathbf{y}) \mathbf{x}^T, \quad \frac{\partial L}{\partial \mathbf{b}} = \mathbf{a} - \mathbf{y} \tag{97}$$

Therefore, summing the gradient over some minibatch $M \subset [N]$ gives

$$\nabla_{\mathbf{W}} \ell_M = \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) (\mathbf{x}^{(i)})^T, \quad \nabla_{\mathbf{b}} \ell_M = \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \tag{98}$$

and our stochastic gradient descent algorithm is

$$\begin{aligned} \boldsymbol{\theta} = \begin{pmatrix} \mathbf{W} \\ \mathbf{b} \end{pmatrix} &= \begin{pmatrix} \mathbf{W} \\ \mathbf{b} \end{pmatrix} - \eta \begin{pmatrix} \nabla_{\mathbf{W}} \ell_M \\ \nabla_{\mathbf{b}} \ell_M \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{W} \\ \mathbf{b} \end{pmatrix} - \eta \begin{pmatrix} \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) (\mathbf{x}^{(i)})^T \\ \sum_{i \in M} (\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}) \end{pmatrix} \end{aligned}$$

4.3.1 Sparse Logistic Regression

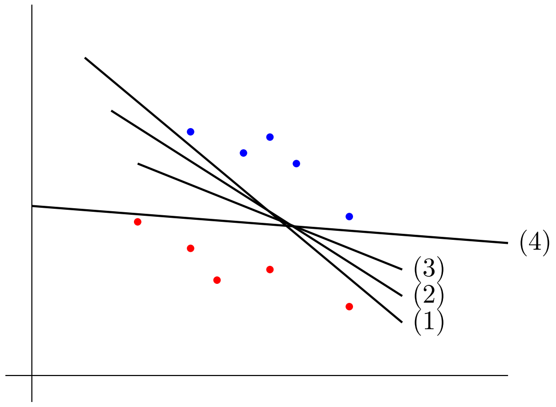
4.4 Support Vector Machines

Definition 4.8 (Hinge Loss)

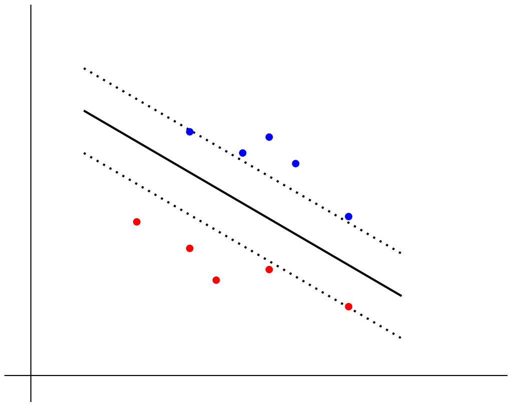
The **hinge loss** is a convex surrogate loss function for the 0-1 loss function. It is defined as

$$L(y, \hat{y}) = \max(0, 1 - y \cdot \hat{y}) \tag{99}$$

A support vector machine focuses only on the points that are most difficult to tell apart, whereas other classifiers pay attention all of the points. A SVM is a discriminative, non-probabilistic model. Let us first assume that our dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}$ is linearly separable with $y_i \in \{-1, +1\}$. Based on previous algorithms like the perceptron, it will find some separating hyperplane. However, there's an infinite number of separating hyperplanes as shown in Figure 11a. What support vector machines want to do is to find the best one, with the "best" defined as the hyperplane that maximizes the distance between either the closest positive or negative samples, shown in Figure 11b.



(a) Planes such as (1) and (4) are "too close" to the positive and negative samples.



(b) SVMs try to find the separating hyperplane with the best minimum margin.

Figure 11: Motivating problem

We want to formalize the concepts of these margins that we wish to maximize. To do this, we will define two terms.

Definition 4.9 (Geometric margin)

Given a point $\mathbf{x}_0$ and a hyperplane of equation $\mathbf{w} \cdot \mathbf{x} + b = 0$, the distance from $\mathbf{x}_0$ to the hyperplane, known as the **geometric margin**, can be computed with the formula

$$d = \frac{|\mathbf{x}_0 \cdot \mathbf{w} + b|}{||\mathbf{w}||} \tag{100}$$

Therefore, the geometric margin of the ith sample with respect to the hypothesis f is defined

$$\gamma_i = \frac{y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)}{||\mathbf{w}||} \tag{101}$$

We wish to optimize the parameters $\mathbf{w}, b$ in order to maximize the minimum of the geometric margins (the distance between the closest point and the hyperplane).

$$\underset{\mathbf{w},b}{\text{argmax}} \min_i \gamma_i = \underset{\mathbf{w},b}{\text{argmax}} \left\{ \frac{1}{||\mathbf{w}||} \min_i \big[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b \right) \big] \right\} \tag{102}$$

Direct solution of this optimization problem would be very complex, and so we convert this into an equivalent problem that is much easier to solve. Note that the solution to the above term is not unique. If there was a solution $(\mathbf{w}^*, b^*)$, then

$$\frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{||\mathbf{w}||} = \frac{y_i(\lambda \mathbf{w} \cdot \mathbf{x}_i + \lambda b)}{||\lambda \mathbf{w}||} \tag{103}$$

That is, the geometric margin is not sensitive to scaling of the parameters of the hyperplane. Therefore, we can scale the numerator and the denominator by whatever we want and use this freedom to set

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$$

for the point that is closest to the surface. In that case, all data points will satisfy the constraints

$$y_n(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

In the case of data points for which the equality holds, the constraints are said to be *active*, whereas for the remainder they are *inactive*. Therefore, it will always be the case that $\min_i \big[y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b \right) \big] = 1$, and the constraint problem reduces to

$$\underset{\mathbf{w},b}{\text{argmax}} \frac{1}{||\mathbf{w}||} = \underset{\mathbf{w},b}{\text{argmin}} \frac{1}{2} ||\mathbf{w}||^2 \text{ subject to constraints } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

This final step is the most significant step in this derivation and may be hard to wrap around the first time. So we dedicate the next subsubsection for this.

4.5 Functional and Geometric Margins

We could just work straight with this geometric margin, but for now, let's try to extend what we did with the perceptron into SVMs. We will find out that extending the concept of functional margins into SVMs leads to ill-defined problems. In the perceptron, we wanted to construct a function $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ such that

$$y_i \, f(\mathbf{x}_i) \geq 0 \text{ for all } i = 1, 2, \ldots, N$$

Definition 4.10 (Functional Margin)

The value of $y_i \, f(\mathbf{x}_i)$ gives us our confidence on our classification, and in a way it represents a kind of distance away from the separating hyperplane (if this value was 0, then we would be 50 50 split on whether to label it positive or negative). Therefore, we shall define

$$\hat{\gamma}_i = y_i f(\mathbf{x}_i)$$

as the **functional margin** of $(\mathbf{w}, b)$ with respect to the training sample $(\mathbf{x}_i, y_i)$. Therefore, the smallest of the function margins can be written

$$\hat{\gamma} = \min_i \gamma_i$$

called the **function margin**.

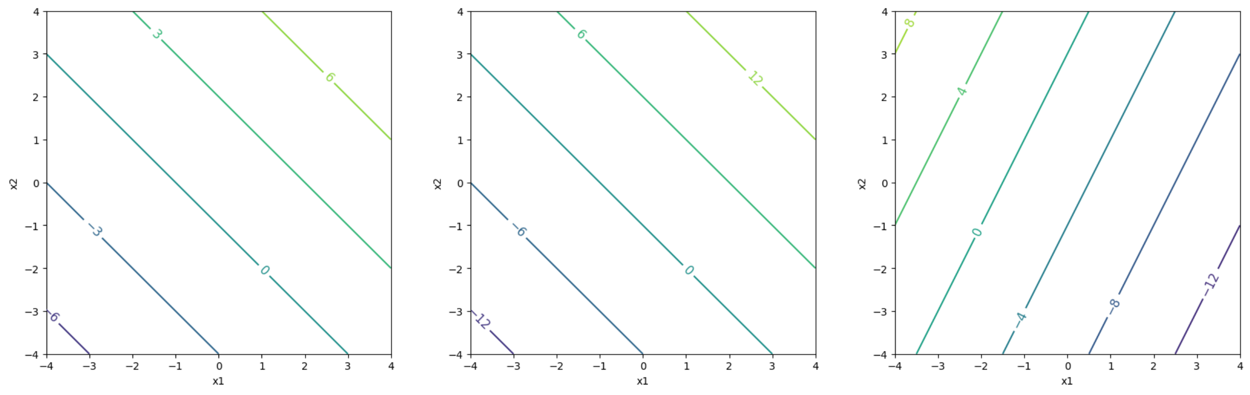
Note that the geometric margin and functional margin are related by a constant scaling factor. Given a sample $(\mathbf{x}_i, y_i)$, we have

$$\text{GeometricMargin} = \frac{y_i\left(\mathbf{w} \cdot \mathbf{x}_i + b\right)}{||\mathbf{w}||_2} = \frac{\text{FunctionalMargin}}{||\mathbf{w}||_2}$$

As we can see, the perceptron works with the functional margin, and since it does not care about how large the margin is (just whether it's positive or negative), we are left with an underdetermined system in which there exists infinite $(\mathbf{w}, b)$'s. Now what we want to do is impose a certain minimum margin $\gamma > 0$ and solve for $(\mathbf{w}, b)$ again, and keep increasing this γ until there is some unique solution. We can view this problem in two ways:

1. Take a specific minimum margin γ and find a $(\mathbf{w}, b)$, which may not exist, be unique, or exist infinitely that satisfies
$$y_i f(\mathbf{x}) = y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq \gamma \text{ for all } i = 1, \ldots, N$$
2. Take a specific $(\mathbf{w}, b)$ and calculate the maximum γ that satisfies the constraint equations above.

They're both equivalent problems, but both ill-posed if we look at (2). Since the samples are linearly separable by assumption, we can say that there exists some $\epsilon > 0$ such that $y_i f(\mathbf{x}_i) \geq \epsilon$ for all i. Therefore, if we just scale $(\mathbf{w}, b) \mapsto (\lambda\mathbf{w}, \lambda b)$ for some large λ, this leads to the solution for γ being unbounded. We can see in Figure 12 that we can increased confidence at no cost. Looking at (1), we can also see that if $(\mathbf{w}, b)$ does exist, then every other $(\lambda\mathbf{w}, \lambda b)$ for $\lambda > 1$ satisfies the property.



(a) $f(x) = x_1 + x_2 + 1$ (b) $f(x) = 2x_1 + 2x_2 + 2$ (c) $f(x) = -2x_1 + x_2 - 3$

Figure 12: From (a), you can see that simply multiplying everything by two automatically increases our confidence by 2, meaning that the functional margin can be scaled arbitrarily by scaing $(\mathbf{w}, b)$. There are still too many degrees of freedom in here and so extra constraints must be imposed.

4.5.1 Lagrange Duality

To minimize the equations with the constraint equations, we can use the method of Lagrange multipliers, which leads to to Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_i \alpha_i \big[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1\big]$$

We can take the gradients with respect to $\mathbf{w}$ and b and set them to 0, which gives the two conditions

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$0 = \sum_i \alpha_i y_i \mathbf{x}_i$$

Now let's substitute our evaluated $\mathbf{w}$ back into $\mathcal{L}$, which gives the **dual representation** of the maximum margin problem in which we maximize

$$\begin{aligned} L &= \frac{1}{2}\bigg(\sum_i \alpha_i y_i \mathbf{x}_i\bigg)\bigg(\sum_j \alpha_j y_j \mathbf{x}_j\bigg) - \sum_i \alpha_i y_i x_i \cdot \bigg[\sum_j \alpha_j y_j x_j\bigg] - \sum_i \alpha_i y_i b + \sum_i \alpha_i \\ &= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \, \mathbf{x}_i \cdot \mathbf{x}_j \end{aligned}$$

The summation with the b in it is 0 since we can pull the b out and the remaining sum is 0 from before. Now the optimization only depends on the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$ of all pairs of sample vectors, which is very interesting. We will see more of this when we talk about kernel methods. Now, we need to solve the dual problem

$$\max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha})$$

which can be done using some generic quadratic programming solver or some other method to get the optimum $\boldsymbol{\alpha}^*$, which gives us

$$\mathbf{w}^* = \sum_i \alpha_i^* y_i \mathbf{x}_i$$

4.6 Nonseparable Case

4.7 Gaussian/Linear Discriminant Analysis

4.7.1 Discriminative vs. Generative Models

Now we introduce our first example of a generative model, which introduces another division between models (in addition to parametric vs nonparametric, frequentist vs bayesian). Generally, there are two ways to model $\mathbb{P}_{Y|X=x}$.

Definition 4.11 (Discriminative Models)

Discriminative models attempt to do this directly by modeling only the conditional probability distribution $\mathbb{P}_{Y|X=x}$. We don't care about the underlying distribution of X, but rather we just want to try and predict Y given X. Essentially, we are trying to approximate the conditional expectation $h(X) = \mathbb{E}[Y \mid X]$, which is the best we can do. Given $X = x$, we use our model of $\mathbb{P}_{Y|X=x}$, and our hypothesis function will predict the its mean.

$$h(x) = \mathbb{E}[Y \mid X = x] \tag{104}$$

Definition 4.12 (Generative Models)

Generative models approximate this conditional probability by taking a more general approach. They attempt to model the joint probability distribution $\mathbb{P}_{X\times Y}$ (also called **inference**), which essentially gives everything we need about the data. Doing this allows us to *generate* more data (hence the name), which may be useful.
One way to approximate the joint distribution is to model the conditional distribution $\mathbb{P}_{X|Y=y}$, which gives the distribution of each labels. Now combined with the probability measure $\mathbb{P}_Y$, we can get the joint distribution. Usually in classification, the $\mathbb{P}_Y$ is easy to approximate (the MLE is simply the fequencies of the labels), so conventionally, modeling $\mathbb{P}_{X\times Y}$ and modeling $\mathbb{P}_{X|Y=y}$ are considered the same thing. Once we have these, we can calculate the joint distribution, but in high-dimensional spaces this tends to be computationally hard. Therefore, we usually resort to simply calculating $\mathbb{P}_{X|Y=y}$ and then using Bayes rule to approximate

$$\mathbb{P}_{Y|X} = \frac{\mathbb{P}_{X|Y}\mathbb{P}_Y}{\mathbb{P}_X} \tag{105}$$

where the normalizing term is computed using Monte Carlo simulations.

This is the first example of a generative model. In GDA, we basically write the likelihood as

$$\prod_{i=1}^n p(x_i, y_i) = \prod_i p(x_i \mid y_i) p(y_i) \tag{106}$$

where each $p(x_i \mid y_i)$ is Gaussian and $p(y_i)$ is Bernoulli. This specifies $p(x_i, y_i)$ and therefore is called a generative model. In logistic regression, we have

$$\prod_{i=1} p(x_i, y_i) = \bigg(\prod_i p(y_i \mid x_i) \bigg) \bigg(\prod_i p(x_i) \bigg) \tag{107}$$

and the first term is the logistic function and the second term is unknown. We only use the first part to classify, and this is a discriminative model. You can be agnostic about the data generating process and you can work with less data since there are less things to fit. Some people ask why should you model more unless you have to, so people tend to try to model the minimum, which is why logistic regression is more popular.

4.7.2 Construction

GDA assumes that $\mathbb{P}(x\,|y)$ is distributed according to a multivariate Gaussian distribution. Let us assume that the input space is d-dimensional and this is a binary classification problem. We set

$$\begin{aligned} y &\sim \text{Bernoulli}(\pi) \\ x\,|\,y = 0 &\sim \mathcal{N}_d(\mu_0, \Sigma) \\ x\,|\,y = 1 &\sim \mathcal{N}_d(\mu_1, \Sigma) \end{aligned}$$

This method is usually applied using only one covariance matrix Σ. The distributions are

$$\begin{aligned} p(y) &= \pi^y (1-\pi)^{1-y} \\ p(x\,|\,y=0) &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\bigg(-\frac{1}{2} (x-\mu_0)^T \Sigma^{-1} (x - \mu_0) \bigg) \\ p(x\,|\,y=1) &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\bigg(-\frac{1}{2} (x-\mu_1)^T \Sigma^{-1} (x - \mu_1) \bigg) \end{aligned}$$

Now, what we have to do is optimize the distribution parameters $\pi \in (0,1)\mathbb{R}, \mu_0 \in \mathbb{R}^d, \mu_1 \in \mathbb{R}^d, \Sigma \in \text{Mat}(d \times d, \mathbb{R}) \simeq \mathbb{R}^{d \times d}$ so that we get the best-fit model. Assuming that each sample has been picked independently, this is equal to maximizing

$$L(\pi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^n \mathbb{P}\big(x^{(i)}, y^{(i)}\,;\, \pi, \mu_0, \mu_1, \Sigma\big) \tag{108}$$

which is really just the probability that we get precisely all these training samples $(x^{(i)}, y^{(i)})$ given the 4 parameters. This can be done by optimizing its log-likelihood, which is given by

$$\begin{aligned} l(\pi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^n \mathbb{P}(x^{(i)}, y^{(i)}; \pi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^n \mathbb{P}(x^{(i)}\,|\,y^{(i)}; \mu_0, \mu_1, \Sigma)\, \mathbb{P}(y^{(i)}; \pi) \\ &= \sum_{i=1}^n \log \bigg(\mathbb{P}(x^{(i)}\,|\,y^{(i)}; \mu_0, \mu_1, \Sigma)\, \mathbb{P}(y^{(i)}; \pi) \bigg) \end{aligned}$$

and therefore gives the maximum likelihood estimate to be

$$\begin{aligned}
\pi &= \frac{1}{N} \sum_{n=1}^N 1\{y^{(n)} = 1\} \\
\boldsymbol{\mu}_0 &= \frac{\sum_{n=1}^n 1_{\{y^{(n)}=0\}} \mathbf{x}^{(n)}}{\sum_{n=1}^N 1_{\{y^{(n)}=0\}}} \\
\boldsymbol{\mu}_1 &= \frac{\sum_{n=1}^n 1_{\{y^{(n)}=1\}} \mathbf{x}^{(n)}}{\sum_{n=1}^N 1_{\{y^{(n)}=1\}}} \\
\boldsymbol{\Sigma} &= \frac{1}{N} \sum_{n=1}^N (\mathbf{x}^{(n)} - \mu_{y^{(n)}})(\mathbf{x}^{(n)} - \mu_{Y^{(i)}})^T
\end{aligned}$$

A visual of the algorithm is below, with contours of the two Gaussian distributions, along with the straight line giving the decision boundary at which $\mathbb{P}(y = 1 \,|\, x) = 0.5$.

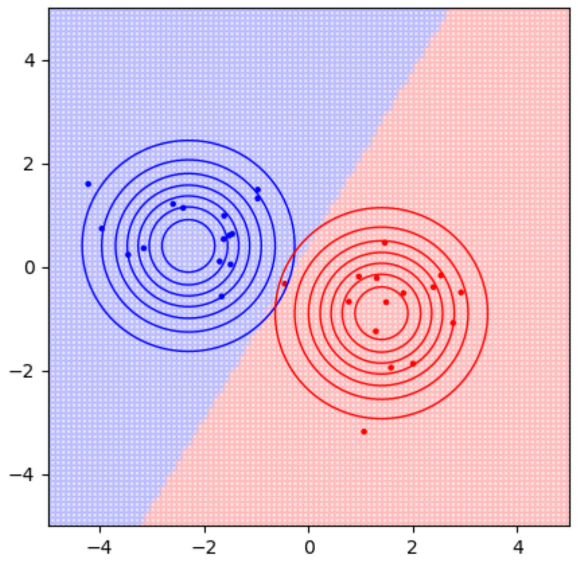


Figure 13: GDA of Data Generated from 2 Gaussisans centered at $(-2.3, 0.4)$ and $(1.4, -0.9)$ with unit covariance. The decision boundary is slightly off since MLE approximates the true means.

4.8 Fisher Linear Discriminant

5 Nonparametric Classification

5.1 K Nearest Neighbors

Question 5.1 (To Do)

Maybe similar like a kernel regression?

Given a bunch of points in a metric space $(\mathcal{X}, d)$ that have classification labels, we want to label new datapoints $\hat{\mathbf{x}}$ based on the labels of other points that already exist in our dataset. One way to look at it is to look for close points within the dataset and use their labels to predict the new ones.

Definition 5.1 (Closest Neighborhood)

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}$ and a point $\hat{\mathbf{x}} \in (\mathcal{X}, d)$, let the **k closest neighborhood** of $\hat{\mathbf{x}}$ be $N_k(\hat{\mathbf{x}}) \subset [N]$ defined as the indices i of the k points in $\mathcal{D}$ that is closest to $\hat{\mathbf{x}}$ with respect to the distance metric $d_{\mathcal{X}}$.

Definition 5.2 (K Nearest Neighbors)

The **K Nearest Neighbors (KNN)** is a discriminative nonparametric supervised learning algorithm that doesn't have a training phase. Given a new point $\hat{\mathbf{x}}$, we look at all points in its k closest neighborhood, and $h(\hat{\mathbf{x}})$ will be equal to whatever the majority class will be in. Let us one-hot encode the labels $\mathbf{y}^{(i)}$ into $\mathbf{e}_i$'s, and the number of data point in the ith class can be stored in the variable

$$a_i = \sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_i\}} \tag{109}$$

which results in the vector storing the counts of labels in the k closest neighborhood

$$\mathbf{a} = (a_1, a_2, \ldots, a_{\mathcal{K}}) = \left(\sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_1\}}, \sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_2\}}, \ldots, \sum_{i \in N_k(\hat{\mathbf{x}})} 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_{\mathcal{K}}\}} \right) \tag{110}$$

and take the class with the maximum element as our predicted label.

The best choice of K depends on the data:

1. Larger values of K reduces the effect of noise on the classification, but make boundaries between classes less distinct. The number of misclassified data points (error) increases.
2. Smaller values are more sensitive to noise, but boundaries are more distinct and the number of misclassified data points (error) decreases.

Too large of a K value may increase the error too much and lead to less distinction in classification, while too small of a k value may result in us overclassifying the data. Finally, in binary (two class) classification problems, it is helpful to choose K to be odd to avoid tied votes.

This is an extremely simple algorithm that may not be robust. For example, consider $K \geq 3$, and we are trying to label a point $\hat{\mathbf{x}}$ that happens to be exactly where one point is on our dataset $\mathbf{x}^{(i)}$. Then, we should do $h(\hat{\mathbf{x}}) = y^{(i)}$, but this may not be the case if there are no other points with class $y^{(i)}$ in the k closest neighborhood of $\mathbf{x}^{(i)}$. Therefore, we want to take into account the distance of our new points from the others.

Definition 5.3 (Weighted Nearest Neighbor Classifier)

Let us define a monotinically decreasing function $\omega : \mathbb{R}_0^+ \mapsto \mathbb{R}_0^+$. Given a point $i \in N_k(\hat{\mathbf{x}})$, we can construct the weight of our matching label as inversely proportional to the distance: $\omega_i[d(\hat{\mathbf{x}}, \mathbf{x}^{(i)})]$ and store them as

$$\mathbf{a} = (a_1, a_2, \ldots, a_{\mathcal{K}}) = \bigg(\sum_{i \in N_k(\hat{\mathbf{x}})} \omega_i 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_1\}}, \sum_{i \in N_k(\hat{\mathbf{x}})} \omega_i 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_2\}}, \ldots, \sum_{i \in N_k(\hat{\mathbf{x}})} \omega_i 1_{\{\mathbf{y}^{(i)} = \mathbf{e}_{\mathcal{K}}\}} \bigg) \tag{111}$$

and again take the class with the maximum element.

One caveat of KNN is in high dimensional spaces, as its performance degrades quite badly due to the curse of dimensionality.

Example 5.1 (Curse of Dimensionality in KNN)

Consider a dataset of N samples uniformly distributed in a d-dimensional hypercube. Now given a point $x \in [0,1]^d$, we want to derive the expected radius r_k required to encompass its k nearest neighbors. Let us define this ball to be $B_{r_k} \coloneqq \{z \in \mathbb{R}^d \mid ||z - x||_2 \leq r_k\}$. Since thse N points are uniformly distributed, the expected number of points contained in $B_{r_k}(x)$ is simply the proportion of the volume that $B_{r_k}(x)$ encapsulates in the box, multiplied by N. Therefore, for some fixed x and r, let us denote $Y(x, y)$ as the random variable representing the number of points contained within $B_r(x)$. By linearity of expectation and summing over the expectation for whether each point will be in the ball, we have

$$\mathbb{E}[Y(x, r)] = N \cdot \frac{\mu(B_r(x) \cap [0,1]^d)}{\mu([0,1]^d)}$$

where μ is the Lebesgue measure of $\mathbb{R}^d$. Let us assume for not that we don't need to worry about cases where the ball is not fully contained within the cube, so we can just assume that Y is only dependent on r: $Y(r)$. Also, since the volume of the hypercube is 1, $\mu([0,1]^d) = 1$ and we get

$$\mathbb{E}[Y(r)] = N \cdot C_d \cdot r^d$$

which we set equal to k and evaluate for r. C_d is a constant such that the volume of the hypersphere of radius r can be derived as $V = C_d \cdot r^d$. We therefore get

$$N \cdot C_d \cdot r_k^d = k \implies r_k = \left(\frac{k}{NC_d} \right)^{1/d}$$

It turns out that C_d decreases exponentially, so the radius r_k explodes as d grows. Another way of looking at this is that in high dimensions, the ℓ_2 distance between all the pairwise points are close in every single dimension, so it becomes harder to distinguish points that are close vs those that are far.

5.1.1 Approximate K Nearest Neighbors

5.2 Classification Trees

Definition 5.4 (Decision Trees)

Like K nearest neighbors, **decision trees** are discriminative nonparametric classification algorithms that involves creating some sort of tree that represents a set of decisions using a given set of input data $\mathbf{x}^{(i)}$ with its given classification $\mathbf{y}^{(i)}$. When predicting the class of a new input $\hat{\mathbf{x}}$, we would look at its attributes in some order, e.g. $\hat{x_1}, \hat{x}_2, \hat{x}_3$, and make a decision on which class it is in.

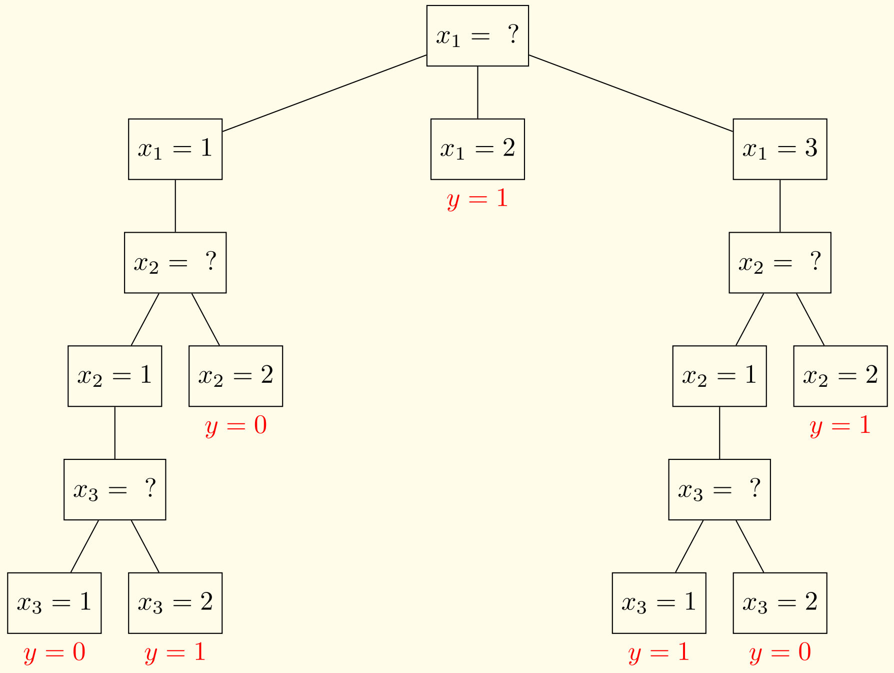


Figure 14: An example of a decision tree that splits at x_1 first, then x_2, and finally x_3. Note that you can still split on x_2 if $x_1 = 1$ and x_3 if $x_1 = 3$.

The decision tree tries to take advantage of some nontrivial covariance between X and Y by constructing nested partitions of the dataset $\mathcal{D}$, and within a partition, it predicts the label that comprises the majority.

For now, let us assume that $\mathcal{X}$ is a Cartesian product of discrete sets, and we will extend them to continuous values later. Let us look at an example to gain some intuition.

Example 5.2 (Restaurant Dataset)

Consider the following dataset.

	OthOptions	Weekend	WaitArea	Plans	Price	Precip	Restaur	Wait	Crowded	Stay?
x_1	Yes	No	No	Yes	$$$	No	Mateo	0-5	some	Yes
x_2	Yes	No	No	Yes	$	No	Juju	16-30	full	No
x_3	No	No	Yes	No	$	No	Pizza	0-5	some	Yes
x_4	Yes	Yes	No	Yes	$	No	Juju	6-15	full	Yes
x_5	Yes	Yes	No	No	$$$	No	Mateo	30+	full	No
x_6	No	No	Yes	Yes	$$	Yes	BlueCorn	0-5	some	Yes
x_7	No	No	Yes	No	$	Yes	Pizza	0-5	none	No
x_8	No	No	No	Yes	$$	Yes	Juju	0-5	some	Yes
x_9	No	Yes	Yes	No	$	Yes	Pizza	30+	full	No
x_{10}	Yes	Yes	Yes	Yes	$$$	No	BlueCorn	6-15	full	No
x_{11}	No	No	No	No	$	No	Juju	0-5	none	No
x_{12}	Yes	Yes	Yes	Yes	$	No	Pizza	16-30	full	Yes

Table 1: Dataset of whether to go to a restaurant for a date depending on certain factors.

Let us denote $\mathcal{D}$ as the dataset, and say that $F_1, \ldots, F_d$ were the features. This is a binary classification problem, and we can count that there are 6 positives and 6 negative labels.

The simplest decision tree is the trivial tree, with one node that predicts the majority of the dataset. In this case, the data is evenly split, so without loss of generality we will choose $h_0(\mathbf{x}) = 1$. We want to quantify how good our model is, and so like always we use a loss function.

Just like how a linear model is completely defined by its parameter $\boldsymbol{\theta}$, a decision tree is completely defined by the sequences of labels that it splits on. Therefore, training this is equivalent to defining the sequence, but we can't define this sequence unless we can compare how good a given decision tree is, i.e. unless we have defined a proper loss function. Depending on the training, we can use a greedy algorithm or not, and we have the flexibility to choose whether or not we can split on the same feature multiple times.

Definition 5.5 (Misclassification Error)

We will simply use the misclassification loss function.

$$L(h; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^N 1_{\{y^{(i)} \neq h(x^{(i)})\}} = 1 - \text{accuracy} \tag{112}$$

Minimizing this maximizes the accuracy, so this is a reasonable one to choose. How do we train this? Unlike regression, this loss is not continuous, so the gradient is 0, and furthermore the model isn't even parametric, so there are no gradients to derive!

Fortunately, the nature of the decision tree only requires us to look through the explanatory variables $x_1, \ldots, x_n$ and decide which one to split.

Let us take a decision tree h and model the accuracy of it as a random variable: $1_{\{Y=h_0(X)\}} \sim \text{Bernoulli}(p)$, where p is the accuracy. A higher accuracy of h corresponds to a lower entropy, and so the entropy of the random variable is also a relevant indicator.

$$H(1_{\{Y=h_0(X)\}}) = p \log p + (1-p) \log(1-p)$$

Therefore, when we are building a tree, we want to choose the feature x_i to split based on how much it lowers the entropy of the decision tree.

To set this up, let us take our dataset $\mathcal{D}$ and set X_i as the random variable representing the distribution (a multinomial) of the $x_i^{(j)}$'s, and Y as the same for the $y^{(j)}$'s. This is our maximum likelihood approximation for the marginalized distribution of the joint measure $X \times Y = X_1 \times \ldots \times X_D \times Y$.

Given a single node, we are simply going to label every point to be whatever the majority class is in $\mathcal{D}$. Therefore, we start off with the entropy of our trivial tree $H(Y)$. Then, we want to see which one of the X_d features to split on, and so we can compute the conditional entropy $H(Y, X_d)$ to get the information gain $I(Y; X_d) = H(Y) - H(Y \mid X_d)$ for all $d = 1, \ldots, D$. We want to find a feature X_d that maximize this information gain, i.e. decreases the entropy as much as possible (a greedy algorithm), and we find the next best feature (with or without replacement), so that we have a decreasing sequence.

$$H(X) \geq H(X; Y) \geq H(X; Y, Z) \geq H(X; Y, Z, W) \geq \ldots \geq 0$$

Example 5.3 (Crowded Restaurants)

Continuing the example above, since there are 6 labels of 0 and 1 each, we can model this $Y \sim \text{Bernoulli}(0.5)$ random variable, with entropy

$$H(Y) = \mathbb{E}[-\log_2 p(Y)] = \frac{1}{2}\Big(-\log_2 \frac{1}{2}\Big) + \frac{1}{2}\Big(-\log_2 \frac{1}{2}\Big) = 1$$

Now what would happen if we had branched according to how crowded it was, X_{crowded}. Then, our decision tree would split into 3 sections:

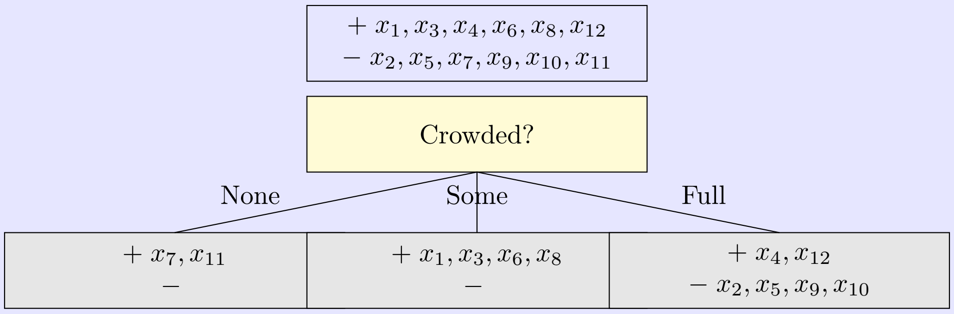

$+ x_1, x_3, x_4, x_6, x_8, x_{12}$
$- x_2, x_5, x_7, x_9, x_{10}, x_{11}$
Crowded?
None
Some
Full
$+ x_7, x_{11}$
$-$
$+ x_1, x_3, x_6, x_8$
$-$
$+ x_4, x_{12}$
$- x_2, x_5, x_9, x_{10}$

Figure 15: Visual of decision tree splitting according to how crowded it is.

In this case, we can define the multinomial distribution X_{crowded} representing the proportion of the data that is crowded in a specific level. That is, $X_{\text{crowded}} \sim \text{Multinomial}(\frac{2}{12}, \frac{4}{12}, \frac{6}{12})$, with

$$\mathbb{P}(X_{\text{crowded}} = x) = \begin{cases} 2/12 & \text{if } x = \text{ none} \\ 4/12 & \text{if } x = \text{ some} \\ 6/12 & \text{if } x = \text{ full} \end{cases} \tag{113}$$

Therefore, we can now compute the conditional entropy of this new decision tree conditioned on how crowded the store is

$$H(Y \mid X_{\text{crowded}}) = \sum_x \mathbb{P}(X_{\text{crowded}} = x) H(Y \mid X_{\text{crowded}} = x) \tag{114}$$

$$= \frac{2}{12} H(\text{Bern}(1)) + \frac{4}{12} H(\text{Bern}(0)) + \frac{6}{12} H(\text{Bern}(1/3)) = 0.459 \tag{115}$$

$$I(Y; X_{\text{crowded}}) = 0.541 \tag{116}$$

We would do this for all the features and greedily choose the feature that maximizes our information gain.

Example 5.4 (Ferrari F1 Race)

The Ferrari F1 team hired you as a new analyst! You were given the following table of the past race history of the team. You were asked to use information gain to build a decision tree to predict race wins. First, you will need to figure out which feature to split first.

Rain	Good Strategy	Qualifying	Win Race
1	0	0	0
1	0	0	0
1	0	1	0
0	0	1	1
0	0	0	0
0	1	1	1
1	0	1	0
0	1	0	1
0	0	1	1
0	0	1	1

Let $X \sim \text{Bernoulli}(1/2)$ be the distribution of whether a car wins a race over the data. Then its entropy is

$$H(X) = \mathbb{E}[-\log_2 p(x)] = \frac{1}{2}\Big(-\log_2 \frac{1}{2}\Big) + \frac{1}{2}\Big(-\log_2 \frac{1}{2}\Big) = 1$$

Let $R \sim \text{Bernoulli}(4/10), G \sim \text{Bernoulli}(2/10), Q \sim \text{Bernoulli}(6/10)$ be the distribution of the features rain, good strategy, and qualifying over the data, respectively. Then, the conditional entropy of X conditioned on each of these random variables is

$$\begin{aligned}
H(X \mid R) &= \mathbb{P}(R=1)\, H(X \mid R=1) + \mathbb{P}(R=0)\, H(X \mid R=0) \\
&= \frac{4}{10} \cdot -\big(1 \cdot \log_2 1 + 0 \cdot \log_2 0\big) + \frac{6}{10} \cdot -\big(\frac{1}{6} \cdot \log_2 \frac{1}{6} + \frac{5}{6} \cdot \log_2 \frac{5}{6}\big) \approx 0.390 \\
H(X \mid G) &= \mathbb{P}(G=1)\, H(X \mid G=1) + \mathbb{P}(G=0)\, H(X \mid G=0) \\
&= \frac{2}{10} \cdot -\big(1 \cdot \log_2 1 + 0 \cdot \log_2 0\big) + \frac{8}{10} \cdot -\big(\frac{3}{8} \cdot \log_2 \frac{3}{8} + \frac{5}{8} \log_2 \frac{5}{8}\big) \approx 0.763 \\
H(X \mid Q) &= \mathbb{P}(Q=1)\, H(X \mid Q=1) + \mathbb{P}(Q=0)\, H(X \mid Q=0) \\
&= \frac{6}{10} \cdot -\big(\frac{4}{6} \cdot \log_2 \frac{4}{6} + \frac{2}{6} \cdot \log_2 \frac{2}{6}\big) + \frac{4}{10} \cdot -\big(\frac{1}{4} \log_2 \frac{1}{4} + \frac{3}{4} \log_2 \frac{3}{4}\big) \approx 0.875
\end{aligned}$$

Therefore, the information gain are

$$\begin{aligned}
I(X; R) &= 1 - 0.390 = 0.610 \\
I(X; G) &= 1 - 0.763 = 0.237 \\
I(X; Q) &= 1 - 0.875 = 0.125
\end{aligned}$$

And so I would split on R, the rain, which gives the biggest information gain.

Finally, we can use the Gini index of $X \sim \text{Bernoulli}(p)$, defined

$$G(X) = 2p(1-p) \tag{117}$$

Example 5.5 (Ferrari Example Continued)

We do the same as the Ferrari example above but now with the Gini reduction. Let $X \sim \text{Bernoulli}(1/2)$ be the distribution of whether a car wins a race over the data. Then its Gini index, which I will label with $\mathcal{G}$, is

$$\mathcal{G}(X) = 2 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$$

Let $R \sim \text{Bernoulli}(4/10), G \sim \text{Bernoulli}(2/10), Q \sim \text{Bernoulli}(6/10)$ be the distribution of the features rain, good strategy, and qualifying over the data, respectively. Then we compute the conditional expectation

$$\begin{aligned}
\mathbb{E}[\mathcal{G}(X \mid R)] &= \mathbb{P}(R=1)\, \mathcal{G}(X \mid R=1) + \mathbb{P}(R=0)\, \mathcal{G}(X \mid R=0) \\
&= \frac{4}{10}\left[2 \cdot \frac{4}{4} \cdot \frac{0}{4}\right] + \frac{6}{10}\left[2 \cdot \frac{1}{6} \cdot \frac{5}{6}\right] \approx 0.167 \\
\mathbb{E}[\mathcal{G}(X \mid G)] &= \mathbb{P}(G=1)\, \mathcal{G}(X \mid G=1) + \mathbb{P}(G=0)\, \mathcal{G}(X \mid G=0) \\
&= \frac{2}{10}\left[2 \cdot \frac{2}{2} \cdot \frac{0}{2}\right] + \frac{8}{10}\left[2 \cdot \frac{3}{8} \cdot \frac{5}{8}\right] \approx 0.375 \\
\mathbb{E}[\mathcal{G}(X \mid Q)] &= \mathbb{P}(Q=1)\, \mathcal{G}(X \mid Q=1) + \mathbb{P}(Q=0)\, \mathcal{G}(X \mid Q=0) \\
&= \frac{6}{10}\left[2 \cdot \frac{4}{6} \cdot \frac{2}{6}\right] + \frac{4}{10}\left[2 \cdot \frac{1}{4} \cdot \frac{3}{4}\right] \approx 0.417
\end{aligned}$$

Therefore, the Gini reduction, which I'll denote as $I_{\mathcal{G}}$, is

$$\begin{aligned}
I_{\mathcal{G}}(X; R) &= 0.5 - 0.167 = 0.333 \\
I_{\mathcal{G}}(X; G) &= 0.5 - 0.375 = 0.125 \\
I_{\mathcal{G}}(X; Q) &= 0.5 - 0.417 = 0.083
\end{aligned}$$

Since branching across the feature R, the rain, gives the biggest Gini reduction, we want to split on the rain feature first.

5.2.1 Regularization

Given a dataset with D binary features, let $g(H, D)$ be the number of binary trees with depth at most H (including root node), with the restriction that the trees may not split on some variable multiple times within a path to a leaf node. Then, g can be defined recursively.

1. First, if $H = 1$, then $g(H, D) = 1$ always since we are just creating the trivial binary tree of one node.
2. If $D = 0$, then there are no features to split on and therefore we just have the single node $g(H, D) = 1$.
3. If $H > 1$ and $D > 0$, then say that we start with a node. We can either make this a leaf node by not performing any splitting at all, or split on one of the D variables. Then for each of the 2 nodes created on the split, we are now working with $D - 1$ features and a maximum height of $H - 1$ for each of the subtrees generated from the 2 nodes.

All this can be expressed as

$$g(H, D) = \begin{cases} 1 + D\left[g(H-1, D-1)\right]^2 & \text{if } H > 1, D > 0 \\ 1 & \text{if } H = 1 \text{ or } D = 0 \end{cases}$$

which is extremely large (in fact, NP hard). Therefore, some tricks like regularization must be implemented to limit our search space.

By defining the complexity of our decision tree $\Omega(h)$ as the number of nodes within the tree, we can modify our objective function to

$$L(h; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} 1_{\{y^{(i)} \neq h(x^{(i)})\}} + \lambda \Omega(h)$$

We can impose this constraint directly on the training algorithm, or we can calculate the regularized loss after the tree has been constructed, which is a method called **tree pruning**.

Given a large enough λ, we can in fact greatly reduce our search space by not considering any trees further than a certain point.

Theorem 5.1 ()

We describe a tree as a set of leaves, where leaf k is a tuple containing the logical preposition satisfied by the path to leaf k, denoted p_k, and the class label predicted by the leaf, denoted $\hat{y}_k$. For a dataset with d binary features, $p_k : \{0, 1\}^d \to \{0, 1\}$ is a function that returns 1 if a sample x_i satisfies the preposition, and 0 otherwise. That is, leaf k is $(p_k, \hat{y}_k)$, and a tree f with K leaves is described as a set $f = \{(p_1, \hat{y}_1), \ldots, (p_K, \hat{y}_K)\}$. Assume that the label predicted by $\hat{y}_k$ is always the label for the majority of samples satisfying p_k. Finally, let $m_k = \sum_{i=1}^{n} p_k(x_i)$ denote the number of training samples "captured" by leaf k.
Given a (potentially optimal) tree

$$f = \{(p_1, \hat{y}_1), \ldots, (p_\kappa, \hat{y}_\kappa), \ldots, (p_K, \hat{y}_K)\},$$

the tree $f' = \{(p_1, \hat{y}_1), \ldots, (p_{\kappa_1}, \hat{y}_{\kappa_1}), (p_{\kappa_2}, \hat{y}_{\kappa_2}), \ldots, (p_K, \hat{y}_K)\}$ produced by splitting leaf $(p_\kappa, \hat{y}_\kappa)$ into two leaves $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ and $(p_{\kappa_2}, \hat{y}_{\kappa_2})$ and any tree produced by further splitting $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ or $(p_{\kappa_2}, \hat{y}_{\kappa_2})$ cannot be optimal if $m_\kappa < 2n\lambda$.

Proof.

Let c be the number of misclassifications in leaf $(p_\kappa, \hat{y}_\kappa)$. Since a leaf classifies according to the majority of m_κ, we must have

$$c \leq \frac{m_\kappa}{2} < n\lambda$$

By splitting leaf $(p_\kappa, \hat{y}_\kappa)$ into leaves $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ and $(p_{\kappa_2}, \hat{y}_{\kappa_2})$, assume that we have reduced the number of misclassifications by $b \leq c$. Then, we have

$$\ell(f', \mathbf{X}, \mathbf{y}) = \ell(f, \mathbf{X}, \mathbf{y}) - \frac{b}{n}$$

However, we have increased the number of leaves by 1, and so

$$\lambda s(f') = \lambda s(f) + \lambda$$

Combining the last two equations, we have obtained

$$R(f', \mathbf{X}, \mathbf{y}) = R(f, \mathbf{X}, \mathbf{y}) + \lambda - \frac{b}{n}$$

However, we know that

$$\begin{aligned} b \leq c &\implies \frac{b}{n} \leq \frac{c}{n} < \frac{n\lambda}{n} = \lambda \\ &\implies -\frac{b}{n} > -\lambda \\ &\implies \lambda - \frac{b}{n} > \lambda - \lambda = 0 \end{aligned}$$

and so $R(f', \mathbf{X}, \mathbf{y}) > R(f, \mathbf{X}, \mathbf{y})$. This means that f' cannot be optimal according to our regularized objective. We have also proved that further splitting $(p_{\kappa_1}, \hat{y}_{\kappa_1})$ or $(p_{\kappa_2}, \hat{y}_{\kappa_2})$ cannot be optimal since we can just set $f = f'$, and apply the same argument.

6 Generalized Linear Models

Remember the linear model looked like this, where we use the conventional β notation to represent parameters.

$$Y = X^T \beta + \epsilon, \ \ \epsilon \sim N(0, \sigma^2 I) \tag{118}$$

which implies that $Y \mid X \sim N(X^T\beta, \sigma^2 I)$. Basically, given x, I assume some distribution of Y, and the value of x will help me guess what the mean of this distribution is. Note that we in here assume that only the mean depends on X. I could potentially have something crazy, like

$$Y \mid X \sim N(X^T\beta, (X^T\gamma)(XX^T + I))$$

where the covariance will depend on X, too, but in this case we only assume that that mean is dependent on X.

$$Y \mid X \sim N(\mu(X), \sigma^2 I)$$

where in the linear model, $\mu(X) = X^T\beta$. So, there are three assumptions we are making here:

1. $Y \mid X$ is Gaussian.
2. X only affects the mean of $Y \mid X$, written $\mathbb{E}[Y \mid X] = \mu(X)$.
3. X affects the mean in a linear way, such that $\mu(X) = X^T\beta$.

So the two things we are trying to relax are:

1. **Random Component**: the response variable $Y \mid X$ is continuous and normally distributed with mean $\mu = \mu(X) = \mathbb{E}[Y \mid X]$.
2. **Link**: I have a link that explains the relationship between the X and the μ, and this relationship is $\mu(X) = X^T\beta$.

So when talking about GLMs, we are not changing the fact that we have a linear function $X \mapsto X^T\beta$. However, we are going to assume that $Y \mid X$ now comes from a broader **family of exponential distributions**. Second, we are going to assume that there exists some **link function** g

$$g(\mu(X)) = X^T\beta$$

Admittedly, this is not the most intuitive way to think about it, since we would like to have $\mu(X) = f(X^T\beta)$, but here we just decide to call $f = g^{-1}$. Therefore, if I want to give you a GLM, I just need to give you two things: the conditional distribution $Y \mid X$, which can be any distribution in the exponential family, and the link function g.

We really only need this link function due to compatibility reasons. Say that $Y \mid X \sim \text{Bern}(p)$. Then, $\mu(X) = \mathbb{E}[Y \mid X]$ always lives in $[0, 1]$, but $X^T\beta$ always lives in $\mathbb{R}$. We want our model to be realistic, and we can clearly see the problem shown in Figure 16.

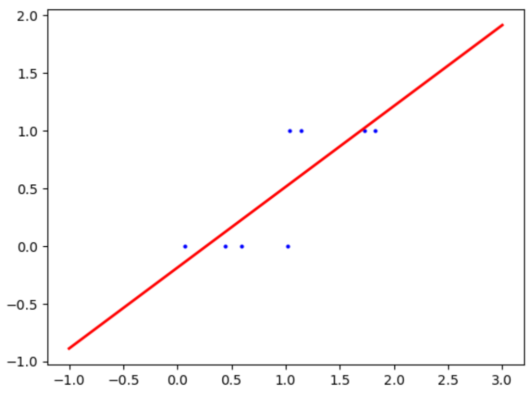


Figure 16: Fitting a linear model for Bernoulli random variables will predict a mean that is outside of $[0, 1]$ when getting new datapoints.

If $Y \mid X$ is some exponential distribution, then its support is always positive and so $\mu(X) > 0$. But if we stick to the old form of $\mu(X) = X^T\beta$, then $\text{Im}(\mu) = \mathbb{R}$, which is not realistic when we predict negative values. Let's take a couple examples:

Example 6.1 (Disease Epidemic)

In the early stages of a disease epidemic, the rate at which new cases occur can often increase exponentially through time. Clearly, $\mu(X) = \mathbb{E}[Y \mid X]$ should be positive and we should have some sort of exponential trend. Hence, if $\mu(x)$ is the expected number of cases on data x, a model of the form

$$\mu(x) = \gamma \exp(\delta x) \tag{119}$$

seems appropriate, where γ and δ are simply scaling factors. Clearly, $\mu(X)$ is not of the form $f(X^T\beta)$. So what I do is to transform μ in such a way that I can get something that is linear.

$$\log(\mu(X)) = \log(\gamma) + \delta X \tag{120}$$

which is now linear in X, of form $\beta_0 + \beta_1 X$. This will have some effects, but this is what needs to be done to have a genearlized linear model. Note that what I did to μ was take the log of it, and so the link function is $g = \log$, called the **log-link**. Now that we have chosen the g, we still need to choose what the conditional distribution $Y \mid X$ would be. This is determined by speaking with industry professionals, experience, and convenience. In this case, Y is a count, and since this must be a discrete distribution. Since it is not bounded above, we think Poisson.

Example 6.2 (Prey Capture Rate)

The rate of capture of preys, Y, by a hunting animal, tends to increase with increasing density of prey X, but eventually level off when the predator is catching as much as it can cope with. We want to find a perhaps concave function that levels off, and suitable model might be

$$\mu(X) = \frac{\alpha X}{h + X} \tag{121}$$

where α represents the maximum capture rate, and h represents the prey density at which the capture rate is half the maximum rate. Again, we must find some transformation g that turns this into a linear function of X, and what we can do it use the **reciprocal-link**.

$$\frac{1}{\mu(X)} = \frac{h+X}{\alpha X} = \frac{h}{\alpha}\frac{1}{X} + \frac{1}{\alpha} \tag{122}$$

The standard deviation of capture rate might be approximately proportional to the mean rate, suggesting the use of a Gamma distribution for the response.

Example 6.3 (Kyphosis Data)

The Kyphosis data consist of measurements on 81 children following corrective spinal surgery. The binary response variable, Kyphosis, indicates the presence or absence of a postoperative deforming. The three covariates are: age of the child in months, number of the vertebrae involved in the operation, and the start of the range of the vertebrae involved. The response variable is binary so there is no choice: $Y \mid X$ is Bernoulli with expected value $\mu(X) \in (0,1)$. We cannot write $\mu(X) = X^T\beta$ because the right hand side ranges through $\mathbb{R}$, and so we find an invertible function that squishes $\mathbb{R}$ to $(0,1)$, and so we can choose basically any CDF.

For clarification, when writing a distribution like Bernoulli(p), or Binomial(n,p), Poisson(λ), or $N(\mu,\sigma^2)$, the hyperparameters that we usually work with we will denote as $\boldsymbol{\theta}$, and the space that this $\boldsymbol{\theta}$ lives in will denote Θ. For example, for the Bernoulli, $\Theta = [0,1]$, and for Poisson, $\Theta = [0,+\infty)$.

Ultimately, a GLM consists of three steps:

1. The observed input X enters the model through a linear function $\beta^T X$.
2. The conditional mean of response, is represented as a function of the linear combination
$$\mathbb{E}[Y \mid X] = \mu = f(\beta^T X) \tag{123}$$
3. The observed response is drawn from an exponential family distribution with conditional mean μ.

6.1 Exponential Family

We can write the pdf of a distribution as a function of the input x and the hyperparameters θ, so we can write $P_\theta(x) = p(\theta, x)$. For now, let's think that both $x, \theta \in \mathbb{R}$. Think of all the functions that depend on θ and x. There are many of them, but we want θ and x to interact in a certain way. The way that I want them to interact with each other is that they are multiplied within an exponential term. Now clearly, this is not a very rich family, so we are just slapping some terms that depend only on θ and only on x.

$$p_\theta(x) = \exp(\theta x) h(x) c(\theta)$$

But now if $\theta \in \mathbb{R}^k$ and $x \in \mathbb{R}^q$, then we cannot simply take the product nor the inner product, but what we can do is map both of them into a space that has the same dimensions, so I can take the inner product. That is, let us map $\boldsymbol{\theta} \mapsto \boldsymbol{\eta}(\boldsymbol{\theta}) \in \mathbb{R}^k$ and $\mathbf{x} \mapsto \mathbf{T}(\mathbf{x}) \in \mathbb{R}^k$, and so our exponential distribution form would be generalized into something like

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp\big[\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \mathbf{T}(\mathbf{x})\big] h(\mathbf{x}) c(\boldsymbol{\theta})$$

We can think of $c(\boldsymbol{\theta})$ as the normalizing term that allows us to integrate the pdf to 1.

$$\int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathbf{x}) = c(\boldsymbol{\theta}) \int \exp\big[\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \mathbf{T}(\mathbf{x})\big] h(\mathbf{x})\, d\mathbf{x}$$

We can just push the $c(\boldsymbol{\theta})$ term into the exponential by letting $c(\boldsymbol{\theta}) = e^{-\log(c(\boldsymbol{\theta}))^{-1}}$ to get our definition.

Definition 6.1 (Exponential Family)

A **k-parameter exponential family** is a family of distributions with pdf/pmf of the form

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp\big[\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \mathbf{T}(\mathbf{x}) - B(\boldsymbol{\theta})\big] h(\mathbf{x})$$

The h term, as we will see, will not matter in our maximum likelihood estimation, so we keep it outside the exponential.

1. $\boldsymbol{\eta}$ is called the **canonical parameter**. Given a distribution parameterized by the regular hyperparameters $\boldsymbol{\theta}$, we would like to parameterize it in a different way $\boldsymbol{\eta}$ under the function $\boldsymbol{\eta} : \Theta \to \mathbb{R}$
2. $\mathbf{T}(\mathbf{x})$ is called the **sufficient statistic**.
3. $h(\mathbf{x})$ is a nonnegative scalar function.
4. $B(\boldsymbol{\theta})$ is the normalizing factor.

Let's look at some examples.

Example 6.4 (Gaussian)

If we put the coefficient into the exponential and expand the square term, we get

$$p_\theta(x) = \exp\left(\frac{\mu}{\sigma^2} \cdot x - \frac{1}{2\sigma^2} \cdot x^2 - \frac{\mu^2}{2\sigma^2} - \log(\sigma\sqrt{2\pi})\right)$$

where

$$\boldsymbol{\eta}(\boldsymbol{\theta}) = \begin{pmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{pmatrix},\ T(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix},\ B(\theta) = \frac{\mu^2}{2\sigma^2} + \log(\sigma\sqrt{2\pi}),\ h(x) = 1$$

This is not a unique representation since we can take the $\log(\sqrt{2\pi})$ out of the exponential, but why bother to do this when we can just stuff everything into B and keep h simple.

Example 6.5 (Gaussian with Known Variance)

If we have known variance, we can write the Gaussian pdf as

$$p_\theta(x) = \exp\left[\frac{\mu}{\sigma} \cdot \frac{x}{\sigma} - \frac{\mu^2}{2\sigma^2}\right] \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

where

$$\eta(\theta) = \frac{\mu}{\sigma},\ T(x) = \frac{x}{\sigma},\ B(\theta) = \frac{\mu^2}{2\sigma^2},\ h(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

Example 6.6 (Bernoulli)

The pmf of a Bernoulli with θ is

$$\begin{aligned} p_\theta(x) &= \theta^x (1-\theta)^{(1-x)} \\ &= \exp\big[x\log(\theta) + (1-x)\log(1-\theta)\big] \\ &= \exp\left(x \log\left[\frac{\theta}{1-\theta}\right] - \log\left[\frac{1}{1-\theta}\right]\right) \end{aligned}$$

where

$$\eta(\theta) = \log\left[\frac{\theta}{1-\theta}\right],\ T(x) = x,\ B(\theta) = \log\left[\frac{1}{1-\theta}\right],\ h(x) = 1$$

Example 6.7 (Binomial with Known Number of Trials)

We can transform a binomial with known N as

$$\begin{aligned} p_\theta(x) &= \binom{N}{x} \theta^x (1-\theta)^{1-x} \\ &= \exp\Big[x \ln\Big(\frac{\theta}{1-\theta}\Big) + \ln(1-\theta)\Big] \cdot \binom{N}{x} \end{aligned}$$

where

$$\eta(\theta) = \ln\Big(\frac{\theta}{1-\theta}\Big),\ T(x) = x,\ B(\theta) = \ln(1-\theta),\ h(x) = \binom{N}{x}$$

Example 6.8 (Poisson)

The pmf of Poisson with θ can be expanded

$$\begin{aligned} p_\theta &= \frac{\theta^{-x}}{x!} e^{-\theta} \\ &= \exp\big[-\theta + x\log(\theta) - \log(x!)\big] \\ &= \exp\big[x\log(\theta) - \theta\big] \frac{1}{x!} \end{aligned}$$

where

$$\eta(\theta) = \log(\theta),\ T(x) = x,\ B(\theta) = \theta,\ h(x) = \frac{1}{x!}$$

However, the uniform is not in here. In fact, any distribution that has a support that does not depend on the parameter is not an exponential distribution.

Let us now focus on one parameter families where $\theta \in \Theta \subset \mathbb{R}$, which do not include the Gaussian (with unknown mean and variance, Gamma, multinomial, etc.), which has a pdf written in the form

$$p_\theta(x) = \exp\big[\eta(\theta)\, T(x) - B(\theta)\big]\, h(x)$$

6.1.1 Canonical Exponential Family

Now a common strategy in statistical analysis is to reparamaterize a probability distribution. Suppose a family of probability distributions $\{P_\theta\}$ is parameterized by $\theta \in \Theta \subset \mathbb{R}$. If we have an invertible function $\eta : \Theta \to \mathcal{T} \subset \mathbb{R}$, then we can paramaterize the same family with η rather than θ, with no loss of information. Typically, it is the case that η is invertible for exponential families, so we can just reparameterize the whole pdf and write

$$p_\eta(x) = \exp\big[\eta\, T(x) - \phi(\eta)\big]\, h(x)$$

where $\phi = B \circ \eta^{-1}$.

Definition 6.2 (Canonical One-Parameter Exponential Family)

A family of distributions is said to be in **canonical one-parameter exponential family** if its density is of form

$$p_\eta(x) = \exp\big[\eta\, T(x) - \phi(\eta)\big]\, h(x)$$

which is a subfamily of the exponential family. The function ψ is called the **cumulant generating function**.

Before we move on, let us just provide a few examples.

Example 6.9 (Poisson)

The Poisson can be represented as

$$p_\theta(x) = \exp\left[x \log \theta - \theta\right] \frac{1}{x!}$$

Now let $\eta = \log \theta \implies \theta = e^\eta$. So, we can reparamaterize the density as

$$p_\eta(x) = \exp\left[x\eta - e^\eta\right] \frac{1}{x!}$$

where $P_\eta = \text{Poisson}(e^\eta)$ for $\eta \in \mathcal{T} = \mathbb{R}$, compared to $P_\theta = \text{Poisson}(\theta)$ for $\theta \in \Theta = \mathbb{R}^+$.

Example 6.10 (Gaussian)

Recall that the Gaussian with known parameter σ^2 and unknown $\theta = \mu$ is in the exponential family, since we can expand it as

$$p_\theta(x) = \exp\left[\frac{\mu}{\sigma^2} \cdot x - \frac{\mu^2}{2\sigma^2}\right] \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

We can perform the change of parameter $\eta = \mu^2/2\sigma^2 \implies \mu = \sigma^2\eta$, and substituting this in will give the canonical representation

$$p_\eta(x) = \exp\left[\eta x - \frac{\sigma^2\eta^2}{2}\right] \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

where now $P_\eta = N(\sigma^2\eta, \sigma^2)$ for $\eta \in \mathcal{T} = \mathbb{R}$, compared to $P_\theta = N(\theta, \sigma^2)$ for $\theta \in \Theta = \mathbb{R}$, which is basically the same space.

Example 6.11 (Bernoulli)

The Bernoulli has an exponential form of

$$p_\theta(x) = \exp\left[x \log\left(\frac{\theta}{1-\theta}\right) + \log(1-\theta)\right]$$

Now setting $\eta = \log\left(\frac{\theta}{1-\theta}\right) \implies \theta = \frac{1}{1+e^{-\eta}}$, and so $B(\theta) = -\log(1-\theta) = -\log\left(\frac{e^{-\eta}}{1+e^{-\eta}}\right) = \log(1+e^\eta) = \psi(\eta)$, and so the canonical paramaterization is

$$p_\eta(x) = \exp\left[x\eta - \log(1+e^\eta)\right]$$

We present two useful properties of the exponential family.

Theorem 6.1 (Moments)

Let random variable X be in the canonical exponential family P_η

$$p_\eta(x) = e^{\eta T(x) - \psi(\eta)} h(x)$$

Then, the expectation and variance are encoded in the cumulant generating function in the following way

$$\mathbb{E}[T(X)] = \psi'(\eta) \qquad \text{Var}[T(X)] = \psi''(\eta)$$

Proof.

Example 6.12 ()

We show that this is consistent with the Poisson, normal, and Bernoulli distributions.

1. In the Poisson, $\psi(\eta) = e^\eta$, and so $\psi'(\eta) = e^\eta = \theta = \mathbb{E}[X]$. Taking the second derivative gives $\psi''(\eta) = e^\eta = \theta = \mathrm{Var}[X]$, too.
2. In the Normal with known variance σ^2, we have $\psi(\eta) = \frac{1}{2}\sigma^2\eta^2$. So
$$\begin{aligned} \mathbb{E}[X] &= \psi'(\eta) = \sigma^2\eta = \mu \\ \mathrm{Var}[X] &= \psi''(\eta) = \sigma^2 \end{aligned}$$
3. In the Bernoulli, we have $\psi(\eta) = \log(1 + e^{-\eta})$. Therefore,
$$\begin{aligned} \mathbb{E}[X] &= \psi'(\eta) = \frac{x^\eta}{1 + x^\eta} = \frac{1}{1 + e^{-\eta}} = \theta \\ \mathrm{Var}[X] &= \psi''(\eta) = -\left(\frac{1}{1 + e^{-\eta}}\right)^2 e^{-\eta} \cdot -1 = \theta^2 \cdot \frac{1 - \theta}{\theta} = \theta(1 - \theta) \end{aligned}$$

Theorem 6.2 (Convexity)

Consider a canonical exponential family with density
$$p_\eta(x) = e^{\eta T(x) - \psi(\eta)} h(x)$$
and natural parameter space $\mathcal{T}$. Then, the set $\mathcal{T}$ is convex, and the cumulant generating function ψ is convex on $\mathcal{T}$.

Proof.

This can be proven using Holder's inequality. However, from the theorem above, note that $\mathrm{Var}[T(X)] = \psi''(\eta)$ must be positive since we are talking about variance. This implies that the second derivative of ψ is positive, and therefore must be convex.

We will look at a subfamily of the exponential family. Now remember that we introduce the functions $\boldsymbol{\eta}$ and $\mathbf{T}$ so that we can capture a much broader range of distributions, but if we have one parameter $k = 1$, then we can just set $\boldsymbol{\eta}(\boldsymbol{\theta})$ to be the new parameter θ. The **canonical exponential family** for $k = 1, y \in \mathbb{R}$, is defined to have the pdf
$$f_\theta(y) = \exp\left(\frac{y\theta - b(\theta)}{\phi} + c(y, \phi)\right) \tag{124}$$
where
$$h(y) = \exp\big(c(y, \phi)\big) \tag{125}$$
If ϕ is known, this is a one-parameter exponential family with θ being the **canonical parameter**, and if ϕ is unknown, the $h(y)$ term will not depend on θ, which we may not be able to split up into the exponential pdf form. In this case ϕ is called the **dispersion parameter**. For now, we will always assume that ϕ is known.

We can prove this for all other classes, too. We can think of the $c(y, \phi)$ as just a term that we stuff every other term into. What really differentiates the different distributions of the canonical exponential family is the $b(\theta)$. The form of b will determine whether this distribution is a Gaussian, or a Bernoulli, or etc. This b will capture information about the mean, the variance, the likelihood, about everything.

6.2 Cumulant Generating Function

Definition 6.3 (Score)

The **score** is the gradient of the log-likelihood function with respect to the parameter vector. That is, given that $L(\boldsymbol{\theta})$ is the likelihood, then

$$s(\boldsymbol{\theta}) \coloneqq \frac{\partial \log L(\boldsymbol{\theta}; \mathbf{x})}{\partial \boldsymbol{\theta}}$$

which gives a row covector.

Now, remember that the score also depends on the observations $\mathbf{x}$. If we rewrite the likelihood as a probability density function $L(\boldsymbol{\theta}; \mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta})$, then we can say that the expected value of the score is equal to 0, since

$$\begin{aligned}
\mathbb{E}[s(\boldsymbol{\theta})] &= \int_{\mathcal{X}} f(\mathbf{x}; \boldsymbol{\theta}) \frac{\partial}{\partial \boldsymbol{\theta}} \log L(\boldsymbol{\theta}; \mathbf{x}) \, d\mathbf{x} \\
&= \int_{\mathcal{X}} f(\mathbf{x}; \boldsymbol{\theta}) \frac{1}{f(\mathbf{x}; \boldsymbol{\theta})} \frac{\partial f(\mathbf{x}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \, dx \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} \int_{\mathcal{X}} f(\mathbf{x}; \boldsymbol{\theta}) \, d\mathbf{x} \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} 1 = 0
\end{aligned}$$

where we take a leap of faith in switching the derivative and integral in the penultimate line. Furthermore, we can get the second identity

$$\mathbb{E}\left[\frac{\partial^2 \ell}{\partial \theta^2}\right] + \mathbb{E}\left[\frac{\partial \ell}{\partial \theta}\right]^2 = 0$$

We can apply these two identities as follows. Since

$$\ell(\theta) = \frac{Y\theta - b(\theta)}{\phi} + c(Y; \phi)$$

therefore

$$\frac{\partial \ell}{\partial \theta} = \frac{Y - b'(\theta)}{\phi}$$

which yields

$$0 = \mathbb{E}\left[\frac{\partial \ell}{\partial \theta}\right] = \frac{\mathbb{E}[Y] - b'(\theta)}{\phi} \implies \mathbb{E}[Y] = \mu = b'(\theta)$$

On the other hand, we have

$$\frac{\partial^2 \ell}{\partial \theta^2} + \left(\frac{\partial \ell}{\partial \theta}\right)^2 = -\frac{b''(\theta)}{\phi} + \left(\frac{Y - b'(\theta)}{\phi}\right)^2$$

and from the previous result, we get

$$\frac{Y - b'(\theta)}{\phi} = \frac{Y - \mathbb{E}[Y]}{\phi}$$

together with the second identity, yields

$$0 = -\frac{b''(\theta)}{\phi} + \frac{\mathrm{Var}(Y)}{\phi^2} \implies \mathrm{Var}(Y) = \phi''(\theta)$$

Since variance is always positive, this implies that $b'' > 0$ and therefore b must be convex.

6.3 Link Functions

Now let's go back to GLMs. In linear models, we said that the conditional expectation of Y given $X = \mathbf{x}$ must be a linear function in x

$$\mathbb{E}[Y \mid X = \mathbf{x}] = \mu(\mathbf{x}) = \mathbf{x}^T \beta$$

But if the conditional distribution takes values in some subset of $\mathbb{R}$, such as $(0,1)$, then it may not make sense to write this as a linear function, since $X^T\beta$ has an image spanning $\mathbb{R}$. So what we need is a link function that relates, i.e. transforms the restricted subset of μ, onto the real line, so that now you can express it of the form $X^T\beta$.

$$g\big(\mu(X)\big) = X^T\beta$$

Again, it is a bit more intuitive to talk about g^{-1}, which takes our $X^T\beta$ and transforms it to the values that I want, so we will talk about both of them simultaneously. If g is our link function, we want it to satisfy 3 requirements:

1. g is continuously differentiable
2. g is strictly increasing
3. $\text{Im}(g) = \mathbb{R}$, i.e. it spans the entire real line

This implies that g^{-1} exists, which is also continuously differentable and is strictly increasing.

Example 6.13 ()

If I have a conditional distribution...

1. that is Poisson, then we want our μ to be positive, and so we need a link function $g : \mathbb{R}^+ \to \mathbb{R}$. One choice would be the logarithm

$$g(\mu(X)) = \log\big(\mu(X)\big) = X^T\beta$$

2. that is Bernoulli, then we want our μ to be in $(0,1)$ and we need a link function $g : (0,1) \to \mathbb{R}$. There are 2 natural choices, which may be the **logit** function

$$g(\mu(X)) = \log\left(\frac{\mu(X)}{1-\mu(X)}\right) = X^T\beta$$

or the **probit** function

$$g(\mu(X)) = \Phi^{-1}\big(\mu(X)\big) = X^T\beta$$

where Φ is the CDF of a standard Gaussian. The two functions can be seen in Figure 17.

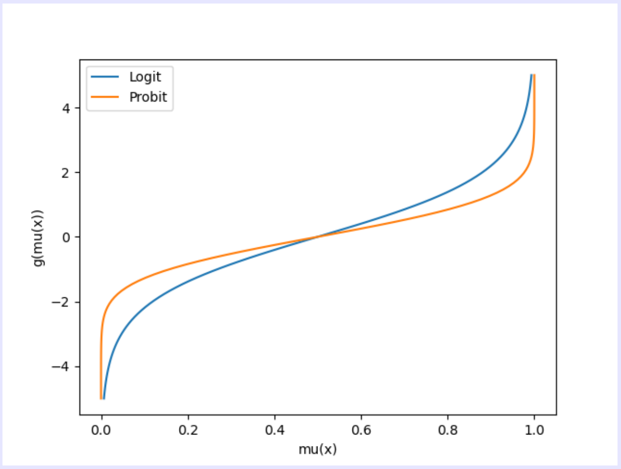


Figure 17: Logit and Probit Functions

Now there are many choices of functions we can take. In fact, if μ lives in $(0, 1)$, then we can really just take our favorite distribution that has a density that is supported everywhere in $\mathbb{R}$ and take the inverse cdf as our link. So far, we have no reason to prefer one function to another, but in the next section, we will see that there are more natural choices.

6.3.1 Canonical Link Functions

Now let's summarize what we have. We assume that the conditional distribution $Y \mid X = x$ follows a distribution in the exponential family, which we can completely characterize by the cumulant generating function ψ. For different values of x, the conditional distribution will be paramaterized by different $\eta(x)$, and the resulting distribution P_η will have some mean $\mu(x)$, which is usually not the natural parameter η. Now, let's forget about our knowledge that $\psi'(\eta) = \mu$, but we know that there is some relationship between $\eta \leftrightarrow \mu$.

Given an x, I need to use the linear predictor $x^T\beta$ to predict $\mu(x)$, which can be done through the link function g.

$$g\big(\mu(x)\big) = x^T\beta$$

Now what would be a natural way of choosing this g? Note that our natural parameter η for this canonical family takes value on the entire real line. I must construct a function g that maps μ onto the entire real line, and so why not make it map to η. Therefore, we have

$$\eta(x) = g\big(\mu(x)\big) = x^T\beta$$

Definition 6.4 (Canonical Link)

The function g that links the mean μ to the canonical parameter θ is called the **canonical link**.

$$g(\mu) = \theta$$

Now using our knowledge that $\psi'(\eta) = \mu$, we can see that

$$g = (\psi')^{-1}$$

This is indeed a valid link function.

1. $\psi'' > 0$ since it models the variance, and so ψ' is strictly increasing and so $g = (\psi')^{-1}$ is also strictly increasing.
2. The domain of ψ' is the real line since it takes in the natural parameter η which exists over $\mathbb{R}$, so $\text{Im}(g) = \mathbb{R}$.

So, given our cumulant generating function ψ and our link function g, both satisfying

$$\psi'(\eta) = \mu \text{ and } g(\mu) = x^T\beta$$

we can combine them to get

$$(g \circ \psi')(\eta) = g(\mu) = x^T\beta$$

and so, even though the mean of the response variable is not linear with respect to x, the value of $(g \circ \psi')(\eta)$ is indeed linear. In fact, if we choose the canonical link, then the equation

$$\eta = x^T\beta$$

means that the natural parameter of our conditional distribution in the exponential family is linear with respect to x! From this we can find the conditional mean $\mu(x)$.

The reason we focus on canonical link functions is because, when the canonical link is used, the components of the model (the parameters of the linear predictor) have an additive effect on the response variable in the transformed (linked) scale, which makes the interpretation of the results easier. It's also worth noting that while using the canonical link function has some desirable properties, it is not always the best or only choice, and other link functions may be used if they provide a better fit for the data or make more sense in the context of the problem at hand.

Let us evaluate some canonical link functions.

Example 6.14 ()

The Bernoulli has the canonical exponential form of

$$p_\eta(x) = \exp\big[x\eta - \log(1+e^\eta)\big]$$

where $\eta = \log\left(\frac{\theta}{1-\theta}\right)$. Since we have prior knowledge that $\theta = \mu$ (i.e. the expectation of a Bernoulli is the hyperparameter θ itself), we have a function that maps $\mu \mapsto \eta$.

$$\eta = g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$$

which gives us our result. We can also take the inverse of $\psi' = \frac{e^\eta}{1+e^\eta}$ to get our result

$$g(\mu) = (\psi')^{-1}(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$$

6.4 Likelihood Optimization

Now let us have a bunch of data points $\{(x_n, y_n)\}_{n=1}^N$. By our model assumption, we know that the conditional distribution $Y \mid X = x_n$ is now of an exponential family with parameter $\eta_n = \eta(x_n)$ and density

$$p_{\eta_n}(y_n) = \exp\big[y_n\eta_n - \psi(\eta_n)\big]h(y_n)$$

Now we want to do likelihood optimization on β (not η or μ), and to do this, we must rewrite the density function in a way so that it depends on β. Given a link function g, note the following relationship between β and η:

$$\begin{aligned}\eta_n = \eta(x_n) &= (\psi')^{-1}(\mu(x_n)) \\ &= (\psi')^{-1}\big(g^{-1}(x_n^T\beta)\big) \\ &= h(x_n^T\beta)\end{aligned}$$

where for shorthand notation, we define $h := (g \circ \psi')^{-1}$. Subtituting this into the above likelihood, taking the product of all N samples, and logarithming the equation gives us the following log likelihood to optimize over β.

$$\ell(\beta) = \log\prod_{n=1}^N p_{\eta_n}(y_n) = \sum_{n=1}^N y_n h(x_n^T\beta) - \psi(h(x_n^T\beta))$$

where we dropped the $h(y_n)$ term at the end since it is a constant and does not matter. If g was the canonical link, then h is the identity, and we should have a linear relationship between $\eta(x_n) = x_n^T\beta$. This means that the η_n reduces only to $x_n^T\beta$, which is much more simple to optimize.

$$\ell(\beta) = \log\prod_{n=1}^N p_{\eta_n}(y_n) = \sum_{n=1}^N y_n x_n^T\beta - \psi(x_n^T\beta)$$

Note that the first term is linear w.r.t β, and ψ is convex, so the entire sum must be concave w.r.t. β. With this, we can bring in some tools of convex optimization to solve.

7 Cross Validation

We have understood the theoretical foundations of overfitting and underfitting with the bias variance decomposition. But in practice, we don't have an ensemble of datasets; we just have one. Therefore, we don't actually know what the bias, the variance, or the noise is at all. Therefore, how do we actually *know* in practice when we are underfitting or overfitting? Easy. We just split our dataset into 2 different parts: the training set and testing sets.

$$\mathcal{D} = \mathcal{D}_{train} \sqcup \mathcal{D}_{test} \tag{126}$$

What we usually have is a **training set** that allows us to train the model, and then to check its performance we have a **test set**. We would train the model on the training set, where we will always minimize the loss, and then we would look at the loss on the test set. Though we haven't made a testing set, since we know the true model let us just generate more data and use that as our testing set. For each model, we can calculate the optimal $\boldsymbol{\theta}$, which we will denote $\boldsymbol{\theta}^*$, according to the **root mean squared loss**

$$h_{\boldsymbol{\theta}^*} = \operatorname*{argmin}_{h_{\boldsymbol{\theta}}} \sqrt{\frac{1}{N} \sum_{i=1}^N \big(y^{(i)} - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\big)^2} \tag{127}$$

where division of N allows us to compare different sizes of datasets on equal footing, and the square root ensures that this is scaled correctly. Let us see how well these different order models perform on a separate set of data generated by the same function with Gaussian noise.

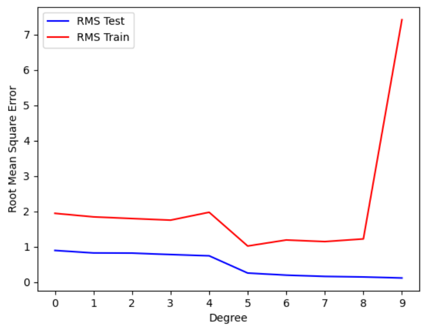


Figure 18: We can see that the RMS decreases monotonically on the training error as more complex functions become more fine-tuned to the data. However, when we have a 9th degree polynomial the RMS for the testing set dramatically increases, meaning that this model does not predict the testing set well, and performance drops.

Now we know that a more complex model (i.e. that captures a greater set of functions) is not necessarily the best due to overfitting. Therefore, researchers perform **cross-validation** by taking the training set $(\mathcal{X}, \mathcal{Y})$. We divide it into S equal pieces

$$\bigcup_{s=1}^S D_s = (\mathcal{X}, \mathcal{Y}) \tag{128}$$

Then, we train the model $\mathcal{M}$ on $S-1$ pieces of the data and then test it across the final piece, and do this S times for every test piece, averaging its perforance across all S test runs. Therefore, for every model $\mathcal{M}_k$, we must train it S times, for all K models, requiring KS training runs. If data is particularly scarce, we set $S = N$, called the **leave-one-out** technique. Then we just choose the model with the best average test performance.

The following result shows that cross-validation (data splitting) leads to an estimator with risk nearly as good as the best model in the class.

Theorem 7.1 (Gyorfi, Kohler, Krzyak, Walk (2002))

Let $\mathcal{M} = \{m_h\}$ be a finite class of regression estimators indexed by a parameter h, with m being the true risk minimizer, $m_{\hat{h}}$ being the empirical risk minimizer over the whole dataset $\mathcal{D}$, and m_H being the empirical risk minimizer over the test set $\mathcal{D}_{\text{test}}$ for ordinary least squares loss.

$$m_H = \operatorname*{argmin}_{m_h} \frac{1}{N} \sum_{i \in \mathcal{D}_{\text{test}}} (y_i - m_h(x_i))^2 \tag{129}$$

$$m_{\hat{h}} = \operatorname*{argmin}_{m_h} \frac{1}{N} \sum_{i \in \mathcal{D}} (y_i - m_h(x_i))^2 \tag{130}$$

If the data Y_i and estimators are bounded by L, then for any $\delta > 0$, we have

$$\mathbb{E} \int |m_H(x) - m(x)|^2 \, d\mathbb{P}(x) \leq (1+\delta)\mathbb{E} \int |m_{\hat{h}}(x) - m(x)|^2 \, d\mathbb{P}(x) + \frac{C(1 + \log |M|)}{n} \tag{131}$$

where $c = L^2(16/\delta + 35 + 19\delta)$.

Code 7.1 (Minimal Example of Train Test Split in scikit-learn)

To implement this in scikit-learn, we want to use the `train_test_split` class. We can also set a random state parameter to reproduce results.

```python
from sklearn.model_selection import train_test_split

# Split into training (80\%) and test (20\%) data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
    random_state=66)
```

However, this process requires a lot of training runs and therefore may be computationally infeasible. Therefore, various **information criterion** has been proposed to efficiently select a model.

7.1 Leave 1 Out Cross Validation

7.1.1 Generalized (Approximate) Cross Validation

7.1.2 Cp Statistic

7.2 K Fold Cross Validation

7.3 Data Leakage

7.4 Information Criterion

8 Practical Methodology

8.1 Model Selection

We've talked about the theory and implementation behind all these models, but in practice, how do we even use them? If we are trying to predict lung cancer in a patient, do we use linear regression, a nonparametric model, or something else? It's not clear at all what to do with the data. Unfortunately, this just comes with domain expertise and experience with data, but we can provide some general pointers.

As stated before, we have the flexibility to choose whatever model to train on. So how do we choose which form is the best? Well this is just an assumption that most researchers make, and this is called **model selection**.

Example 8.1 (Polynomial Regression)

The number of terms M, i.e. the degree $M-1$ of the polynomial

$$h_{\boldsymbol{\theta}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{M-1} x^{M-1}$$

in polynomial regression gives us models with different complexities, where $\mathcal{M}_M$ determines the model with a $M-1$th degree polynomial.

Example 8.2 ()

Suppose I have data sampled data $x^{(1)}, \ldots, x^{(N)}$ on age at death for N people from an unknown distribution X. Then, possible models that model the distribution are

1. $\mathcal{M}_1$: the exponential distribution $p(x \mid \lambda) = \lambda e^{-\lambda y}$ with parameter $\theta = \lambda$.
2. $\mathcal{M}_2$: the gamma distribution $p(y \mid a, b) = (b^a / \Gamma(a)) y^{a-1} e^{-by}$ with parameter $\boldsymbol{\theta} = (a, b)$.
3. $\mathcal{M}_3$: the log-normal distribution with $X \sim N(\mu, \sigma^2)$ where $\boldsymbol{\theta} = (\mu, \sigma^2)$.

Example 8.3 ()

A mixture of Gaussians model

$$p(\mathbf{y}) = \sum_{m=1}^{M} \pi_m N(\mathbf{y} \mid \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$$

has submodels where we must determine the number of Gaussians M.

Now if we assume that the actual true distribution X or the true regressor $\mathbb{E}[Y \mid X]$ is contained within our model $\mathcal{M}$, then we say our model is **well-specified**. But since researchers have no idea what the data generating process is, so $\mathbb{E}[Y \mid X] \not\in \mathcal{M}$. Hence there is the saying that saying that "all models are wrong," since we never know what the true data generating process is, and thus the quantity

$$\mathbb{E}[Y \mid X] - h^*_{\boldsymbol{\theta}}(X)$$

where $h^*_{\boldsymbol{\theta}}(X)$ is the optimized hypothesis functions within $\mathcal{M}$, will always be nonzero. How close we can get this quantity to 0 determines how useful the model is, and a misspecified model is fundamentally a convenient (or even necessary) assumption on the distribution underlying the data, which may only be a reasonable approximation.

8.2 Feature Engineering

This is also very domain specific.

8.3 Data Preprocessing

8.3.1 Feature Extraction

The simplest linear function for regression is simply

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

This is called linear regression not because h is a linear function of $\mathbf{x}$. It is a linear function of $\mathbf{w}$. Therefore, we can fix nonlinear functions $\phi_j(\mathbf{x})$ and consider linear combinations of them.

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

We usually choose a dummy basis function $\phi_0(\mathbf{x}) = 1$ for notational convenience, so that if $\boldsymbol{\phi}$ is the vector of the function ϕ_j, then we can write $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$. This mapping from the original variables $\mathbf{x} \in \mathbb{R}^D$ to the basis functions $\{\phi_j(\mathbf{x})\}$, which span a linear function space of dimension M, is called **preprocessing** or **feature extraction** of the data.

Example 8.4 ()

Here are some examples of how we can extract features.

1. The mapping from a single variable x to its powers
$$x \mapsto (1, x, x^2, \ldots, x^{M-1}) \tag{132}$$
2. The mapping from a configuration of K atoms with their momenta in $\mathbb{R}^{6K}$ to their atomic cluster expansion polynomials.
3. The Legendre polynomials, which form an orthonormal basis in the space of polynomials.
4. Using equally spaced Gaussian basis functions over the dataset.

Changing the input space from D dimensions to M dimensions (i.e. extracting our M features) gives the design matrix

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \mathbf{x}^{(3)} \\ \vdots \\ \mathbf{x}^{(n)} \end{pmatrix} \implies \boldsymbol{\Phi} = \begin{pmatrix} — & \phi(\mathbf{x}^{(1)}) & — \\ — & \phi(\mathbf{x}^{(2)}) & — \\ \vdots & \vdots & \vdots \\ — & \phi(\mathbf{x}^{(n)}) & — \end{pmatrix} \tag{133}$$

We have shown that the `PolynomialFeatures` transformer converts our features to a polynomial basis. We can do this for an arbitrary number of features, for example if we map $D = 2$ to a second degree polynomial, we would have the transformation

$$(x_1, x_2) \mapsto (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$$

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
```

```
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

Sometimes, we are only worried about the interaction terms among features, so we can set the parameter `interaction_only=True`, which would, in the third degree case, transform the features

$$(x_1, x_2, x_3) \mapsto (1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3)$$

Spline transformers are piecewise polynomials, which is also built in. We notice that it is cumbersome to transform the dataset `X` with the transformer, store it into another variable, and train the model on that. We can "combine" the transforming (even multiple layers of transformers) and the model by implementing a "pipeline," which is initialized by inputting a list of tuples (name and the object) and has the same methods as the model.

```python
from sklearn.pipeline import Pipeline
model = Pipeline([("poly_transform", PolynomialFeatures(degree=2)),
                  ("lin_regression", LinearRegression())
                  ])
model.fit(X, y)
```

Now, let's talk about how we can implement a custom transformer. We basically have to create a new subclass that implements the `fit` (which always returns `self`) and the `transform` (which returns the transformed matrix) methods. Here we show for Gaussian basis functions.

```python
from sklearn.base import BaseEstimator, TransformerMixin

class GaussianFeatures(BaseEstimator, TransformerMixin):
    """Uniformly spaced Gaussian features for one-dimensional input"""

    def __init__(self, N, width_factor=2.0):
        self.N = N
        self.width_factor = width_factor

    def fit(self, X, y=None):
        # create N centers spread along the data range
        self.centers_ = np.linspace(X.min(), X.max(), self.N)
        self.width_ = self.width_factor * (self.centers_[1] - self.centers_[0])
        return self

    def transform(self, X):
        transformed_rows = []
        for mu in self.centers_:
            transformed_rows.append(stats.norm.pdf(X, mu, self.width_))

        return np.hstack(tuple(transformed_rows))

model = Pipeline([("gauss_transform", GaussianFeatures(20)),
                  ("lin_regression", LinearRegression())
                  ])

N = 60
X = np.random.uniform(-1, 1, size=(N, 1))
Y = true_func(X) + np.random.normal(0, 0.3, size=(N, 1))

model = Pipeline([("gauss_transform", GaussianFeatures(10)),
              ("lin_regression", LinearRegression())
              ])
model.fit(X, Y)
```

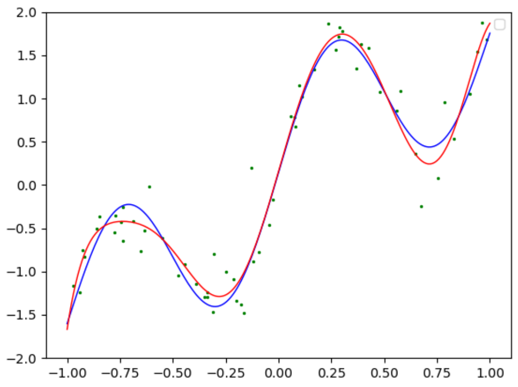
If we would like to impelment the fourier expansion of a function of form

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{N} a_n \cos(nx) + \sum_{n=1}^{N} b_n \sin(nx)$$

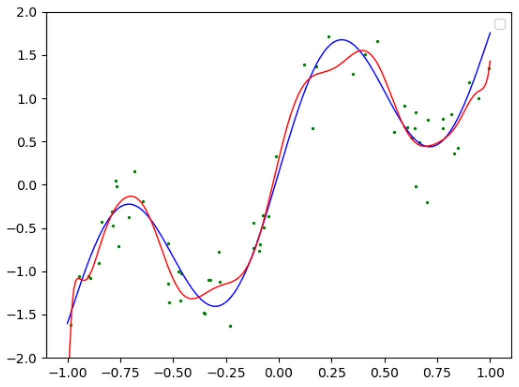
Then we would create the basis functions according to

```python
class FourierFeatures(BaseEstimator, TransformerMixin):
    "Fourier Expansion for one-dimensional input"

    def __init__(self, N):
        self.N = N

    def fit(self, X, Y=None):
        return self

    def transform(self, X):
        transformed_columns = []
        transformed_columns.append(np.ones(shape=X.shape))

        for n in range(self.N):
            transformed_columns.append(np.sin(n * X))
            transformed_columns.append(np.cos(n * X))

        print(np.hstack(tuple(transformed_columns)).shape)
        return np.hstack(tuple(transformed_columns))
```

and both of them would give the following fits to our original function $f(x) = \sin(2\pi x) + 2\cos(x - 1.5)$.



(a) Fitting with 10 Gaussian basis functions.



(b) Fitting with 10 Fourier basis functions.

Figure 19

8.3.2 Standardizing Data

Standardizing typically meanss that our featuers will be rescaled to have the properties of a standard normal distribution with mean of 0 and a standard deviation of 1. Here are a few methods to scale our data, with their results shown on a dataset of 30 points in $\mathbb{R}^2$.

1. **StandardScaler**: This is probably the most used method for standardizing data. It standardizes features by removing the mean and scaling to unit variance. The standard score of a sample $x^{(n)}$ is $(x - \bar{x})/S$ where $\bar{x}$ is the mean of the training samples and S is the standard deviation of the training samples.

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

2. **MinMaxScaler**: While not technically "standardization," MinMaxScaler is another preprocessing method for scaling. It transforms features by scaling each feature to a given range, typically between zero and one, or so that the maximum absolute value of each feature is scaled to unit size.

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

3. **MaxAbsScaler**: This scaler works similarly to the MinMaxScaler but scales in a way that the training data lies within the range $[-1, 1]$ by dividing through the largest maximum value in absolute value. It is meant for data that is already centered at zero or sparse data.

```python
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
scaled_data = scaler.fit_transform(data)
```

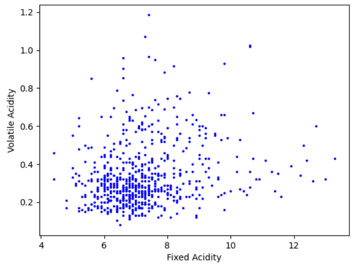
4. **RobustScaler**: This scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). It's robust to outliers, which makes it a good choice if you have data with possible outliers.

```python
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaled_data = scaler.fit_transform(data)
```

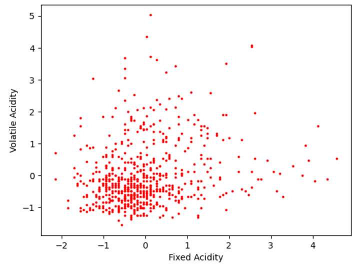
5. **QuantileTransformer**: Note that the presence of outliers messes with our scaling. More generally for skewed distributions (like an exponential), a linear transformation does not take care of these outliers, so we would like some nonlinear preprocessing algorithm. One common one is the QuantileTransformer, which takes the quantiles (percentiles) of the dataset and transforms then so that those are equidistant from each other. By default, it divides up the data into 1000 quantiles.

```python
from sklearn.preprocessing import QuantileTransformer
transformer = QuantileTransformer(n_quantiles = 100, output_distribution='normal')
transformed_data = transformer.fit_transform(data)
```

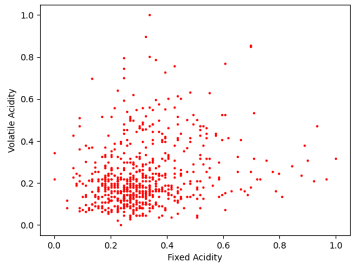
Let's talk about how these scalers will work on some data. We take a wine data with the two variables representing fixed acidity and volatile acidity.



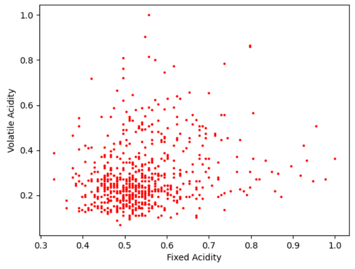
(a) Original Data



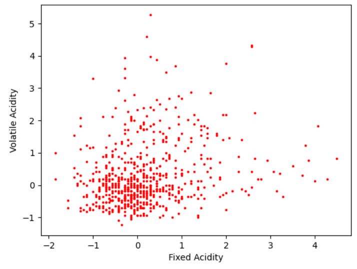
(b) StandardScaler



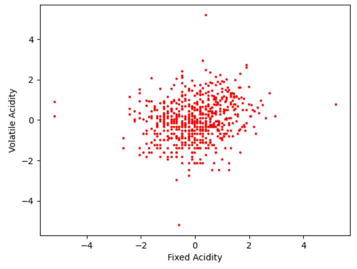
(c) MinMaxScaler



(d) MaxAbsScaler



(e) RobustScaler



(f) QuantileTransformer

Figure 20: The StandardScaler simply standardizes the data to have 0 mean and unit variance.

It's important to note that whether you should standardize your data and how you should do it depends on the specific characteristics of your data and the machine learning algorithm you're using. For example, some algorithms, like many in deep learning, assume that all features are on the same scale. Others, like Decision Trees and Random Forests, do not require feature scaling at all.

8.4 Data Augmentation

Bibliography

[GKKW02] L. Györfi, M. Kohler, A. Krzyzak, and H. Walk. *A Distribution-Free Theory of Nonparametric Regression.* Springer Series in Statistics. Springer New York, 2002.

[HKZ14] Daniel Hsu, Sham M. Kakade, and Tong Zhang. Random design analysis of ridge regression, 2014.