Class Notes (experimental)

Jonathan Rosenblatt

May 18, 2015

Contents

List of Algorithms

List of Definitions

List of Examples

Chapter 1

Introduction

This text is intended to collect many machine learning methods and algorithms, present then, and organize them. The treatment of the different concepts attempts to be as intuitive as possible, and mathematics is presented only when unavoidable, or when adding special insight.

Extra effort has been put into organizing the methods and algorithms along some fundamental building blocks, which are now briefly presented.

The learning problem The first distinction between the different methods is along the tasks they perform, which closely corresponds to the type of data at hand. This includes Supervised Learning (§3) and Unsupervised Learning (§5). Within each, we find several sub-tasks:

Supervised Learning Includes classification tasks, and regression tasks. The first, predicting a categorical outcome, and the latter, a continuous outcome.

Unsupervised Learning Includes the learning of the data generating distribution, or related tasks such as detecting high density regions and clustering.

As we will see, not all learning tasks fall into these categories. Collaborative Filtering (§6) is an example of a learning task that is neither. It is a missing data imputation task.

An optimization problem vs. an algorithm Both supervised and unsupervised learning methods can be classified as either an explicit algorithm, or as an optimization problem, agnostic to the optimization algorithm used to solve it.

Dimension Reduction Both supervised and unsupervised learning methods can include a dimensionality reduction stage. This can be motivated by the need to reduce computational complexity, to apply low-dimensional algorithms down the road, to allow a visualization of the data, or simply to allow some human tractable interpretation of the data. This is discussed in Appendix E.

We can further stratify dimensionality reduction methods along these lines:

Linear-Space vs. Non-Linear-Space Embeddings When reducing the dimension of the data, it can be mapped (embedded) into a linear subspace of the original space, or a non linear one.

Linear vs. Non-Linear Space Embeddings Not to be confused with the previous item. The dimensionality reduction can be a linear operation on the data or a non-linear one.

Learning an Embedding vs. Learning an Embedding Function When learning a mapping to a lower dimensional space, we can map the original data points (an embedding), or learn a mapping of the whole data space (an embedding function).

Kernel Trick Both supervised and unsupervised learning methods can include a "kernel trick". This will happen when we wish to learn complex functions of the data, but keep computations quick. The kernel trick is applicable to methods that do not need the whole data, but rather, only some measure of similarity between the points. The idea is that many complicated functions, are merely linear combinations of the distances to other points. This is further elaborated in Appendix B.

Generative vs. Discriminative Models Both supervised and unsupervised learning methods can benefit from an assumption on the data generating process, i.e., the sampling distribution. Generative models as those where we assume this process. Discriminative models, which appear in supervised learning, we do not assume the data generating process, but merely the nature of the relation between features and outcome.

Feature based vs. Graph Based Unsupervised learning tasks can be classified to those that require the full features of the data, and those who require only some measure of similarity between data points. As such, the latter methods can be seen as graph based methods, where the similarities are represented as a graph.

Fully Observed vs. Latent Space Models Both supervised and unsupervised learning methods can include unobservable, i.e. latent, variables.

Fully Automated Processes? The machine learning literature draws heavily from the statistical literature. You should bear in mind that the ultimate goal of machine learning, is replacing a "hard-coded" algorithm, which externalizes the programmer's knowledge, into a self teaching algorithm. It may thus seem that problems like visualization do not belong in the realm of machine learning, as they are not completely automated. This is not completely accurate because, while we want the *application* stage of an algorithm to be automated, we can sometimes allow for a human to be involved in the *learning* stage.

Notation The notation conventions may seem non standard as they borrow from several lines of literature. These conventions were chosen as we find them to be clear and concise. They are collected in Appendix H.

Sources This text draws mostly from Hastie et al. [2003] and Shalev-Shwartz and Ben-David [2014]. The former is freely available online. For a softer introduction, with more hands-on examples, see James et al. [2013], also freely available online. All books are very well written and strongly recommended. More references can be found in the Bibliography (Appendix H).

Chapter 2

Estimation

In this section, we present several estimation principles. Their properties are not discussed, as the section is merely a reminder and a preparation for what follows. These concepts and examples can be found in many introductory books to statistics. I particularly recommend Wasserman [2004] or Abramovich and Ritov [2013].

2.1 Moment matching

The fundamental idea: match empirical moments to theoretical. I.e., estimate

$$\mathbf{E}\left[g(X)\right]$$

by

$$\mathbb{E}\left[g(X)\right]$$

where $\mathbb{E}\left[g(x)\right] := \frac{1}{n}\sum_i g(x_i)$, is the empirical mean.

Example 2.1.1 (Exponential Rate)**.** Estimate λ in $\mathbf{x}_i \sim exp(\lambda)$, $i = 1, \ldots, n$, i.i.d. $\mathbf{E}\left[x\right] = 1/\lambda$. $\Rightarrow \widehat{\lambda} = 1/\mathbb{E}\left[x\right]$.

Example 2.1.2 (Linear Regression)**.** Estimate β in $\mathbf{y} \sim \mathcal{N}\left(X\beta, \sigma^2 I\right)$, a p dimensional random vector. $\mathbf{E}\left[y\right] = X\beta$ and $\mathbb{E}\left[y\right] = y$. Clearly, moment matching won't work because no β satisfies $X\beta = y$. A technical workaround: Since β is p dimensional, I need to find some $g(\mathbf{y}) : \mathbb{R}^n \mapsto \mathbb{R}^p$. Well, $g(y) := Xy$ is such a mapping. I will use it, even though my technical justification is currently unsatisfactory. We thus have: $\mathbf{E}\left[X'y\right] = X'X\beta$ which I match to $\mathbb{E}\left[X'y\right] = X'y$:

$$X'X\beta = X'y \Rightarrow \widehat{\beta} = (X'X)^{-1}X'y.$$

2.2 Quantile matching

The fundamental idea: match empirical quantiles to theoretical. Denoting by $F_{\mathbf{x}}(t)$ the CDF of $\mathbf{x}$, then $F_{\mathbf{x}}^{-1}(\alpha)$ is the α quantile of $\mathbf{x}$. Also denoting by $\mathbb{F}_x(t)$ the Empirical CDF of $x_1, \ldots, x_n$, then $\mathbb{F}_x^{-1}(\alpha)$ is the α quantile of $x_1, \ldots, x_n$. The quantile matching method thus implies estimating

$$F_{\mathbf{x}}^{-1}(\alpha)$$

by

$$\mathbb{F}_x^{-1}(\alpha).$$

Example 2.2.1 (Exponential rate)**.** Estimate λ in $\mathbf{x}_i \sim exp(\lambda)$, $i = 1, \ldots, n$, i.i.d.

$$\begin{aligned}
&F_{\mathbf{x}}(t) = 1 - \exp(-\lambda t) = \alpha \Rightarrow \\
&F_{\mathbf{x}}^{-1}(\alpha) = \frac{-\log(1-\alpha)}{\lambda} \Rightarrow \\
&F_{\mathbf{x}}^{-1}(0.5) = \frac{-\log(0.5)}{\lambda} \Rightarrow \\
&\widehat{\lambda} = \frac{-\log(0.5)}{\mathbb{F}_x^{-1}(0.5)}.
\end{aligned}$$

2.3 Maximum Likelihood

The fundamental idea is that if the data generating process (i.e., the *sampling distribution*) can be assumed, then the observations are probably some high probability instance of this process, and not a low probability event: Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim P_\theta$, with density (or probability) $p_\theta(x_1, \ldots, x_n)$. Denote the likelihood, as a function of θ: $\mathcal{L}(\theta) : p_\theta(x_1, \ldots, x_n)$. Then

$$\widehat{\theta}_{ML} := argmax_\theta \left\{ \mathcal{L}(\theta) \right\}.$$

Using a monotone mapping such as the log, does not change the $argmax$. Denote

$$L(\theta) := \log(\mathcal{L}(\theta)).$$

Example 2.3.1 (Exponential rate)**.** Estimate λ in $X_i \sim exp(\lambda)$, $i = 1, \ldots, n$, i.i.d. Using the exponential PDF and the i.i.d. assumption

$$\mathcal{L}(\lambda) = \lambda^n \exp(-\lambda \sum_i X_i),$$

and

$$L(\lambda) = n\log(\lambda) - \lambda\sum_i X_i.$$

By differentiating and equating 0, we get $\widehat{\lambda}_{ML} = 1/\mathbb{E}[X]$.

Example 2.3.2 (Discrete time Markov Chain)**.** Estimate the transition probabilities, p_1 and p_2 in a two state, $\{0,1\}$, discrete time, Markov chain where: $P(\mathbf{x}_{t+1} = 1|x_t = 0) = p_1$ and $P(\mathbf{x}_{t+1} = 1|X_t = 1) = p_2$. The likelihood:

$$\mathcal{L}(p_1, p_2) = P(X_2, \ldots, X_T; X_1, p_1, p_2) = \prod_{t=1}^{T} P(X_{t+1} = x_{t+1}|X_t = x_t).$$

We denote n_{ij} the total number of observed transitions from i to j and get that $\widehat{p}_1 = \frac{n_{01}}{n_{01}+n_{00}}$, and that $\widehat{p}_2 = \frac{n_{11}}{n_{11}+n_{10}}$.

Remark 2.3.1 (Confession)**.** Well, this is a rather artificial example, as because of the Markov property, and the stationarity of the process, we only need to look at transition events, themselves Bernoulli distributed. This example does show, however, the power of the ML method to deal with non i.i.d. samples. As does the next example.

Example 2.3.3 (Autoregression of order 1 (AR(1)))**.** Estimate the drift parameter a, in a discrete time Gaussian process where: $\mathbf{x}_{t+1} = a\mathbf{x}_t + \varepsilon; \varepsilon \sim \mathcal{N}(0, \sigma^2) \Rightarrow \mathbf{x}_{t+1}|\mathbf{x}_t \sim \mathcal{N}(ax_t, \sigma^2)$.

We start with the conditional density at time $t+1$:

$$p_{\mathbf{x}_{t+1}|\mathbf{x}_t = x_t}(x_{t+1}) = (2\pi\sigma^2)^{-1/2}\exp\left(-\frac{1}{2\sigma^2}(x_{t+1} - ax_t)^2\right).$$

Moving to the likelihood:

$$\mathcal{L}(a) = (2\pi\sigma^2)^{-T/2}\exp\left(-\frac{1}{2\sigma^2}\sum_{t=1}^{T}(x_{t+1} - ax_t)^2\right).$$

Taking the log and differentiating with respect to a and equating 0 we get $\widehat{a}_{ML} = \frac{\sum x_{t+1}x_t}{\sum x_t^2}$.

We again see the power of the ML device. Could we have arrived to this estimator by intuiton alone? Hmmmm... maybe. See that $Cov[X_{t+1}, X_t] = a\ Var[X_t] \Rightarrow a = \frac{Cov[X_{t+1}, X_t]}{Var[X_t]}$. So a can also be derived using the moment matching method which is probably more intuitive.

Example 2.3.4 (Linear Regression)**.** Estimate β in $Y \sim \mathcal{N}(X\beta, \sigma^2 I)$, a p dimensional random vector. Recalling the multivariate Gaussian PDF:

$$p_{\mu,\Sigma}(y) = (2\pi)^{-n/2}|\Sigma|^{-1/2}\exp\left(-\frac{1}{2}(y-\mu)'\Sigma^{-1}(y-\mu)\right)$$

So in the regression setup:

$$\mathcal{L}(\beta) = p_{\beta,\sigma^2}(y) = (2\pi)^{-n/2}|\sigma^2 I|^{-1/2}\exp\left(-\frac{1}{2\sigma^2}\|y - X\beta\|_2^2\right)$$

and $\widehat{\beta}_{ML}$ equals

$$\widehat{\beta}_{ML} = (X'X)^{-1}X'y. \tag{2.1}$$

2.4 M-Estimation and Empirical Risk Minimization

M-Estimation, know as Empirical Risk Minimizaton (ERM) in the machine learning literature, is a very wide framework which stems from statistical desicion theory. The underlying idea is that each realization of $\mathbf{x}$ incurs some loss, and we seek to find a "policy", in this case a parameter, θ^* that minimizes the average loss. In the econometric literature, we dot not incur a loss, but rather a utility, we thus seek a policy that maximizes the average utility.

Definition 2.1 (Loss Function)**.** The penalty for predicting θ when observing x:

$$l(x;\theta). \tag{2.2}$$

Definition 2.2 (Risk Function)**.** The expected loss:

$$R(\theta) := \mathbf{E}\left[l(\mathbf{x};\theta)\right]. \tag{2.3}$$

Then the best prediction, θ^*, being the minimizer of the expected risk is

$$\theta^* := \underset{\theta}{argmin}\left\{R(\theta)\right\}. \tag{2.4}$$

As we do not know the distribution of $\mathbf{x}$, we cannot solve Eq.(2.4), so we minimize the *empirical* risk.

Definition 2.3 (Empirical Risk)**.** The average loss in the sample:

$$\mathbb{R}(\theta) := \mathbb{E}\left[l(x;\theta)\right] = \frac{1}{n}\sum_i l(x_i,\theta). \tag{2.5}$$

A prediction that can actually be computed with data is thus the empirical minimizer $\widehat{\theta}$:

$$\widehat{\theta} := \underset{\theta}{argmin}\left\{\mathbb{R}(\theta)\right\}. \tag{2.6}$$

Example 2.4.1 (Squared Loss)**.** Let $l(x;\theta) = (x-\theta)^2$. Then $R(\theta) = \mathbf{E}\left[(\mathbf{x}-\theta)^2\right] = (\mathbf{E}\left[\mathbf{x}\right]-\theta)^2 + Var[\mathbf{x}]$. Clearly $Var[\mathbf{x}]$ does not depend on θ so that $R(\theta)$ is minimized by $\theta^* = \mathbf{E}\left[\mathbf{x}\right]$. **We thus say that the expectation of a random variable is the minimizer of the squared loss.**

How do we estimate the population expectation? Well a natural estimator is the empirical mean, which is also the minimizer of the empirical risk $\mathbb{R}(x)$. The proof is immediate by differentiating.

Example 2.4.2 (Ordinary Least Squares (OLS))**.** Define the loss $l(y,x;\beta) := \frac{1}{2}(y-x\beta)^2$. Computing the risk, $\mathbf{E}\left[\frac{1}{2}\|\mathbf{y}-\mathbf{x}\beta\|_2^2\right]$ will require dealing with the joint distribution of $(\mathbf{x},\mathbf{y})$. We don't really care about that right now. We merely want to see that the empirical risk minimizer, is actually the classical OLS problem. And well, it is (by definition actually):

$$\mathbb{R}(\beta) = \sum_{i=1}^{n}\frac{1}{2}(y_i - x_i\beta)^2 = \frac{1}{2}\|y-x\beta\|_2^2.$$

Minimization is easiest with vector derivatives, but I will stick to regular derivatives:

$$\frac{\partial\mathbb{R}(\beta)}{\partial\beta_j} = \sum_i\left[(y_i - \sum_{j=1}^{p}x_{ij}\beta_j)(-x_{ij})\right]$$

Equating 0 yields $\widehat{\beta_j} = \frac{\sum_i y_i x_{ij}}{\sum_i x_{ij}^2}$. Solving for all j's and putting in matrix notation we get

$$\widehat{\beta}_{OLS} = (X'X)^{-1}X'y. \tag{2.7}$$

2.5 Notes

2.5.1 Maximum Likelihood

Maximum likelihood estimators are a particular instance of M-estimators, if we set the loss function to be the negative log likelihood of the (true) sampling distribution.

2.5.2 Choosing the Loss Function

While by far the most popular, we do not automatically revert to minimizing a squared error loss. There are several considerations when choosing the loss function. Most ERM learning methods can be applied with different loss functions.

The first consideration is computational complexity: if you choose a loss function that leads to a non-convex empirical risk, you are in trouble. There are no guarantees you will be able to find the risk minimizer in finite computing time.

The second consideration is the nature of the outcome y. Some loss functions are more appropriate to continuous y's and some for categorical y's. Typical loss functions for continuous y's are the squared loss, absolute loss, and hinge loss. Typical loss functions for categorical y's are the Binomial likelihood loss (also known as the Cross Entropy, or Deviance), and the hinge loss.

A third consideration, which is rarely given the importance it should get, is "What is the meaning of θ^*"? Or, "What are we actually estimating"? As we have seen in Example 2.4.1, the squared loss implies we are aiming to estimate the population mean. What are we estimating when we use the hinge loss? The binomial loss? We will not discuss these matters, as we are discussing methods where these choices have already been made for us. When the day you start thinking of your own learning algorithms, you will need to give some thought to this question.

Chapter 3

Supervised Learning

3.1 Supervised Learning By Empirical Risk Minimization (EMR)

3.1.1 Empirical Risk Minimization and Inductive Bias

In Supervised Learning, or *learning with teacher*, we want to extract the relation $y = f(x)$ between attributes x and some outcome y. It is implicit, and essential, that the outcomes are observed. If this is not the case, see *Unsupervised Learning* (§5). The attributes, also know as *features*, or *predictors*, are assumed to belong to some *feature space* $\mathcal{X}$.

Feature Space

Unlike classical statistics, we don't care to explain the causal process relating x and y, we just want good predictions. The implied ERM problem is

$$\widehat{f}(x) := \underset{f}{argmin}\left\{\mathbb{R}(f(x))\right\} = \underset{f}{argmin}\left\{\frac{1}{n}\sum_i l(y_i - f(x_i))\right\}. \tag{3.1}$$

Alas, there are clearly infinitely many f for which $\mathbb{R}(\widehat{f}(x)) = 0$, in particular, all those where $\widehat{f}(x_i) = y_i, \forall i$. All these f feel like very bad predictors, as they *overfit* the observed data, at a cost of generalizability. We formalize this intuition in Section 4.

Overfitting

We need to make sure that we do not learn overly complex predictors, which generalize poorly to new data. Motivated by the fact that humans approach new problems equipped with their past experience, this regularization is called *Inductive Bias*. There are several ways to introduce this bias, which can be combined:

Inductive Bias

Constrained Hypothesis Classes We typically do not allow f to be "any function" but rather restrict it to belong to a certain class. In the

machine learning terminology, f is a Hypothesis, and it belongs to $\mathcal{F}$ which is the Hypothesis Class.

Hypothesis

Regularization We do not need to treat all $f \in \mathcal{F}$ equivalently. We might have prior preferences towards particular f's , typically simple f's, and we can introduce these preferences in the learning process. This is called *Regularization*.

Regularization

Non ERM Approaches Many learning problems can be cast as ERM problems, but another way to introduce bias is by learning f in a non-optimal manner. Inductive bias is thus introduced by the (non) optimization algorithm Learning algorithms that cannot be cast as ERMs include: Nearest Neighbour (§3.2.1), Kernel Regression(§3.2.2), Boosting (§3.2.4).

We now proceed to show that many supervised learning algorithms are in fact ERMs with some type of inductive bias: constrained hypothesis class and\or or regularization and\or suboptimality.

3.1.2 Ordinary Least Squares (OLS)

As seen in Example 2.4.2, by adopting a squared error loss, and restricting $\mathcal{F}$ by assuming f is a linear function of x, we get the OLS problem. In this case learning f is effectively the same as learning β as they are isomorphic, as will be the case in all hypothesis classes that admit a finite dimensional parametrization.

Regularization While the OLS problem seemingly has no regularization parameters, we are in no way restricted to the original X matrix supplied to us. We are free to choose any subset of the X variables. Selecting a subset of the X variables is known as *model selection*. We will discuss many different methods to choose the regularization level, in this case, the variables in the OLS, in Section 4.

Model Selection

We are not only free to select a subset of Xs, but we can also augment X with new variables, consisting of non linear transformation of the original ones. This is know as *basis augmentation*. The idea of basis augmentation is a useful and fundamental one that will revisit us throughout the course, even if we typically don't call it by its name.

Basis Augmentation

Remark 3.1.1 (No Sampling Distribution)**.** We emphasize, that unlike the classical statistical literature, we never assumed any sampling distribution.

This is why we will not be doing hypothesis testing on the model parameters. ERM can be thus seen as *exploratory* statistics.

In particular, we are not discussing hypothesis testing and confidence intervals for the OLS problem, for this same reason: because we are not assuming the data generating distribution. This, in contrast to the classical statistical literature where the linear model is both the data generating process, and the assume hypothesis class.

In statistical terminology, we may be fitting *misspecified models*, we simply do not care about it, as long as predictions are good.

Misspecified Model

Remark 3.1.2 (OLS Extensions)**.** Most if not all extensions of OLS, such as Generalized Least Squared (GLS) and Generalized Linear Models (§3.1.5) are ERM problems.

3.1.3 Ridge Regression

Consider the Ridge regression problem:

$$\underset{\beta}{argmin} \left\{ \frac{1}{n} \sum_i (y_i - x_i\beta)^2 + \frac{\lambda}{2} \|\beta\|_2^2 \right\} \tag{3.2}$$

$$\widehat{\beta}_{Ridge} = (X'X + \lambda I)^{-1} X'y \tag{3.3}$$

We can see that again, $\mathcal{F}$ is restricted to be the space of linear functions of x, but we also add a regularization that favours linear functions with small coefficients.

The regularization of β can have several interpretations and justifications:

A mathematical device Strengthening the diagonal of $X'X$ makes is more easily invertible. This is a standard tool in applied mathematics called *Tikhonov Regularization*. It is also helpful when dealing with multicolinearity, as $(X'X + \lambda I)$ is always invertible.

A Subjective Bayesian View If we believe that β should be small; say our beliefs can be quantified by $\beta \sim \mathcal{N}(0, 1/\lambda I)$, then the Ridge solution is actually the mean of our posterior beliefs on $\beta|y$.

Whatever justification you prefer, it can be easily shown that $\frac{\partial R(\lambda,\beta)}{\partial \lambda}$ at $\lambda = 0$ is negative, thus, predictions only improve by introducing some small regularization (how small is small? See §4).

For more on Ridge regression see Hastie et al. [2003].

3.1.4 LASSO

Consider the LASSO problem:

$$\underset{\beta}{argmin} \left\{ \frac{1}{n} \sum_i (y_i - x_i\beta)^2 + \lambda \|\beta\|_1 \right\} \tag{3.4}$$

As can be seen, just like in Ridge regression, $\mathcal{F}$ is restricted to linear functions of x. The regularization however differs. Instead of l_2 norm penalty, we use an l_1 norm penalty. Eq.(3.4) does not have a closed form solution for $\widehat{\beta}_{LASSO}$ but the LARS algorithm, a quadratic programming algorithm, solves it efficiently.

The LASSO has gained much popularity as it has the property that $\widehat{\beta}_{LASSO}$ has many zero entries. It is thus said to be *sparse*. The sparsity property is very attractive as it acts as a model selection method, allowing to consider Xs where $p > n$, and making predictions computationally efficient.

Sparsity

The sparsity property can be demonstrated analytically for the orthogonal design case ($X'X = I$), in which $\widehat{\beta}$ admits a closed form solution:

$$\widehat{\beta}_{j,LASSO} = sign(\widehat{\beta}_{j,OLS}) \left[|\widehat{\beta}_{j,OLS}| - \frac{\lambda}{2} \right]_+ . \tag{3.5}$$

We thus see that the LASSO actually performs *soft thresholding* on the OLS estimates.

Soft Thresholding

3.1.5 Logistic Regression

The logistic regression is the first categorical prediction problem we consider. I.e., the outcome y is not a continuous variable, but rather takes values in some finite set $\mathcal{G}$. In the logistic regression problem, it can take two possible values. In the statistical literature, y is encoded as $\mathcal{G} = \{0, 1\}$ and f is assumed to take to take the following form:

Categorical Prediction

$$p(x) := P(\mathbf{y} = 1|x) = \Psi(x\beta) \tag{3.6}$$

$$\Psi(t) = \frac{1}{1 + e^{-t}} \tag{3.7}$$

The hypothesis class is thus $\mathcal{F} = \{f : f(x) = \Psi(x\beta)\}$.

In the $\{0, 1\}$ encoding, the loss is the negative log likelihood, i.e.:

$$l(y, x, \beta) = -\log \left[p(x)^y (1 - p(x))^{1-y} \right] = -\log \left[\Psi(x\beta)^y (1 - \Psi(x\beta))^{1-y} \right]. \tag{3.8}$$

In the learning literature it is more common for $\mathcal{G} = \{1, -1\}$ encoding of y in which case the loss is

$$l(y, x, \beta) = -\log\left[1 + \exp(-yf(x))\right]. \tag{3.9}$$

How to classify? In the $\{0, 1\}$ encoding, we predict class 1 if $\Psi(x\beta) > 0.5$ and class 0 otherwise. The logistic problem thus defines a separating hyperplane $\mathbb{L} \subset \mathcal{X}$ between the classes: $\mathbb{L} = \{x : f(x) = 0.5\}$.

In the $\{1, -1\}$ encoding, we predict class 1 if $\Psi(x\beta) > 0$ and class 0 otherwise. The plane $\mathbb{L}$ is clearly invariant to the encoding of y.

Remark 3.1.3 (Generalized Linear Models (GLM))**.** Logistic regression is a particular instance of the very developped theory of Generalized Linear Models. These models include the OLS, Probit Regression, Poisson Regression, Quasi Likelihood, Multinomial Regression, Proportional Odds regression and more. The ultimate reference on the matter is McCullagh and Nelder [1989]. GLM

3.1.6 Regression Classifier

Can we use the OLS framework for prediction? Yes! With proper encoding of y. Solving the same problem from Example 2.4.2 by encoding y as $\{0, 1\}$ gives us the linear separating hyperplane $\mathbb{L} = \left\{x : x\widehat{\beta}_{OLS} = 0.5\right\}$.

Remark 3.1.4. We can interpret $\widehat{y}$ as the probability of an event, but there a slight technical difficulty as $\widehat{y}$ might actually be smaller than 0 or larger than 1.

3.1.7 Linear Support Vector Machines (SVM)

The literature typically presents the SVM method using the geometrical arguments that originally motivated the problem. For our purposes, we present the ERM formulation of the problem, followed by the original arguments. Assuming a linear hypothesis class, $f(x) = \beta_0 + \beta x$, and $\mathcal{G} = \{-1, 1\}$ encoding, the SVM ERM problem is

$$\min_{\beta, \beta_0} \left\{ \sum_i \left[1 - y_i f(x_i)\right]_+ + \frac{\lambda}{2}\|\beta\|_2^2 \right\}. \tag{3.10}$$

Eq.(3.10) thus reveals that the linear SVM is actually an ERM problem, over a linear hypothesis class, with what is called the *hinge* loss function and l_2 regularization of β.

Geometrical Motivation for SVM

For completeness, we present Vapnik's original geometric motivation [Vapnik, 1998] for the formulation of the SVM problem.

We do not assume anything on the data, and seek for a hyperplane $\mathbb{L}$ that seperates two classes. Encode y as $\mathcal{G} = \{-1, 1\}$. Define a plane $\mathbb{L} = \{x : f(x) = 0\}$, and assume a linear hypothesis class, $f(x) = x\beta + \beta_0$. Now find the plane $\mathbb{L}$ that maximizes the (sum of) distances to the data points. We call the minimal distance from $\mathbb{L}$ to the data points, the *Margin*, and denote it by M.

To state the optimization problem, we need to note that $f(x) = x\beta + \beta_0$ is not only the value of our classifier, but it is actually proportional to the signed distance of x from $\mathbb{L}$.

Proof. The distance of x to $\mathbb{L}$ is defined as $\min_{x_0 \in \mathbb{L}} \{\|x - x_0\|_2\}$. Note $\beta^* := \beta/\|\beta\|_2$ is a normal vector to $\mathbb{L}$, since $\mathbb{L} = \{x : x\beta + \beta_0 = 0\}$, so that for $x_1, x_2 \in \mathbb{L} \Rightarrow \beta(x_1 - x_2) = 0$. Now $x - x_0$ is orthogonal to $\mathbb{L}$ because x_0 is, by definition, the orthogonal projection of x onto $\mathbb{L}$. Since $x - x_0$ are both orthogonal to $\mathbb{L}$, they are linearly dependent, so that by the Cauchy Schwarz inequality $\|\beta^*\|_2\|x - x_0\|_2 = \|\beta^*(x - x_0)\|_2$. Now recalling that $\|\beta^*\|_2 = 1$ and $\beta^* x_0 = -\beta_0/\|\beta\|_2$ we have $\|x - x_0\|_2 = \frac{1}{\|\beta\|_2}(x\beta + \beta_0) = \frac{1}{\|\beta\|_2} f(x)$. □

Using this fact, then $y_i f(x_i)$ is the distance from $\mathbb{L}$ to point i, positive for correct classifications and negative for incorrect classification. The (linear) support vector classifier is defined as the solution to

$$\max_{\beta, \beta_0} \{M \quad s.t. \quad \forall i : y_i f(x_i) \geq M, \quad \|\beta\|_2 = 1\} \tag{3.11}$$

If the data is not separable by a plane, which is typically the case, we need to allow some slack. We thus replace $y_i f(x_i) \geq M$ with $y_i f(x_i) \geq M(1 - \xi_i)$, for $\xi_i > 0$ but require that the missclassifications are controlled using a regularization parameter C: $\sum_i \xi_i \leq C$. Eq.(3.11) now becomes [Hastie et al., 2003, Eq.(12.25)]

$$\max_{\beta, \beta_0} \left\{ M \quad s.t. \quad \forall i : y_i f(x_i) \geq M(1 - \xi_i),\ \|\beta\|_2 = 1,\ \sum_i \xi_i \leq C,\ \forall i : \xi_i \ \geq 0 \right\} \tag{3.12}$$

See Section 12 in Hastie et al. [2003] for more details on SVMs.

Remark 3.1.5 (Name Origins)**.** SVM takes its name from the fact that $\widehat{\beta}_{SVM} = \sum_i \widehat{\alpha}_i y_i x_i$. The explicit form of $\widehat{\alpha}_i$ can be found in [Hastie et al.,

2003, Section 12.2.1]. For our purpose, it suffices to note that $\widehat{\alpha}_i$ will be 0 for all data points far away from $\mathbb{L}$. The data points for which $\widehat{\alpha}_i > 0$ are the *support vectors*, which give the method its name.

Remark 3.1.6 (Solve the right problem)**.** Comparing with the logistic regression, and the linear classifier, we see that the SVM cares only about the decision boundary $\mathbb{L}$. Indeed, if only interested in *predictions*, estimating *probabilities* is a needless complication. As Put by Vapnik:

> When solving a given problem, try to avoid a more general problem as an intermediate step.

Then again, if the assumed logistic model of the logistic regression, is actually a good approximation of reality, then it will outperform the SVM as it borrows information from all of the data, and not only the support vectors.

3.1.8 Generalized Additive Models (GAMs)

A way to allow for a more broad hypothesis calss $\mathcal{F}$, that is still not too broad, so that overfitting is hopefully under control, is by allowing the predictor to be an additive combination of simple functions. We thus allow $f(x) = \beta_0 + \sum_{j=1}^{p} f_j(x_j)$. We also not assume the exact form of $\{f_j\}_{j=1}^{p}$ but rather learn them from the data, while constraining them to take some simple form. The ERM problem of GAMs is thus

$$\underset{f \in \mathcal{F}}{argmin} \left\{ \frac{1}{n} \sum_i (y_i - f(x_i))^2 \right\} \tag{3.13}$$

where f is as defined above.

Remark 3.1.7 (Not a pure ERM)**.** The learning of $\{f_j\}_{j=1}^{p}$ is in fact not performed by optimization, but rather by kernel smoothing (§ 3.2.2). The solution to Eq.(3.13) is not a pure ERM problem, but a hybrid between ERM and kernel smoothing.

3.1.9 Projection Pursuit Regression (PPR)

Another way to generalize the hypothesis class $\mathcal{F}$, which generalizes the GAM model, is to allow f to be some simple function of a linear combination of the predictors. Let

$$f(x) = \sum_{m=1}^{M} g_m(w_m x) \tag{3.14}$$

where both $\{g_m\}_{m=1}^M$ and $\{w_m\}_{m=1}^M$ are learned from the data. The regularization is now performed by choosing M and the class of $\{g_m\}_{m=1}^M$. The ERM problem is the same as in Eq.(3.13), with the appropriate f.

Remark 3.1.8 (Not a pure ERM)**.** Just like the GAM problem, in the PPR problem $\{g_m\}_{m=1}^M$ are learned by Kernel Regresson (§3.2.2). Solving the PPR problem is thus a hybrid of ERM and Kernel Regresson algorithms.

Remark 3.1.9 (Universal Approximator)**.** By choosing a sufficiently large M, the class $\mathcal{F}$ can approximate any continuous function. This property of the class is called a *Universal Approximator*.

Universal Approximator

3.1.10 Neural Networks (NNETs)

Single Hidden Layer

In the spirit of [Hastie et al., 2003, Section 11], we introduce the NNET model via the PPR model, and not through its historically original construction. In the language of Eq.(3.14), a *single-layer–feed-forward* neural network, is a model where $\{g_m\}_{m=1}^M$ are not learned from the data, but rather assumed a-priori.

$$g_m(x) := \beta_m \sigma(x\alpha_m)$$

where only $\{\beta_m, \alpha_m\}_{m=1}^M$ are learned from the data. A typical *activation function*, $\sigma(t)$ is the standard logistic CDF: $\sigma(t) = \frac{1}{1+e^{-t}}$. Another popular alternative are Gaussian radial functions.

Activation Function

As can be seen, the NNET is merely a non-linear regression model. The parameters of which are often called *weights*.

NNETs have gained tremendous popularity, as they strike a good balance between model complexity and regularity; particularly in the machine vision, sound analysis and natural language processing domains, where data samples are abundant.

Loss Functions Like any other ERM problem, we are free to choose the appropriate loss function. For regression the squared loss is used. For classification, one can still use the squared error (as in the Regression Classifier), or the binomial likelihood leading to what is know as the *deviance*, or *cross-entropy* loss.

Deviance & Cross Entropy

Universal Approximator Like the PPR, even when $\{g_m\}_{m=1}^M$ are fixed beforehand, the class is still a universal approximator (although it might require a larger M than PPR).

Regularization Regularization of the model is done via the selection of the σ, the number of nodes/variables in the network and the number of layers. More that one hidden layer leads to *Deep Neural Networks* which offer even more flexibility at the cost of complexity, thus requiring many data samples for fitting.

Deep Neural Networks

Back Propagation The fitting of such models is done via a coordinate-wise gradient descent algorithm called *back propagation*.

Further Reading For more on NNETs see [Hastie et al., 2003, Chapter 11]. For a recent overview of Deep Learning in Neural Networks see Schmidhuber [2015].

3.1.11 Classification and Regression Trees (CARTs)

CARTs are a type of ERM where $f(x)$ include very non smooth functions that can be interpreted as "if-then" rules, also know as *decision trees*.

Decision Tree

The hypothesis class of CARTs includes functions of the form

$$f(x) = \sum_{m=1}^{M} c_m I_{\{x \in R_m\}} \tag{3.15}$$

where I_A is the indicator function of the event A. The parameters of the model are the different conditions $\{R_m\}_{m=1}^{M}$ and the function's value at each condition $\{c_m\}_{m=1}^{M}$.

Regularization is done by the choice of M which is called the *tree depth*.

Tree Depth

As $f(x)$ is defined over indicator functions, CARTs have no difficulty to deal with categorical variables and even missing values, on top of the more standard continuous predictors. More on the many advantages of CARTs can be found in the references.

Optimization As searching over all possible partitions of the x's to optimize $\{R_m\}_{m=1}^{M}$ is computationally impossible, optimization is done is a greedy fashion, by splitting on each variable x_j at a time. The problem presented in Eq. 3.15 is actually a decision *list*. It is the nature of the optimization that makes the solution a decision *tree*.

Decision List

Loss Functions As usual, a squared loss can be used for continuous outcomes y. For categorical outcomes, the loss function is called the *impurity*

measure. One can use either a misclassification error, the multinomial likelihood (knows as the deviance, or cross-entropy), or a first order approximation of the latter known as the *Gini Index*.

Impurity Measure

Universal Approximator Like many other hypothesis classes, CART's $\mathcal{F}$ can approximate any function, with large enough M.

A Personal Note While CARTs are very rich hypothesis classes, and easily interpretable, I have poor personal experience with them. I suspect it is their very non smooth nature that is simply inadequate for the problems I have encountered. Then again, the Bagging algorithm, deals with this matter nicely by averaging many trees.

Bagging

Further Reading For more on CARTs and Bagging see [Hastie et al., 2003, Section 9].

Dendogram

As CARTs are essentially decision trees, they can typically viewed as a hirarchy of if-then rules applied on the data. This type of visualization if called a *dendogram*.

Dendogram

3.1.12 Smoothing Splines

The GAM (§3.1.8) model is an example of learning a non-parametric model, since we did not constrain the parametric form of the functions $\{f_j\}_{j=1}^p$. The regularization was part of the optimization algorithm, and not explicit in the problem's definition. *Smoothing Splines* is another non-parametric approach, since we do not assume the parametric form of $\{f_j\}_{j=1}^p$. Unlike GAMs, however, the regularization is indeed explicit in the formulation of the problem. Regularization is achieved by penalizing the second derivative of f, thus forcing f to be smooth. Considering a single predictor, the ERM problem for *Smoothing Spline*

$$\underset{f}{argmin} \left\{ \frac{1}{n} \sum_i (y_i - f(x_i))^2 + \lambda \|\nabla^2 f\|_2^2 \right\}. \tag{3.16}$$

It is quite surprising, and useful, that even though the optimization is performed over an *infinite* dimensional function space, the solution belongs to a *finite* dimensional parametric sub-space. The magic continues! It turns

out that the fitted values, $\left\{\widehat{f}(x_i)\right\}_{i=1}^{n}$ are linear in $\{y_i\}_{i=1}^{n}$. A fact that greatly facilitates the analysis of the statistical properties of these models.

In the presence of $p > 1$ predictors, Eq.(3.16) can be generalized as

$$\underset{f}{argmin}\left\{\frac{1}{n}\sum_{i}(y_i - f(x_i))^2 + \lambda J(\nabla^2 f)\right\}. \tag{3.17}$$

where J is some matrix norm. A natural extension is the squared Frobenius norm[1], which returns a *thin plate spline* as the risk minimizer.

Thin Plate Spline

3.2 Non ERM Supervised Learning

Up until now, we have focused on purely ERM, or hybrid ERM methods. Inductive bias, however, can also be introduced by avoiding the optimization approach of ERM. ERM decouples between the motivation of a method and the particular learning algorithm (ultimately, some optimization algorithm). In the following methods, the learning algorithm is an integral part of the method. This restricts the learnable hypothesis class, thus acting as a regularizer.

Note that in most of the previous sections[2], the restriction of the hypothesis class $\mathcal{F}$ has been imposed by some parametric representation of $\mathcal{F}$, over which optimization is performed. The methods in this section do not impose any such parametrization. They are thus also known as *non parametric*.

Non Parametric

3.2.1 k-Nearest Neighbour (KNN)

The fundamental idea behind the KNN approach is that an observation is similar in y to its surroundings in x. So say I want to classify the type of a bird (y), I merely need to look at the classes of birds which are similar in their attributes (x's). Clearly this requires some notion of distance in the feature space $\mathcal{X}$, as typically all non-parametric methods do[3].

The predicted value of a continuous y at some point x has the form

$$\widehat{y}(x) = \frac{1}{k}\sum_{i\in N_k(x) y_i} \tag{3.18}$$

[1]The Frobenius matrix norm is the sum of squares over all elements: $\|A\|_F^2 = \sum_{i,j} A_{i,j}^2$

[2]Well, bar CARTs and Smoothing Splines

[3]Why is this the case? Well, because non parametric methods replace the idea of optimization in a parameter space, with the idea of similarity in neighbourhoods.

where $N_k(x)$ are the indexes of the k nearest observations to x in $\mathcal{X}$. Eq.(3.18) merely states that we predict y to have the average value of the y_i's in its neighbourhood.

For a categorical y, we would replace the averaging with a majority vote of the classes in its neighbourhood.

Regularization The regularization of the KNN is performed by choice of k.

Universal Approximator KNN is a universal approximator, in that it can recover approximate any f relating x to y.

Sample Complexity While KNN is a universal approximator, it is notoriously known to require many observations to recover even simple relations. With realistic sample sizes, the performance of KNN is typically dominated by other methods.

3.2.2 Kernel Regression

Not to be confused with the Kernel Trick in Kernel SVM, Kernel PCA (§5.5.1), and Kernel Density Estimation (§5.2.2).

Kernel *Regression* is essentially a generalization of a moving average. It is also known as a *scan statistic*, or *searchlight statistic*.

Scan Statistic

The *Nadaraya-Watson* weighted average encompasses most if not all kernel smoothers and is defined as

Nadaraya-Watson

$$\widehat{f}(x) := \frac{\sum_i \mathcal{K}_\lambda(x, x_i) y_i}{\sum_i \mathcal{K}_\lambda(x, x_i)} \tag{3.19}$$

where $\mathcal{K}_\lambda$ is the kernel function, which is merely a measure of the distance between the evaluation point x and the data points $\{x_i\}_{i=1}^n$ and weights the contribution of each $\{y_i\}_{i=1}^n$ to the value at x. The denominator merely ensures the weights sum to 1. Regularization is controlled by some λ parameter.

The moving average in a window of width λ is an instance where $\mathcal{K}_\lambda(x, x_i)$ is fixed in the window. The Band-Pass filter, popular in Electrical Engineering, is an instance of the Nadaraya-Watson smoother, with a Gaussian kernel.

Band Pass

Metric Spaces Clearly, as $\mathcal{K}_\lambda(x, x_i)$ measures distances, it only makes sense if there indeed exists a notion of distance between the x's, which is saying that the feature space $\mathcal{X}$ is a metric space. This can be far from

trivial when dealing with categorical predictors such as countries, genomes, gender, etc. Have no worries however, as even with these type of variables, one can define some notion of distance (not claiming it will prove useful in application).

Relation to KNN There is an intimate relation between KNN and Kernel Regression. Essentially, both predict the value of y at some x by averaging their neighbourhood. Averaging can be weighted or not, in both cases. The important distinction is how neighbourhoods are defined. In KNN, the neighbourhood of x is the k nearest data points. The radius of the neighbourhood thus varies, while the number of data points does not. In Kernel Regression, the neighbourhood of x is all the data points in the support of the kernel. The radius of the neighbourhood is now fixed, while the number of data points can vary.

3.2.3 Local Likelihood and Local ERM

Although presented as two competing concepts, they can augment each other. We have already seen that Kernel Regression plays a part within some ERM algorithms such as GAM (§3.1.8) and PPR (§3.1.9).

Another way to marry the two ideas, is by performing EMR only locally, in each neighbourhood of data points defined by a kernel. This idea is very powerful and leads to, e.g., the LOESS (§3.2.3) and *Local Likelihood* methods.

Local Regression (LOESS)

While not the original historical motivation, we can think of LOESS as a kernel ERM problem. I.e., minimizing the squared loss, in a sliding window, over some very simple hypothesis class, with weighted observations.

It is more flexible than Kernel Regresson. An intuition to this can be gained by thinking of Kernel Regresson as fitting a local regression with an intercept only, while LOESS also allows local polynomials.

It has several attractive statistical properties. In particular, it can also be used for estimation gradients more accurately than by using simple differences. This flexibility however comes at a slight computational cost. It is thus rare to see high dimensional ($p \geq 3$) LOESS fits, even though the theory can easily be generalized to higher dimensions.

3.2.4 Boosting

Boosting is not an ERM problem, but rather an (meta-)optimization scheme due to Schapire [1990]. A *weak learner* is a learning algorithm slightly better than random guessing. The fundamental idea in Boosting is that a *strong learner* is obtained by starting with some weak learner, then training a series of these on the data, while re-weighting the data points inversely to the error, and then taking a weighted average of the series of learners as a (strong) predictor.

The ADABoost algorithm [Freund and Schapire, 1997] is an instance of Boosted when applied to CARTs.

ADABoost

It was initially quite surprising and magical that this heuristic algorithm can produce the great predictors it does. It was later shown that ADABoost can be seen as a greedy, self regularizing, optimization algorithm to solving an ERM with log-likelihood loss [Friedman et al., 2000] over the CART hypothesis class.

For more on Boosting and ADABoost see Chapter 10 in Hastie et al. [2003].

3.2.5 Learning Vector Quantizations (LVQ)

[TODO]

3.3 Dimensionality Reduction In Supervised Learning

We start with finding a low representation of x that predicts y. This practice has several different motivations. As a regularization device avoiding overfitting, and as a computational device reducing memory and time complexity. For a general discussion of dimensionality reduction, see Appendix E.

3.3.1 Variable Selection

A trivial way of reducing the dimension of x is by dropping some of it components. Clearly, with p variables, there are 2^p possible variable combinations. Searching over all possible variable combinations is called *subset selection*, but is rarely performed in practice due to computational complexity. It more practical to perform a *forward search*, i.e., a greedy search where we insert on variable at a time. The search stops when the next variable does not improve the model's accuracy. While tempting, measuring the accuracy

Subset Selection

using the empirical risk will certainly lead to overfitting (see Figure 4.1). We can thus drive the search with a CV scheme, but the computational burden might be overwhelming. Using some estimator of R auch as AIC, BIC,... is a reasonable and practical way to guide the search.

Algorithm 1 Forward Search

Pick your favourite model selection criterion $\widehat{R}(x)$.
while $\widehat{R}(x)$ keeps getting smaller **do**
    $\widehat{f} \leftarrow$ the model with the smallest $\widehat{R}(x)$ after adding a *single* variable.
end while
return $\widehat{R}(x)$.

Remark 3.3.1 (Hypothesis Testing Driven Variable Selection)**.** For a statistician , it is probably very natural to guide a forward search with hypothesis testing. I.e., at each iteration, insert the variable with the most significance. Note however, that when interested in good predictions, we might actually gain accuracy by dropping a significant variable. This is a classical bias-variance tradeoff: dropping a true predictor adds bias, but reduces the variance as we have one less parameter to estimate. If the true effect is small, we can gain from this tradeoff. This intuition is actually used as a model selection criterion [Foster and Stine, 2004]. The bottom line is however, that a **significant variable, does not make it a good predictor.**

3.3.2 LASSO

We presented the LASSO in Section 3.1.4 as a regularized ERM problem. We claimed, without proof, that the l_1 regularization in fact performs soft coefficient thresholding. The estimated $\widehat{\beta}_{LASSO}$ has thus many entries set to zero, acting as a model selection device and reducing the dimension of the problem.

3.3.3 Principal Component Regression (PCAR)

A very general framework to reduce the dimension of a supervised learning problem, is to find a low dimensional representation of the features X, and use that representation as the new features. By combining the simplest building blocks of supervised and unsupervised learning, namely, OLS an PCA, we get PCA regression.

PCA regression is a very simple idea that works as follows:

Algorithm 2 PCA Regression

$X_q \leftarrow$ A rank q approximation of X using PCA.
$\hat{\beta}_{PCAR} \leftarrow$ OLS estimates of β using y and X_q.
return $\hat{\beta}_{PCAR}$

Remark 3.3.2 (PCAR and Ridge Regression)**.** It turns out there is an intimate relation between PCAR and Ridge Regression. We refer the reader to Section 3.6 in Hastie et al. [2003].

3.3.4 Partial Least Squares (PLS)

Thinking of PCAR (§3.3.3), there is no guarantee that directions that capture the variability in $\mathbf{x}$, are the same one capturing information about $\mathbf{y}$. The decoupling between the dimensionality reduction and the supervised learning may thus seem unjustifiable. LASSO (§3.1.4) is a possible remedy to this matter. PLS is a different remedy, motivated by PCAR. In PLS, we marry the ERM problem of OLS and PCA. The m'th PLS combination of X's solves [Hastie et al., 2003, Eq.3.64]

$$\max_{\|\alpha_m\|_2=1, \alpha_m S \alpha_l=0 \text{ for } l=1,\ldots,m-1} \left\{\rho^2(y, X\alpha_m) Var(X\alpha_m)\right\}. \tag{3.20}$$

We can thus conclude that PLS embeds X in a linear subspace which aims at balancing a compact description of X (the variance term), and a good prediction of y (the correlation term).

3.3.5 Canonical Correlation Analysis (CCA)

We next consider the case of vector valued $\mathbf{y}$, with K entries. This is the case of predicting several values at once. If these values are unrelated, the problem amounts to solving several separate learning problems. If these values are related, there may be an accuracy gain by solving them together.

For an intuition to the reason that pooling problems together may be beneficial, consider the case of K repeated measures of the same outcome. We would clearly want to somehow pool these measures together when learning a predictor.

The concept of vector valued outcomes is not new in the statistical literature. Indeed, this is precisely the topic of interest in the vast literature on Multivariate Analysis of Variance (MANOVA, e.g. Anderson [2003]). CCA focuses on a particular aspect of MANOVA: learning the relation between some low dimensional representations of $\mathbf{x}$ and $\mathbf{y}$.

[TODOTODO]

3.3.6 Reduced Rank Regression (RRR)

[TODO]

3.4 Generative Models In Supervised Learning

For the mere purpose of making a prediction, we do not need to learn the full data generating distribution $P(y, x)$. Knowing this distribution, however, does permit to make predictions, via Bayes Theorem: $P(y|x) = \frac{P(y,x)}{\int P(y,x)dy}$. Several supervised (and unsupervised) methods make use of this relation to make predictions. These are known as *generative models* . Since the generative distribution serves both in supervised and unsupervised learning, we have dedicated it a section of its own (§D).

3.4.1 Fisher's Linear Discriminant Analysis (LDA)

The fundamental idea behind *Fisher's LDA* (sometimes, just LDA) is that for dichotomous y's: $P(x|y)$ is multivariate Gaussian, with the same covariance for all y. Estimating $P(x|y)$ thus amounts to the estimation of the mean and covariance, which are fairly simple problems that do not require too much data (relatively).

The decision boundaries of this method turn out to be linear in $\mathcal{X}$. Denoting the distance from the class centers by $\widehat{\delta} := (\widehat{\mu}_1 - \widehat{\mu}_2)$, class thus 2 will be predicted if

$$x'\widehat{\Sigma}^{-1}\widehat{\delta} > \frac{1}{2}\widehat{\delta}'\widehat{\Sigma}^{-1}\widehat{\delta} \tag{3.21}$$

and class 1 otherwise.

Remark 3.4.1 (LDA and OLS classification)**.** Interestingly, the two-class LDA, is the same as a regression classifier from Section 3.1.6 [Hastie et al., 2003, Eq. 4.11].

3.4.2 Fisher's Quadratic Discriminant Analysis (QDA)

If you are uncomfortable with the fixed covariance assumption of LDA, one can relax thus assumption. This leads to QDA, which is more flexible than LDA, requiring slightly more data, but permitting the learning of a class of quadratic classifiers, and not only linear.

3.4.3 Naïve Bayes

In LDA (§3.4.1) we dealt with the problem of estimating high dimensional distributions by adding the class-wise Gaussianity assumption. As Gaussians as fairly simple to learn from data, if there is any truth to this assumption, our life improved. Another life-simplifying assumption, replacing the Gaussianity assumption, is that $P(x|y)$ is the product of the margins, i.e., the the features are independent within each class: $P(x|y) = \prod_{j=1}^{p} P(x_j|y)$ This greatly simplifies things as estimating the univariate marginal distributions $P(x_j|y)$ is a fairly simple problem, which can be done non-parametrically by Kernel smoothing for continuous predictors (§3.2.2), and simple relative frequencies for discrete predictors.

3.5 Ensembles

This section follows the lines of Chapter 8 in Hastie et al. [2003].

Our inductive bias constrains the hypothesis class so that we do not overfit the data. It is overly optimistic to think that nature generates samples in accord with our own inductive bias. Ensemble learning is a "meta" class of algorithms, in which we do not confine ourselves to selecting a single f but rather to selecting (finitely) many, possibly from different classes, and them aggregating them into a single predictor. This is a very flexible framework that might actually return classifiers that do not belong to any of underlying hypothesis classes. As such, it is an *improper learning* algorithm..

Improper Learning

One might consider many combinations of hypothesis classes, learning methods and aggregation schemes. Several celebrated algorithms fall in this class.

Remark 3.5.1. As opposed to a *strong learner*, a *weak learner* is a learning algorithm that is only partially optimized for the observed data. Some of the above algorithms are motivated by the idea of combining many weak learners from the same $\mathcal{F}$ to return a strong learner. Such is the Bagging algorithm. It has been noted, however, that it is preferable[4] to combine strong learners from different hypothesis classes, then to combine weak learners from the same hypothesis class Gashler et al. [2008].

Weak/ Strong Learner

3.5.1 Committee Methods

Considering a set of M candidate learning algorithms, a *committee method* is simply an average of the M predictors:

[4]Statistically, not necessarily computationally.

Algorithm 3 Commitee Methods

For M candidate learning algorithms.
for $m \in 1, \ldots, M$ **do**
  $\widehat{f}^m(x) \leftarrow$ the predictor learned with the m'th algorithm.
end for
$\bar{f}(x) \leftarrow$ the average of $\widehat{f}^m(x)$.
return $\bar{f}(x)$

This concept can be generalized by considering different aggregation schemes. The following (meta-)algorithms are merely different aggregation schemes. Most, if not all, can be seen as minimizers of some posterior expected loss. I.e., they minimize some risk with respect to the posterior probabilities of $\left\{\widehat{f}^m(x)\right\}_{m=1}^M$.

3.5.2 Bayesian Model Averaging

The idea of *Bayesian model averaging* is that of using the posterior mean as a predictor. Computing the posterior mean can be quite a tedious task. When dealing with hypotheses from the same parametric class, a more practical approach is that of *inverse BIC weighting*. This is justified by that fact that the BIC (§4.2) is approximately proportional to the posterior probability of $\widehat{f}^m(x)$.

Algorithm 4 Model Averaging

For M candidate learning algorithms.
for $m \in 1, \ldots, M$ **do**
  $\widehat{f}^m(x) \leftarrow$ the predictor learned with the m'th algorithm.
  $BIC^m \leftarrow$ the BIC of $\widehat{f}^m(x)$.
end for
$\bar{f}(x) \leftarrow$ the average of $\widehat{f}^m(x)$ inversely weighted by BIC^m.
return $\bar{f}(x)$

BIC is used as weights as it approximates the posterior

3.5.3 Stacking

A simple and powerful (meta-)algorithm by which you train several predictors, and then use their predictions as features for a new predictor. A built-in Jackknifing (or Cross Validation) ensures over-fit models are not overweighted. Here is stacking example for a continuous y:

Algorithm 5 Stacking

For M candidate learning algorithms.
for $i \in 1, \ldots, n$ **do**
  for $m \in 1, \ldots M$ **do**
    $\widehat{f}_m^{(i)}(x) \leftarrow$ the predictor learned with the m'th algorithm, and without the i'th observation.
  end for
  $\bar{f}(x_i) \leftarrow$ the OLS prediction using $\left\{\widehat{f}_m^{(i)}(x_i)\right\}_{m=1}^{M}$ as features.
end for
return $\bar{f}(x)$

3.5.4 Bootstrap Averaging (Bagging)

Generates bootstrap samples and averages the learned predictors (see Algorithm 6). While seemingly completely frequentist, the Bootstrap average can be seen as an estimator of the posterior mean for some non-parametric prior.

A *random forrest* is such an algorithm, when CARTs are fitted and averaged.

Random Forrest

Algorithm 6 Bagging

Choose the number Bootstraps B.
for $b^* \in 1, \ldots, B$ **do**
  Generate a bootstrap sample of size n: $\mathcal{S}_{b^*}$.
  Learn $\widehat{f}(x)_{b^*}$ from $\mathcal{S}_{b^*}$.
end for
$\bar{f}(x) \leftarrow$ the average of $\widehat{f}(x)_{b^*}$.
return $\bar{f}(x)$

3.5.5 Boosting

The Boosting algorithm in Section 3.2.4 might have the flavour of an Ensemble method, but it should be seen as a regularization scheme. Moving on...

Chapter 4

Statistical Decision Theory

This section follows the spirit of Section 7 in Hastie et al. [2003], up to some changes in notation.

In Section 3.1, we gave an intuitive argument for which without some inductive bias, learning will return models with poor performance on new data. In this section we learn how to quantify the performance of a model. In particular, when given new data. This allows us to select among competing candidate models. It will also allow us to choose the value of the regularization parameter of each method.

Figure 4.1 demonstrate the prediction error (red curve) of some model as the model complexity increases. As can be seen, the prediction error decreases as the model becomes more complex, but saturates at some point. This is because the reduction in the bias is smaller than the increase in variance of learning very complex models. This is the celebrated bias-variance tradeoff.

Bias
Variance
Tradeoff

Once we are able to estimate the prediction error from our data, we will seek for a model which minimizes this error.

Before we proceed, we now need to distinguish between several types of prediction errors. The population *risk* of a model parametrized by θ, was previously defined as the average loss over all possible data instances, and denoted by $R(\theta)$ (§2.4). The empirical risk was defined as the average loss over the observed data points, and denoted by $\mathbb{R}(\theta)$. We now update these

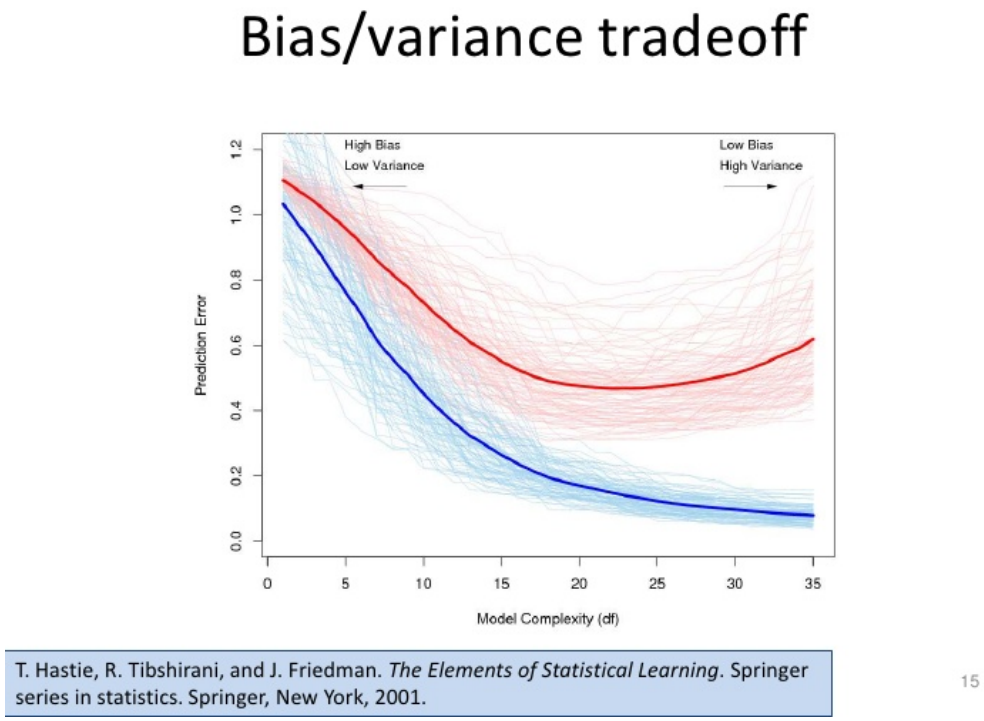

Figure 4.1: Overfitting: Prediction error on new data (red curve) versus the empirical prediction error (light blue). The empirical prediction error will always decrease as more complicated models are fit (moving right). The prediction error on new data, however, will not always decrease and will typically show a local minima.

definitions to deal with the the $f(x)$ notation of the previous section.

$$R(f) := \mathbf{E}_{\mathbf{Y},\mathbf{X}}\left[\mathbf{l}(\mathbf{Y}, \mathbf{f}(\mathbf{X}))\right], \tag{4.1}$$

$$\mathbb{R}(f) := \mathbb{E}\left[l(y, f(x))\right] = \frac{1}{n}\sum_i l(y_i, f(x_i)), \tag{4.2}$$

$$\bar{R}(f) := \frac{1}{n}\sum_i \mathbf{E}_{\mathbf{Y}}\left[\mathbf{l}(\mathbf{Y}, \mathbf{f}(\mathbf{x_i}))\right], \tag{4.3}$$

$$R(\hat{f}_n) := \mathbf{E}_{\widehat{\mathbf{f}}_\mathbf{n}}\left[\mathbf{E}_{\mathbf{Y},\mathbf{X}}\left[\mathbf{l}(\mathbf{Y}, \widehat{\mathbf{f}}_\mathbf{n}(\mathbf{X}))|\widehat{\mathbf{f}}_\mathbf{n}\right]\right]. \tag{4.4}$$

Eq.(4.1) is merely a reformulation of $R(\theta)$ from Section 2.4. It captures the expected loss, a given predictor, $f(X)$, will incur on average when given new X's and Y's. This will be the magnitude which will tell us which models

perform well, and which do not. It is known as the *test error* or also as *prediction error*.

Test Error

Eq.(4.2) is the reformulation of empirical risk, $\mathbb{R}(\theta)$, we have been optimizing in Section 3.1. We referred to it as the *empirical risk*, but it is also known as the *train error*.

Train Error

Eq.(4.3) is the average risk at the observed x's, when given new Y's [1]. This is the *in sample error*.

In Sample Error

Eq.(4.4) is called the *expected prediction error*, i.e., the expected loss when f is also re-learned. Put differently: How much would we err when:(1) we are given n new examples $\mathcal{S}_1$; (2) re-learn $\widehat{f}_n$ on $\mathcal{S}_1$; (3) compute the risk of $\widehat{f}_n$ (in the population, not in $\mathcal{S}_1$. We emphasize this by writing $\widehat{f}_n$ instead of f. $R(\hat{f}_n)$ is thus not a property of a particular predictor f, but rather of a whole learning algorithm on random samples of size n. It could have also been written as $R(algorithm)$, although I have not seen this notation in use.

Expected Prediction Error

We would like to compare the performance of models based on $R(f)$, as this will give us an idea on the quality of the prediction on new data. Alas, computing $R(f)$ requires the distribution of y and x, while we only have access to the n observed samples. Can the empirical risk $\mathbb{R}(f)$ estimate the unknown risk $R(f)$? Figure 4.1 suggests it cannot since $\mathbb{R}(f)$ underestimates $R(f)$. Why is this? At an intuitive level: this is because with ERM we learn the f with smallest error in each sample. It is thus the same as estimating the expected height in a population, by using the minimum in each sample; we will clearly be underestimating the expectation. Then again, there is the hope that we may take this minimum and debias it. This is the goal in the next sections.

Before proceeding, we distinguish between two similar tasks:

Model Selection This is the task of selecting between several candidate models.

Model Assessment This is the task of assessing the prediction error (i.e., the expected loss, the risk) of a given model.

4.1 Train, Validate, Test

If data is abundant, a trivial, assumption free way to estimate $R(f)$[2], is to split the data into 3 sets. A *training set*, used to learn several competing

[1] This magnitude should not be unfamiliar: e.g., inference in ANOVA is performed conditional on the x's, which typically stem from a designed experiment.

[2] Think: why $R(f)$ is being estimated, and not $R(\hat{f}_n)$ nor $\bar{R}(f)$?

models. A *validation set*, used check the performance of the learned models and choose the best performer using some comparison measure. A *test set*, used to estimate the risk, as the empirical risk $\mathbb{R}(f)$ will be unbiased to the population risk $R(f)$.

If there is not enough data for this scheme, keep reading...

4.2 Unbiased Estimators of the Risk

Under appropriate assumptions, the bias in $\mathbb{R}(f)$ when estimating $\bar{R}(f)$[3] can be computed analytically, and accounted for. The bias $\bar{R}(f) - \mathbb{R}(f)$ is called the *optimism* of the algorithm. Akaike's Information Criterion (AIC), the finite sample Corrected AIC (AICc), Mallow's Cp (Cp), the Bayesian Information Criterion (BIC, aka SBC, aka SBIC), the Minimum Description Description Length (MDL), Vapnic's Structural Risk Minimization (SRM), the Deviance Information Criterion (DIC), and the Hannan-Quinn Information Criterion (HQC), all try to estimate $\bar{R}(f)$ by correcting for the optimism under different assumptions.

Optimism

Cp, AIC, BIC, MDL, SRM

The differences, pros, and cons, of each will not be discussed herein. Just remember what they mean when you see them in your favourite software (R!). They all have in common that you will want the model with the smallest criterion. But be careful- as they are used for model selection, they are indifferent to scaling, and thus should be not interpreted as the expected prediction error.

Remark 4.2.1. Not all model selection criteria estimate $\bar{R}(f)$. The Focused Information Criterion (FIC), for example, does not.

Further Reading For a brief review of AIC, BIC, MDL and SRM see Chapter 7 in [Hastie et al., 2003]. For a more rigorous derivation, see Claeskens and Hjort [2008].

4.3 Jackknifing

If concerned with over fitting, here is a simple algorithm to estimate the prediction error:

[3] In this case, note that it is $\bar{R}(f)$ being estimated, and not $R(f)$ nor $R(\hat{f}_n)$.

Algorithm 7 Jackknife

for $i \in 1, \ldots, n$ **do**
  $\widehat{f}^{(i)} \leftarrow$ the learned model with all but the i'th observation.
  $l^{(i)} \leftarrow$ the loss of $\widehat{f}^{(i)}$ on the i'th observation.
end for
return the average loss over $l^{(i)}$.

This process is called the *Jackknife*, or *Leave-One-Out–Cross-Validation*. This algorithm return an estimator of $R(\hat{f}_n)$. This might be quite surprising: every split uses almost an identical sample, so why would it not estimate $R(f)$? See Section 7.12 in Hastie et al. [2003] for details..

But wait! We might be able to stabilize the variability of the estimated error in every split, if instead of leaving only a single observation aside, we leave some more. This lead to way to *K-Fold Cross Validation* in the next section.

4.4 Cross Validation

Algorithm 8 Cross Validation

Split the data into K parts ("folds").
for $k \in 1, \ldots, K$ **do**
  $\widehat{f}^{(k)} \leftarrow$ the learned model with all *except* the observations in the k'th fold.
  $l^{(k)} \leftarrow$ the loss average of $\widehat{f}^{(k)}$ on the observations in the k'th fold.
end for
return the average over $l^{(k)}$.

This simple algorithm estimates $R(\hat{f}_n)$ without any assumption on the data generating process, and less data than would be required for a "train-validate-test" scheme. Well, as it actually serves for model selection, it should be seen as a "train-validate" scheme, without the "test" part. It is thus *not* an unbiased estimate of $R(\hat{f}_n)$. See Section 7.12 in Hastie et al. [2003] for details.

But wait again! The Cross Validation scheme resamples the data *without replacement* to estimate $R(\hat{f}_n)$. Could we have sampled it *with* replacement? Yes. This is the idea underlying the *Bootstrapping* scheme.

4.5 Bootstrapping

Here is the simplest version of Bootstrap validation:

Algorithm 9 Bootstrap

for $b^* \in 1, \ldots, B$ **do**

$\mathcal{S}^{b^*} \leftarrow n$ randomly selected observations, with replacement, from the original data.

$\widehat{f}^{b^*} \leftarrow$ the model learned with $\mathcal{S}^{b^*}$.

$l^{b^*} \leftarrow$ the average loss of $\widehat{f}^{b^*}$ on the observations in the *original* data.

end for

return the over of l^{b^*} .

This algorithm is not a good estimator of $R(\hat{f}_n)$ as observations play a role both in learning and in validating. Several corrections are available. For details see Section 7.11 in Hastie et al. [2003].

The Bootstrap is a very general scheme, which can be used not only for model validation, but for assessing many of its statistical properties. It is possibly best known when used for hypothesis testing. For more on the Bootstrap, see Efron and Tibshirani [1994].

Chapter 5

Unsupervised Learning

5.1 Introduction to Unsupervised Learning

Unlike Supervised Learning, in Unsupervised Learning there is no outcome variable y. Unsupervised learning is thus not a task per-se, but rather a learning setup, which may include many different tasks:

1. **Density Estimation**: Estimate $P(x)$.
2. **High Density Regions**: Find feature combinations which tend to concentrate, hopefully, because the belong to homogenous and interpretable subgroups.
3. **Visualize**: Visually inspect the data.
4. **Cluster**: Assign observation to homogenous groups, i.e., clusters. Alternatively, assign not only observations, but also unobserved values. I.e., learn a *clustering function*.

When thinking of machine learning as the means to automate decision processes, then unsupervised learning is typically as a pre-processing stage before a supervised learning.

We will see that the building blocks presented in the introduction (§1) will make an appearance in practically every task and very algorithm. Namely, dimensionaly reduction (Appendix E), the kernel trick (Appendix B), the spectral trick (Apendix C) and generative approaches (Appendix D).

Graph Method or Feature Methods? As we will see, some methods only require a similarity graph ($\mathfrak{S}$), while other require the underlying features (X). What is better? Well, there is no single answer to the question. Here are some pros and cons:

1. Computational complexity: Storing X requires $n \times p$ memory units[1]. Storing $\mathfrak{S}$ will require $n \times n$ memory units. Most algorithms have *streaming* versions. Meaning that the solutions can be iteratively obtained by processing only parts of X or $\mathfrak{S}$ at a time. The computational complexity of each method can thus vary depending on n,p, and the particular algorithm implementing the method. No approach dominates the other.

   Streaming Algorithm

2. Data reconstruction or space reconstruction: Methods taking $\mathfrak{S}$ inputs, return an embedding. They thus apply only to the observed points. Methods taking $\mathcal{X}$ typically return an embedding *function*. The result can thus effortlessly be applied to new points. Put differently, they embed the whole $\mathcal{X}$ into $\mathcal{M}$ and not only X. Exceptions do exist. PCA (§5.4.1) for instance, can take $\mathfrak{S}$ and return an embedding function.

Remark 5.1.1 (Unsupervised Learning in the ERM framework)**.** Many unsupervised learning problems, can be cast in the ERM framework. We currently do not call it ERM, but essentially every optimization problem we will see can be thought of as the minimization of some loss, over some hypothesis class. This has not been done in the current version of our text due to both time constraints, and compatibility with the referenced literature.

5.2 Density Estimation

We now aim at the possibly hardest target in unsupervised learning- estimating $P(x)$. As previously stated, learning P is of interest for itself, but can also serve for classification (§3.4), for detecting high density regions (§5.3), and for clustering (§5.8).

5.2.1 Parametric Density Estimation

If a parametric generative model can be assumed, this collapses to the parameter estimation problem in the classical statistical literature (§2). Maximum Likelihood estimation being a particularly attractive approach, but certainly not the only one. If a parametric model cannot be assumed, we fall into the realm of non-parametric methods. As we saw for supervised learning, these typically rely on pooling information from neighbourhoods of $\mathcal{X}$.

[1] I hope that readers familiar digital storage will forgive my non-rigorous treatment of storage modes.

5.2.2 Kernel Density Estimation

Much like the Kernel Regression regression, a naïve estimator is the moving average. A natural generalization, in the spirit of the Nadaraya-Watson smoother (Eq.(3.19)) is the *Parzen* estimate:

Parzen Estimate

$$\widehat{p}(x) := \frac{1}{n\lambda} \sum_{i=1}^{n} \mathcal{K}_\lambda(x, x_i). \tag{5.1}$$

Remark 5.2.1. If you have been convinced by the use of KNN (§3.2.1) for regression (or classification), there is no reason not to use it for density estimation. It will keep offering the same pros and cons as in KNN regression (or classification).

As previously stated, these methods may fail when x are high dimensional. We thus recur to other methods.

5.2.3 Graphical Models

Graphical models are graphs that represent joint distributions[2] For our purposes, we discuss only *undirected graphical models*, also known as *Markov random fields*, or *Markov networks*. The reason these models are interesting, and particularly for the purpose of density estimation, is that missing edges in a graphical model represents conditional independencies between random variables. If the graph structure can be assumed, and hopefully has many missing edges, then the dimension of the density estimation problem is considerably reduced.

Markov Random Field

Example 5.2.1 (First Order Univariate Markov Process)**.** Consider a Gaussian random vector $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma) \in \mathbb{R}^p$. To see how the graphical model, i.e. the Markov assumption, helps us with the estimation let's count the number of estimated parameters. Without any further assumptions, there are p parameters in μ and $p(p+1)/2$ parameters in Σ to estimate. This corresponds to a fully connected graph of the distribution. If, however, we can assume a first order Markov structure, implying that the graph of the distribution is merely a chain, there are less parameters to estimate: p parameters in μ and $p + (p-1)$ parameters in Σ^{-1}. All other entries in Σ^{-1} vanish. Say $p = 10$, then the first order graphical model has 29 parameters, compared to 65 in the fully connected graphical model.

[2] Unlike *graph data*, where the graph is the data itself, and not a distribution.

In the cases the graph structure cannot be assumed, then our goal becomes the learning of the graph structure from the data. I.e., learning the (conditional) independencies between the variables. For a multivariate Gaussian distribution, this amounts to estimating the covariance, while imposing some sparsity assumption. An important fact in this regard is that in the multivariate Gaussian case, missing edges in the graphical model, mean zero entries in the inverse covariance matrix. This fact sets the bridge between learning the structure of a graphical model, and (sparse) covariance estimation.

Remark 5.2.2 (Restricted Bolzmann Machine)**.** In this context it is also worth mentioning the Restricted Boltzmann Machine (RBM), which is a multivariate distribution with particular class of graphical structure, motivated from Neural Networks.

Restricted Bolzmann Machine

For a more detailed review of Graphical Models, see Chapters 17 and 20 in Wasserman [2004].

5.3 High Density Regions

When data is high dimensional a full blown density estimation may be statistically and computationally impractical. We can still, however, aim at detecting regions in $\mathcal{X}$ with high probability (which would have been trivial have we had a density estimate).

Once high density regions have been detected, they could be assigned to clusters. Then again, assigning to clusters is typically an easier problem which can be solved directly (see §5.8).

5.3.1 Association Rules

Association rules, or *market basket analysis*, or *affinity analysis*, can be seen as approximating the joint distribution with a region-wise constant function (Eq. 3.15). Put differently, we want to capture high density regions of the joint distribution of x with by approximating it with a decision-list (not tree). Learning a decision-list is a computationally impractical problem in general. Association rules are thus typically learned over binary feature spaces $\mathcal{X}$, using heuristic optimization schemes.

Market Basket Analysis

This type of problems typically occurs in sales analysis, where vendors seek for combinations of products that tend to sell jointly (so that can design the store better, or discount product bundles).

The *Aprioiri* algorithm Agraval and Srikant [1994], is an example of such a heuristic search for high density combinations.

Apriori Algorithm

5.4 Linear-Space Embeddings

Linear space embeddings are a class of dimensionality reduction techniques that map the data X into a lower dimensional *linear* space $\mathcal{M}$. The mapping itself, $f : \mathcal{X} \to \mathcal{M}$ can be linear or non linear. We denote the low dimensional representation of the data by $\hat{X} := f(X) \in \mathcal{M}$.

The idea of ERM and Inductive Bias also applies to unsupervised learning. We seek some f that does not incur too much loss, on average. I.e., we seek to minimize $R(f)$. As usual, we do not have access to the data generating process of x, so we typically content ourselves with the empirical risk minimization. In the context of unsupervised learing, the empirical risk is known as the *reconstruction error*.

Reconstruction Error

For a general discussion of the idea of dimensionality reduction see Appendix E.

Many dimensionality reduction methods, if not all, take inspiration from the origins of PCA. It is thus strongly recommended to read the PCA section (§5.4.1) before proceeding.

Remark 5.4.1 (Interpreting "Linear")**.** Two interpretations of "linear" can be found in the literature. It may refer to the nature of the low dimensional space approximating the data, or to the nature of the embedding operation. In this text, we use the first interpretation, meaning that $\mathcal{M}$ is a linear subspace, even if the embedding operation is non linear.

Remark 5.4.2 (Interpreting the Low Dimensional Representation)**.** I we are not only interested in a compressed representation of our data, but also in assigning some intuitive meaning to this compressed representation, we need to be cautious. An example of the difficulties that may arise try comparing the PCA problem (§5.4.1) with the factor analysis problem (§5.6.1). For interpretation purposes, it may be preferable, in the spirit of statistical inference, to assume some interpretable generative model and approach the dimensionality reduction as a statistical inference problem. This is the theme of the Latent Space Generative Models section (§5.6).

5.4.1 Principal Components Analysis (PCA)

Task	Type	Input	Output	Concept	Remark
dim reduce	optimization	$\mathfrak{S}$	embedding function		

PCA is such a basic technique, it has been rediscovered and renamed independently in many fields. It can be found under the names of *discrete Karhunen–Loève Transform; Hotteling Transform; Proper Orthogonal Decomposition (POD); Eckart–Young Theorem; Schmidt–Mirsky Theorem; Empirical Orthogonal Functions; Empirical Eigenfunction Decomposition; Empirical Component Analysis; Quasi-Harmonic Modes; Spectral Decomposition; Empirical Modal Analysis;* and possibly more[3]. The many names are quite interesting as they offer an insight into the different problems that led to its (re)discovery.

Starting with an example, consider human height and weight data. While clearly two dimensional data, you don't really need both to understand how "big" are the people in the data. This is because, height and weight vary mostly along a single dimension, which can be interpreted as the "bigness" of an individual. This is why, physicians use the Body Mass Index (BMI) as an indicator of size, instead of a two-dimensional measurement. Assume you now wish to give each individual a size score, that is a linear combination of height and weight: PCA does just that. It returns the linear combination that has the most variability, i.e., the combination which best distinguishes between individuals.

The variance maximizing motivation above was the one that guided Hotelling [Hotelling, 1933]. About 30 years before him, Karl Pearson [Pearson, 1901] derived the same procedure with a different motivation in mind. Pearson was also trying to give each individual a score. He did not care about variance maximization, however. He simply wanted a small set of coordinates in some (linear) space that approximates the original features well. As it turns out, the best linear-space approximation of X is also the variance maximizing one. Pearson and Hotelling thus arrived to the same solution, with different motivations.

Terminology PCA has received much attention. As such, it has rich underlying theory and terminology. Here are some terms needed to understand PCA outputs:

- **Principal Components**: The linear combinations of the features, which best separate between observations. In our example- the "bigness" index of each individual. The first components captures the most variance, the second components, the second most-variance, etc. In terms of $\mathcal{M}$, the principal components are an orthogonal basis for $\mathcal{M}$.
- **Scores**: Synonymous to Principal Components.

[3]Wikipedia: `http://en.wikipedia.org/wiki/Principal_component_analysis`

- **Loadings**: The weights of each data point in each principal component. In our example, the importance of the height and weight in constructing the "bigness" score.

Intuition

Notice we have currently offered two motivations for PCA: (i) Find linear combinations that best distinguish between observations, i.e., maximize variance. (ii) Find the linear subspace the bets approximates the data. The reason these two problems are equivalent, is due to the use of the squares error. Informally speaking, the data has some total variance. This variance can be decomposed into the part captured in $\mathcal{M}$, and the part not captured[4]. Since the variance in the data consists of sums of squares, minimizing the distance from X to $\mathcal{M}$, is the same as maximizing the variance of $X \hookrightarrow \mathcal{M}$, since their sum is fixed.

Mathematics of PCA

We now present the derivation of PCA from the two different motivations.

The Variance Maximization View Starting with the first principal component. If $\mathbf{C}ov[x] = \Sigma$ then for a fixed vector v: $\mathbf{C}ov[vx] = v\Sigma v'$. Maximizing w.r.t. v does not make sense as $\mathbf{C}ov[vx]$ will explode. It is most convenient, mathematically, to constrain the l_2 norm: $\|v\|_2^2 = 1$. Maximizing under a constraint, using Lagrange-Multipliers:

$$\underset{v}{argmax} \left\{ v\Sigma v' - \lambda(\|v\|_2^2 - 1) \right\}. \tag{5.2}$$

Differentiating w.r.t v and equating zero:

$$(\Sigma - \lambda I)v' = 0 \tag{5.3}$$

So the P solutions for v are the eigen-vectors of Σ. Which of them to pick? To find a *global* maximum we return to the original problem, as plug our result:

$$\underset{v:\|v\|_2^2=1}{argmax} \{v\Sigma v'\} = \underset{\lambda}{argmax} \{v\lambda v'\} \tag{5.4}$$

so that the global maximum is obtained with the largest eigen-value λ.

[4] Analogous to $SST = SSR + SSE$ in linear regression.

Readers familiar with matrix norms will recognize that this is simply the derivation of the operator norm of Σ.

The second principal component can be found by solving the same problem, with the additional constraint of v_2 orthogonal to v_1.

The last missing ingredient is that instead of the true covariance between the features, Σ, we use the (scaled) empirical covariance $X'X$.

The Linear-Space Embedding View We now seek to find a sequence of p approximations to X that lay in $1, \ldots, p$ dimensional linear subspaces, with respect to a least squares loss. For simplicity of exposition, we will assume that X has been mean centred. The q'th problem to solve is thus

$$\underset{f_q}{argmin} \left\{ \|X - f_q(X)\|_{Frob} \right\}. \tag{5.5}$$

Since f_rank is a map from $\mathbb{R}^p$ to some rank-q linear subspace, it must have the form $f_q(X) = H_q X$ where H_q is a $n \times n$ matrix of rank q. Since Eq.(5.5) minimizes sums of (squared) Euclidean distances, H_q has to be an orthogonal projection, thus symmetric. As such it can decomposed into an outer product $H_q = V_q V_q'$ where V_q is full rank $n \times q$ matrix [Meyer, 2001, Eq.(5.13.4)]. Under the q-space constraint, and squared error, Eq.(5.5) collapses to

$$\underset{V_q}{argmin} \left\{ \|X - V_q V_q'(X)\|_{Frob} \right\}. \tag{5.6}$$

Using some algebraic identities [Shalev-Shwartz and Ben-David, 2014, Eq.(23.3)] Eq.(5.6) is equivalent to

$$\underset{V_q}{argmax} \left\{ \mathrm{Tr}(V_q' X X' V_q) \right\}. \tag{5.7}$$

At this point we should note that the linear-space embedding problem has collapsed to the variance maximization problem! If you do not see this, just set $q = 1$ and compare to Eq.(5.4), recalling that $X'X$ estimates the features' covariance Σ.

PCA as a Graph Method

Starting from the maximal variance motivation, it is perhaps not surprising that PCA depends only on the similarities between features, as measured by their empirical covariance. The linearity of the target manifold was there by assumption.

Following the linear-space embedding motivation, it is was surprising that the solutions depend only on the empirical covariances. This fact can be

attributed to the use of squared error loss, which implied we were trying to decompose the total variance into the part in $\mathcal{M}_q$ and the orthogonal part.

From both motivations we see that the values of X are of no importance given $X'X$, which can be informally thought of as a sufficient statistic[5].

In-turn, $X'X$ depends only on the empirical covariances between *individuals* ($\mathfrak{S} = XX'$), or on the Euclidean distances between individuals ($\mathfrak{D} = (\|x_i - x_j\|)$).

The building blocks of all these graph-based dimensionality reduction methods are:

1. Compute some similarity graph $\mathfrak{S}$ (or dissimilarity graph $\mathfrak{D}$) from the raw features.

2. Call upon graph embedding theory to map the data points into the target manifold $\mathcal{M}$.

The fact that the linear-space embedding of the data depends only some similarity graph has laid a bridge between feature embedding, such as PCA, and *graph embedding* methods such as MDS (§5.4.4). Moreover, it has opened the door for replacing the covariance similarity, with many other similarity measures. Classic MDS (§5.4.4) is simply PCA when starting from $\mathfrak{S}$, thus viewed as a graph embedding problem. kPCA (§5.5.1) plugs kernel similarities (§B) instead of covariance similarities. Isomap (§5.4.6), LocalMDS (§5.4.5), and LLE (§5.5.4) follow a similar motivation using *local* measures of similarity. Spectral Clustering (§5.8.5) does some linear-space embedding à-la PCA, then wrapping up with a clustering algorithm in $\mathcal{M}$ à-la K-means.

We now prove that the PCA solution can be cast in terms of the covariance between individuals ($\mathfrak{S} = XX'$) or the Euclidean distances ($\mathfrak{D} = \|x_i - x_j\|$). In particular, we show that all the information on the location (mean) of X, needed for the PCA reconstruction, is actually encoded in $\mathfrak{S}$ (or $\mathfrak{D}$).

The following exposition takes from [Hastie et al., 2003, Section 18.5.2]

PCA with the Covariance Similarity Graph To begin, we need to cast the solution to the PCA problem in Eq.(5.7) using the Singular Value Decomposition (SVD). SVD

Definition 5.1 (SVD)**.** Any $n \times p$ matrix X, can be decomposed into $X = UDV'$ where U is an $n \times p$ orthogonal matrix ($U'U = I_p$); D is a $p \times p$ diagonal matrix with diagonal elements $d_1 \geq d_2 \geq \cdots \geq d_p$; V is a $p \times p$ orthogonal matrix ($V'V = I_p$).

For mean centered X, the series of embeddings $f_q(X)$ for $q = 1, \ldots,$

[5]It is not a proper sufficient statistic as no generative model has been assumed.

resulting from Eq.(5.7) is given by $f_q(X) = U_q D_q$, where U_q D_q are the q leading columns of U and D respectively. UD is thus the sequence of all solutions.

Now denoting $\mathfrak{S} = XX'$ and calling SVD: $\mathfrak{S} = UD^2U'$. We thus see that by decomposing $\mathfrak{S}$ we can recover U, D, and thus $f_q(X)$.

If X is not mean centred, the relation still holds, but we skip the presentation.

PCA with the Euclidean Distance Dissimilarity Graph Can we convert Euclidean distances to empirical covariances? Yes!

Denote the matrix of distances of a non-centred X: $\mathfrak{D}^2 = (\|x_i - x_j\|^2)$.

$$\begin{aligned}\mathfrak{D}^2_{i,j} =& \|x_i - x_j\|^2 && (5.8)\\ =& \|x_i - \bar{x}\|_2 + \|x_j - \bar{x}\|_2 - 2\langle x_i - \bar{x}, x_j - \bar{x}\rangle && (5.9)\\ =& \|x_i - \bar{x}\|_2 + \|x_j - \bar{x}\|_2 - 2\mathfrak{S}_{i,j} && (5.10)\end{aligned}$$

where $\mathfrak{S}_{i,j}$ is the empirical covariance between individual i and j. We thus have

$$\mathfrak{S} = -(I - M)\frac{\mathfrak{D}^2}{2}(I - M) \tag{5.11}$$

where M is the centring matrix: $M := \frac{1}{n}\mathbf{1}\mathbf{1}'$, and $\mathbf{1}$ an n vector of 1's.

5.4.2 Random Projections

What if instead of optimizing a linear embedding of the features with respect to some criterion, such as PCA (§5.4.1), or FA (§5.6.1), we simply apply a random linear mapping WX. How bad will the distances between the observations be distorted? It turns out that not too much! The Johnson-Lindenstrauss Lemma [Johnson and Lindenstrauss, 1984] quantifies this distortion, essentially implying that we may reduce the dimension of our data in a very naïve manner, while still conserving pair-wise similarities $\mathfrak{S}$ between observations.

5.4.3 Sparse Principal Component Analysis (sPCA)

When analyzing the PCA results, we often with to understand which features contribute to which component. This is much easier when the loadings, A are sparse, i.e., include many zeroes. This is the purpose of sPCA. We will not go into the technical detail, but merely state that sPCA performs this, à-la LASSO style, by means of l_1 regularization.

5.4.4 Multidimensional Scaling (MDS)

MDS aims at representing a network[6] of distances (or similarities) between observations, by embedding the observations in a q dimensional *linear* subspace, while preserving the original distances. The network may be obtained by computing some similarity (or dissimilarity) measure with the raw features X, or simply because the data itself is a network (social, communication, etc.).

Since MDS mainly serves for visualizing data, it is most natural to use $q = 2$. The embedding will distort the original distances (or...), and may even change the ordering of the observations. The good news is that it is easier to visualize and\or cluster them in their new simplified representation.

If the input of MDS is the empirical covariance similarity network, then MDS with "classical scaling" (see below) returns the exact same solution as PCA.

The embedding is merely the assigning of each point to a location in some lower dimensional linear space $\mathcal{M}$. The assignment is driven by a *stress function* which penalizes for the average distortion created by the embedding. The different types of MDSs, such as *Classical MDS*, and *Sammon Mappings*, differ in the stress function driving the embedding.

For more on MDS see Section 14.8 in Hastie et al. [2003] or Borg and Groenen [2005].

Mathematics of MDS We start with either a dissimilarity network $\mathfrak{D} = (d_{i,j})$, or a similarity network $\mathfrak{S} = (s_{i,j})$. Similarities can be thought of as correlations, and dissimilarities as distances (which are indeed the typical measures in use). Define $z_i \in \mathbb{R}^q$ the location of point i in the target linear space of rank q. The z_i's are set to minimize some penalty for geometric deformation called the *stress function*. Typical stress functions include:

Classical MDS Using the centred inner product (i.e. empirical covariance) as the similarity measure and minimizes the squared distortion: $s_{i,j} := \langle x_i - \bar{x}, x_j - \bar{x} \rangle$ and the new location are given by

$$\underset{z_1,\ldots,z_n}{argmin} \left\{ \sum_{i,j=1}^{n} (s_{i,j} - \langle z_i - \bar{z}, z_j - \bar{z} \rangle)^2 \right\}. \tag{5.12}$$

[6]The term Graph is typically used in this context instead of Network. But a graph allows only yes/no relations, while a network, which is a weighted graph, allows a continous measure of similarity (or dissimilarity). It is thus more appropriate.

Least Squares Also known as *Kruskal-Shepard*. Also minimizes the squared distortion.

$$\underset{z_1,\dots,z_n}{argmin} \left\{ \sum_{i \neq j} (d_{i,j} - \|z_i - z_j\|)^2 \right\}. \tag{5.13}$$

Sammon Mapping Also known as *Sammon's stress*, aims at minimizing the *proportion* of distortion:

$$\underset{z_1,\dots,z_n}{argmin} \left\{ \sum_{i \neq j} \frac{(d_{i,j} - \|z_i - z_j\|)^2}{d_{i,j}} \right\}. \tag{5.14}$$

Remark 5.4.3 (Classical and Least Squares MDS)**.** Although they both minimize the squared distortion, working with $\mathfrak{S}$ or $\mathfrak{D}$ lead to different solutions. In particular, Classical is a linear embedding while Stress is not.

5.4.5 Local MDS

Local MDS is motivated by the observation that if the data does not lay in a globally convex subspace, then global distances are a very distorted measure, whereas geodesic distances should be used instead. Their solution is to compute $\mathfrak{D}$ using only local distances, and then calling upon MDS.

Remark 5.4.4 (The Non-Linearity of Local MDS)**.** Local MDS is typically considered a non-linear-space embedding, thus belonging to Section 5.5. I currently do not think is the case, as it is presented as a linear space embedding. Maybe Remark 5.5.1 can explain the confusion in terminology.

5.4.6 Isometric Feature Mapping (Isomap)

Isomap, also known as *principal coordinate analysis*, is another method intimately related to MDS(§5.4.4).

Principal Coordinate Analysis

Isomap follows the same motivation as Local MDS (§5.4.5), but with a different algorithm to compute the dissimilarity matrix $\mathfrak{D}$.

Remark 5.4.5 (The Non-Linearity of Isomap)**.** Just like Local MDS, Isomap is typically considered a non-linear-space embedding (see Remark 5.4.4). I currently do not think is the case.

5.5 Non-Linear-Space Embeddings

Section 5.4 deals with representing the data in a *linear* sub space $\mathcal{M}$. They all aim at finding a basis which efficiently represents that data, with respect to some target function. In this section, we allow $\mathcal{M}$ to be non-linear. We will thus no longer be able to represent the data by its coordinates in some basis.

Remark 5.5.1 (Non Linear Dimensionality Reduction)**.** Do not let the title of this section be confused with the term Non-Linear Dimensionality-Reduction (NLDR). NLDR deals with the nature of the *embedding* operation, and not with the structure of the target manifold $\mathcal{M}$ (see also Remark 5.4.1). This section deals with embeddings into a non-linear subspace, regardless of the nature of the embedding operator. Cases of non-linear embeddings (NLDR) into a *linear* manifolds $\mathcal{M}$ belong in Section 5.4.

NLDR

5.5.1 Kernel Principal Component Analysis (kPCA)

Back to the motivating example from the PCA section (§5.4.1): assume we want to construct a "bigness" score, that best separates between individuals, but we no longer constrain it to be a linear function of the height and weight. Recalling that the best discrimination between observations means maximizing the variance of the *scores* given to individuals, we could try to find the best separating score $g(x)$ by solving

$$\underset{g}{argmax} \left\{ \mathbb{C}ov \left[g(X) \right] \right\} \tag{5.15}$$

where $g(x)$ maps an individual's features to a score in $\mathcal{M}$.

Alas, just like in the supervised learning problem, without any constraints on $\mathcal{M}$, thus on g, we might overfit and\or not be able to compute g as optimization is done in a infinite dimensional space. We thus have two matters to attend: (i) We need to constrain $g(x)$ so that it does not overfit. (ii) We need the problem to be computable. This is precisely the goal of kPCA.

We have already encountered a similar problem with Smoothing Splines (§3.1.12). It is thus not surprising that the solution has the same form. Namely, if we choose the right g's, the solution of Eq.(5.15) takes a very simple form. The classes of such g's are known as Reproducing Kernel Hilbert Spaces (RKHS). They are discussed in Appendix B.

Mathematics of kPCA [TODO]

5.5.2 Self Organizing Maps (SOM)

SOMs, are a non-linear-subspace dimensionality reduction method, aimed at good clustering. It is non-linear because the algorithm (which cannot be cast as an ERM problem) returns an embedding into a non-linear manifold. More details in Section 5.8.4.

5.5.3 Principal Curves and Surfaces

Task	Type	Input	Output	Concept	Remark
dim. reduce	algo.	X	parametric curve or surface	self consistency	

Principal curves (or surfaces), is an algorithm. It cannot be cast as an optimization problem. Being a non-linear space embedding method, the algorithm iterates until it returns a curve of a surface. In the curve case, it will return a curve with the *self consistency* property. I.e., a curve with a path that is the average of all it's closest data points. Roughly speaking, one can think of this curve as a parametrized function, connecting all the k-means cluster centres in the smoothest way possible. Using the same, slightly inaccurate depiction, a principal surface is a *surface* connecting these k-means cluster centres. We stress that in both cases, the output is a continuous parametrization of the curve or the surface.

Self Consisntecy

It is highly uncommon to approximate the data with surfaces (manifolds) with a dimension larger than 2, as typically the method is used for projecting the data before visualizing and\or clustering.

5.5.4 Local Linear Embedding (LLE)

Task	Type	Input	Output	Concept	Remark
dim. reduce	algo.	$\mathfrak{S}$	data embedding	local distance	

LLE aims at finding linear subspaces that are good approximations of small neighbourhoods of the whole data X. It is similar in spirit to Isomap (§5.4.6) and LocalMDS (§5.4.5). It differs, however, in the way similarities are computed, and in the way embeddings are performed. In particular, as the name may suggest, LLE performs local embeddings to linear subspaces.

The resulting approximating manifold $\mathcal{M}$, being the "stitching" of many linear spaces, is ultimately non linear.

5.5.5 Auto Encoders

[TODO]

5.5.6 Information Bottleneck

[TODO]

Remark 5.5.2 (Information Bottleneck and ICA)**.** [TODO]

5.6 Latent Space Generative Models

We have already met generative models for supervised learning (§3.4), and for unsupervised learning (§5.2). We now discuss a class of generative models, that can be seen as a dimensionality reduction device. The following models and methods assume that the data generating process is governed by some low dimensional unobservable *state*, also known as *latent variable*, or *hidden variable*. This is why these models are also known as *state space* models. A term coined by Kalman [1960].

State-Space Models

The simplification of the generative model from its original high-dimension allows us to: (i) *Interpret* the data generating process via its states. (ii) *Formulate* the density estimation problem as a low dimensional problem, we can actually hope to solve, even when the data itself is very high dimensional. (iii) *Visualize* the data.

The fundamental idea of latent space generative models is that while the data generating distribution may have a complicated form, when we condition on the unobserved variable, the distribution greatly simplifies.

5.6.1 Factor Analysis (FA)

Task	Type	Input	Output	Concept	Remark
dim reduce	optimization	X	embedding function		

Examples 5.6.1 and 5.6.2 motivate factor analysis, and many other latent space generative models in this section.

Example 5.6.1 (Intelligence Measure (g-factor))**.** Assume respondents answer p questions: $x_i \in \mathbb{R}^p$. Also assume, their responses are some linear function $A \in \mathbb{R}^p$ of a single attribute, s_i. We can think of s_i as the subject's "intelligence". We thus have

$$x_i = s_i A + \varepsilon_i \tag{5.16}$$

Example 5.6.2 (Face Rotations)**.** [TODO]

Factor Analysis is solved very similarly to PCA (§5.4.1), so that the two are often confused. FA, however, stems from a rather different motivation than PCA. PCA is motivated by finding variable combinations (scores) with most variance. FA is a generative method, aimed at finding uncorrelated latent attributes.

In FA we assume that the observed X's depend linearly on a set of q independent latent (i.e. unobservable) attributes we denote with $\mathbf{s}$. The generative model is thus

$$X = A\mathbf{s} + \varepsilon \tag{5.17}$$

Assuming a generative distribution on $\mathbf{s}$ and ε, we may try to estimate $A\mathbf{s}$ by maximum likelihood. Recovering the particular latent attributes $\mathbf{s}$ from $\widehat{A\mathbf{s}}$ is still impossible as there are infinitely many such solutions. To see this, consider an orthogonal *rotation* matrix R ($R'R = I$). For each such R: $A\mathbf{s} = AR'R\mathbf{s} = A^*\mathbf{s}^*$.

The arbitrary choice of R changes the interpretation of the latent attributes. This is why many researchers find FA an unsatisfactory inference tool.

Remark 5.6.1 (Identifiability in PCA)**.** The non-uniqueness (non-identifiability) of the FA solution under variable rotation is never mentioned in the PCA context. Why is this? This is because the methods solve different problems. Indeed, the PCA problem has no such ambiguity. Generative latent variables, as in FA, are only defined up to rotations. They are thus not unique. Combinations with maximal variance, or series of linear-hyperspaces, as in PCA, are unique. There is no room for rotations.

FA Terminology

- **Factors**: The unobserved attributes $\mathbf{s}$. Not to be confused with the *principal components* in the context of PCA.
- **Loadings**: The A matrix; the contribution of each attribute to the observed X.

- **Rotation**: An arbitrary orthogonal re-combination of the latent attributes $\mathbf{s}$ and loadings, which changes the interpretation of the result.

For a brief review of Factor Analysis see Hastie et al. [2003]. For an full exposition, and a discussion of the differences with PCA, see Jolliffe [2002].

Rotations

- **Varimax**: By far the most popular rotation. Attempts to construct factors that are similar to the original variables, thus facilitating interpretation. This can be seen as a "soft" approach to sPCA (§5.4.3).
- **Quartimax rotation**: Seeks a minimal number of factors to explain each variable. May thus result factors that are uninterpretable, since they all rely on the same variables.
- **Equimax rotation**: A compromise between Varimax and Quartimax.
- **Direct oblimin rotation**: Relaxes the requirement of the factors to be uncorrelated, so that they may be similar to the original variables; even more so than in varimax. This facilitates the interpretability of the factors.
- **Promax rotation**: A computationally efficient approximation of oblimin.

Remark 5.6.2 (Non Linear FA)**.** The FA method presented herein deals with features, X, that are linear in the latent factors, $\mathbf{s}$. It is obviously possible to generalize the framework to deal with non-linear functions of the latent factors: $X = g(S)$. This practice is, however, quite uncommon.

Mathematics of FA [TODO]

5.6.2 Independent Component Analysis (ICA)

Task	Type	Input	Output	Concept	Remark
dim reduce	meta problem	X	embedding function		

ICA is a family of latent space models– a meta-method. It assumes data is generated as some function of the latent variables $\mathbf{s}$. In many cases this function is assumed to be linear in $\mathbf{s}$ so that ICA is compared, if not confused, with PCA and even more so with FA. In its most popular form, X is assume to be a *linear* function of the latent independent components: $X = A\mathbf{s}$.

The fundamental idea of ICA is that $\mathbf{s}$ has a joint distribution of *independent* variables. This is a stronger assumption that the typical FA assumption where the distribution of $\mathbf{s}$ is merely assumed to be uncorrelated. This independence assumption solves the the non-uniquness of $\mathbf{s}$ in FA (§5.6.1).

Two assumptions distinguish ICA from FA, allowing the identification of $\mathbf{s}$: (i) The latent variables are **not** Gaussian distributed. (ii) The latent variables are statistically *independent*.

Being a generative model, estimation of $\mathbf{s}$ can then be done using maximum likelihood, or other estimation principles. A popular information theoretic estimation principle is *infomax*.

Infomax

ICA is a popular technique in signal processing, where $\mathbf{s}$ is actually the signal (e.g. sound) produced by several different sources. Recovering $\mathbf{s}$ is thus recovering the original signals mixing in the recorded X. This is known as *blind source separation*.

Blind Source Seperation

Remark 5.6.3 (ICA and FA)**.** The solutions to the (linear) ICA problem can ultimately be seen as a solution to the FA problem with a particular rotation R implied by the independence assumption. Put differently, the formulation of the (linear) ICA problem, implies a unique rotation.

For a general discussion of ICA see Jolliffe [2002]. For a brief exposition of the linear ICA see Hastie et al. [2003]. For a detailed review of ICA see Hyvärinen and Oja [2000].

Mathematics of ICA For ease of presentation we present a simple setup, which can be considerably generalized. In this setup, we will first analyze the population problem, i.e., in terms of random variables. We thus replace the data X, with the random vector $\mathbf{x}$, and afterwards consider implementation for finite samples.

- $\mathbf{x} = A\mathbf{s}$, implying that $\mathbf{x}$ is *linear* in the latent components, and the latent space is of dimension $q = p$. It follows that $s = A'X$.
- $\mathbf{x}$ has been pre-whitented, so that $\mathbf{C}ov[\mathbf{x}] = I$.
- Distance between distributions are measured using the Kullback-Leibler divergence (KL): $D_{KL}(\mathbf{x}\|\mathbf{s})$.

The optimization problem in this simple ICA is to find an orthogonal matrix A, for which: (i) the components of $A'\mathbf{x}$ are independent; (ii) $A'\mathbf{x}$ is a good approximation of $\mathbf{x}$. Formally:

$$\underset{A \text{ orthogonal }; A'\mathbf{x} \text{ independent}}{argmin} \{D_{KL}(A'\mathbf{x}\|\mathbf{x})\}. \tag{5.18}$$

By enforcing the independence constraint in Eq.(5.18), and due to the properties of the KL divergence, Eq.(5.18) is equivalent to

$$\underset{A \text{ orthogonal}}{argmin} \left\{ \sum_{j=1}^{q} H(A_j \mathbf{x}) - H(\mathbf{x}) \right\} \tag{5.19}$$

where $H(\mathbf{x})$ denotes the Entropy of the random variable $\mathbf{x}$ (Definition G.1). Now, $H(\mathbf{x})$ is obviously fixed, so we need to minimize $H(A_j \mathbf{x})$. A classical result in information theory, is that the Gaussian distribution has the maximal entropy. Minimizing $H(A_j \mathbf{x})$ can thus be interpreted as finding a matrix A such that its columns return random variables, $A_j \mathbf{x}$, that are as *non-Gaussian* as possible.

This is where the population analysis ends. The insight we take from it, is that finding independent components, is actually finding non-Gaussian combinations of $\mathbf{x}$. The different implementations of ICA, indeed look for a matrix A which returns the most non-Gaussian combinations of the observed X.

5.6.3 Exploratory Projection Pursuit

Task	Type	Input	Output	Concept	Remark
dim reduce	algorithm	X	embedding function		

Example 5.6.3 (Intelligence Factor Continued)**.** Returning to the intelligence score example (5.6.1). We seek a single linear combination of answers, a projection $u \in \mathbb{R}^p$, which helps us visualize, understand, and possibly cluster individuals. By assumption, $x_i u = s_i A u + \varepsilon_i u$. For any u that is orthogonal to A then Au will vanish and the index $x_i u$ is dominated by $\varepsilon_i u$, also close to zero. Moreover, $\varepsilon_i u$ will probably look Gaussian due to the CLT. For projections u that are similar to A, then Au will be large and the different $x_i u$'s will spread nicely on a line, since Au amplifies s_i.

The fundamental idea of Exploratory Projection Pursuit, first presented in Friedman and Tukey [1974], is that if the data were pure noise in some high dimension, then almost any projection will look like Gaussian noise. If, however, the data were not pure noise, then projections in the direction of the signal, would show structure.

Remark 5.6.4 (Projection Pursuit and ICA)**.** From the Example 5.6.3 we note that $x_i u$ will look Gaussian for all rotations, u, that do not capture

signal. This observation motivated Friedman and Tukey [1974] to find u's so that they depart from Gaussianity. It turns out that the ICA problem (§5.6.2), while having a different motivation, also culminates in finding rotations that maximally depart from Gaussianity. For this, both ICA and Exploratory Projection Pursuit, can be seen as similar solutions to the problem of finding informative rotations in factor analysis (§5.6.1).

5.6.4 Compressed Sensing

[TODO]

5.6.5 Generative Topographic Map (GTM)

[TODO]

5.6.6 Finite Mixtures

Task	Type	Input	Output	Concept	Remark
many	generative model	X	estimates		

A very simply, perhaps the simplest, class of latent variable generative models, is that of Finite Mixtures. In a finite mixture, the fundamental assumption is that the observed data is sampled from K unobserved classes. Each class, having some simple joint distribution. Being unobserved, we cannot directly learn the conditional simple distributions, without somehow learning the class assignments. The likelihood function of such data, is thus the *marginal* probability over all possible classes, thus a *mixture*. The good news is that a very complex generative model, may be much more easy to learn and interpret if we can assume, or approximately assume, it is actually a finite mixture.

The probability density of a mixture of K classes, each with density $p_k(x)$, is given by

$$p(x) = \sum_{k=1}^{K} \pi_k p_k(x) \tag{5.20}$$

where π_k is the probability of class k.

Remark 5.6.5 (Finite Mixture Distributions)**.** While very simple to understand, mixture distributions are an unpleasant probabilistic creature. Because the probability mass (or density) function is additive in the underlying

mixing components, is has a very challenging functional form. For instance-the likelihood problem will be non-convex with multiple local extrema, so that numerical optimization schemes are not guaranteed to converge. It may also be the case that the likelihood is unbounded, so that the local extrema are not a global extrema, complicating our attempts at maximum likelihood estimation. Hypothesis testing for the number of mixing components is also very challenging as classical statistical theory assumptions do not apply.

Remark 5.6.6 (Mixtures And the Expectation Maximization Algorithm (EM))**.** As previously stated, the likelihood function of mixtures is typically non convex. It's optimization is typically done with a numerical algorithm called the Expectation Maximization (EM) algorithm. Roughly speaking, the idea behind the EM, is that in each iteration observations are assigned to classes, the simple conditional distributions learned, then reasigned using the learnt parameters, etc. until convergence. Even though convergence to an optimum is not guaranteed, it is by far the most popular algorithm for learning mixtures.

Remark 5.6.7 (Mixtures For Clustering)**.** Because the output of learning a mixture, is a density like Eq.(5.20), a learnt mixture can easily serve for clustering by a simple application of Bayes rule. Simply assign an observation to the cluster with highest (posterior) probability:

$$P\left(x \in k|x\right) = \frac{\pi_k p_k(x)}{p(x)} \tag{5.21}$$

Remark 5.6.8 (Mixture in Supervise Learning)**.** The generative models underlying several supervised learning methods, are actually finite mixtures. Fisher's LDA (§3.4.1) implies a finite mixture of Gaussians, with the same covariance and different means. Fisher's QDA (§3.4.2) implies a finite mixture of Gaussian, with different covariance and different means. Naïve Bayes (§3.4.3) implies a mixture of some distribution, with independent components.

5.6.7 Hidden Markov Models (HMM)

Hidden Markov Models may have two possible interpretations.

The first, and perhaps most common, is as a generalization of the mixture model in the case the generative model assumes dependencies between x_i's and we wish to use a mixture model. In this case, we need to model and learn the transition probabilities between states (or classes, are we previously referred to the hidden states of a mixture). Assuming the transition between

states follows a *Markov process*, i.e., the probabilities of transition depends only upon the current state, the resulting model is an *hidden Markov model*, or *hidden Markov chain*.

Markov Process

The second, is as a generalization of the Graphical Models (§5.2.3). This generalization is known a *latent space graphical model* (§5.6.8). In this case, the Markov property does not describe the process of transition between states, but rather the Markov field property of the conditional distribution in each state, $p_k(x)$.

5.6.8 Latent Space Graphical Models

The idea of simplifying a complicated distribution by conditioning on some latent variable can be compounded with the idea of a graphical model (§5.2.3). Put differently, the generative model can be a simple graphical model when conditioned on some latent variable. See Section 17.4.2 in Hastie et al. [2003] for more details.

5.6.9 Matrix Factorization

[TODO- is it generative?]

5.7 Random Graph Models

There are several occasions in which our date does not consist of the classical features X, but rather directly given as a similarity graph $\mathfrak{S}$. This can happen when the data itself is a graph, such as social networks, communication networks, protein interactions networks, etc. It may also be the case that the original data is indeed given as features, then used to compute a similarity graph.

In those occasion where the data is actually $\mathfrak{S}$, we may still want to cluster observations, visualize, etc.

Random Graphs are a class of *generative models*, but unlike other generative methods, the model does not specify a distribution over the features, but rather, a distribution over proximity measures. For this, it is intimately related to *random graph* and *random matrix* theory.

For a review of random graph models see Goldenberg et al. [2010].

5.7.1 Erdős Rènyi

[TODO]

5.7.2 Exchangeable Graph Model

[TODO]

5.7.3 $p1$ Graph Model

[TODO]

5.7.4 $p2$ Graph Model

[TODO]

5.7.5 Stochastic Block Graph Model

[TODO]

5.7.6 Latent Space Graph Model

[TODO]

5.7.7 Exponential Random Graphs (ERGMs)

[TODO]

Remark 5.7.1 (Relation to Spectral Clustering)**.** [TODO]

5.8 Cluster Analysis

In cluster analysis, or *data segmentation*, we aim at assigning observations to (hopefully) homogenous and meaningful clusters. We may also consider the more ambitious goal of finding not only clusters, but a *clustering function*: a mapping between $\mathcal{X}$ to clusters (and not only between X to clusters).

Cluster analysis is typically easier than learning a joint distribution (§5.2) or even detecting high density regions (§5.3).

We will see that for cluster analysis, we don't always need the actual features X; it will turn out that many methods only require the pair-wise similarities ($\mathfrak{S}$). For this reason, clustering is intimately related to graph, or *network*[7] partitioning problems.

[7] A graph describes a yes/no relation. A network is a weighted graph, which not only describes the existence of a relation, but also its strength.

5.8.1 K-Means Clustering

Task	Type	Input	Output	Concept	Remark
Clustering	Algorithm	X, K	Data Clustering	–	–

The goal behind K-means clustering algorithm is finding a representative point for each of K clusters, and assign each data point to one of these clusters. As each cluster has a representative point, this is also a *prototype method.*. The clusters are defined so that they minimize the average distance between all points to the center of the cluster.

Prototype Methods

K-means clustering requires the raw features X as inputs, and not only a similarity graph. This is evident when examining Algorithm 10.

In K-means, the clusters are first defined, and then similarities computed. This is thus a *top-down* method.

Top Down Clustering

Algorithm 10 K-Means

Choose the number of clusters K.
Arbitrarily assign points to clusters.
while Clusters keep changing **do**
  Compute the cluster centers as the average of their points.
  Assign each point to its closest cluster center (in Euclidean distance).
end while
return Cluster assignments and means.

Remark 5.8.1 (The population equivalent of K-means)**.** You may wonder-what population quantity is K-means actually estimating? The estimand of K-means is known as the *K principal points*. Principal points are points which are *self consistent*, i.e., they are the mean of their neighbourhood.

Principal Points

5.8.2 K-Medoids Clustering

Task	Type	Input	Output	Concept	Remark
Clustering	Algorithm	$\mathfrak{S}, K$	Data Clustering	–	–

If a Euclidean distance is inappropriate for a particular $\mathcal{X}$, or that robustness to corrupt observations is required, or that we wish to constrain the cluster centers to be actual observations, then the *K-Medoids* algorithm is an adaptation of K-means that allows this.

Algorithm 11 K-Medoids

Given a similarity graph $\mathfrak{S}$.
Choose the number of clusters K.
Arbitrarily assign points to clusters.
while Clusters keep changing **do**
Within each cluster, set the center as the data point that minimizes the sum of distances to other points in the cluster.
Assign each point to its closest cluster center (in $s(x_i, x_j)$ distance).
end while
return Cluster assignments and centers.

See Section 14.3.10 in Hastie et al. [2003].

5.8.3 Hierarchical Clustering

Task	Type	Input	Output	Concept	Remark
Clustering	Algorithm	$\mathfrak{S}$	Data Clustering	–	–

These algorithms take similarity (dissimilarity) graphs as inputs. Hierarchical clustering is a class of greedy graph-partitioning algorithms. Being hierarchical by design, they have the attractive property that the evolution of the clustering can be presented with a dendogram (§3.1.11). Unlike k-means, the method is the algorithm itself and cannot be cast as an optimization (ERM) problem. Also, it does not require an a-priori choice of the number of cluster.

For more on hierarchical clustering see Section 14.3.12 in Hastie et al. [2003].

Two main sub-classes of algorithms can be identified: *agglomerative*, and *divisive*.

Agglomerative Clustering

Agglomerative clustering algorithms are bottom-up algorithm which build clusters by joining smaller clusters. To decide which clusters are joined at each iteration some measure of closeness between clusters is required.

Single Linkage Cluster distance is defined by the distance between the two **closest** members.

Complete Linkage Cluster distance is defined by the distance between the two **farthest** members.

Group Average Cluster distance is defined by the **average** distance between members.

Divisive Clustering

Divisive clustering algorithms are top-down algorithm which build clusters by splitting larger clusters. There are several way to divide clusters. All amount to considering some homogeneity measure on the set of all possible divisions. As this is typically computationally intense, greedy algorithms can be employed.

5.8.4 Self Organizing Maps (SOM)

Task	Type	Input	Output	Concept	Remark
Clustering	Algorithm	X	Data Clustering, Visualization	Dim. Reduce	–

SOMs, also known as *Kohonen maps*, and *self organizing feature maps* (SOFM), and *constrained topological map*. They are dimensionality reduction methods driven by the quality of clustering. Unlike many similarity based dimensionality reduction methods, it requires the original features X, and not only a similarity graph $\mathfrak{S}$.

Kohonen Map

SOMs is an algorithm. It cannot be cast as an optimization problem. It can be seen as the marriage of K-means clustering (§5.8.1) with a non-linear space embedding (§5.5).

For more on SOMs see Section 14.4 in Hastie et al. [2003].

5.8.5 Spectral Clustering

Task	Type	Input	Output	Concept	Remark
Clustering	Meta-Algorithm	$\mathfrak{S}$	Data Clustering	Spectral Trick	–

Spectral clustering is tailored for situations where naïve similarity measures $\mathfrak{S}$, such as the empirical covariances used in PCA, fail to capture the true notion of distances in the data, as it lay in some non convex set.

For some intuition, let X be n images of facial expressions of the same individual. It would be quite miraculous if the Euclidean distance were to capture the true notion of distance between facial expressions.

Spectral clustering implies following (meta-)algorithm:

Algorithm 12 Spectral Clustering

Compute a similarity graph, $\mathfrak{S}$ with some *local* similarity measure.
Convert the similarity graph $\mathfrak{S}$ to a dissimilarity graph $\mathfrak{D}$ called the *graph-laplacian*.
Use the spectral trick (§C) to reduce the dimension of the data.
Cluster data (say, with K-means, §5.8.1) in the low-dimension representation.
return Low dimensional data representation and cluster assignments.

Graph Laplacian

The idea of using *local* measures of similarity re-appears in several dimensionality reduction techniques such as LocalMDS (§5.4.5), Isomap (§5.4.6) and LLE (§5.5.4). Perhaps the most popular method is that of *mutual K-nearest-neighbor graph*.

Mutual K-Nearest-Neighbor Graph

The spectral trick (Appendix C) is not a new one, and is motivated by the solution to the PCA problem (§5.4.1). In PCA, the solution is given by using the empirical covariances as similarities.

The idea of clustering in a low dimensional representation of the data is a powerful one. Spectral clustering is a bundle of an embedding and clustering. These two, however, can be decoupled: just perform your favourite dimensionality reduction then followed by your favourite clustering technique. SOMs (§5.8.4), and exploratory projection pursuit (§5.6.3) should be noted as techniques where the dimensionality reduction and the clustering *cannot* be decoupled. In both, the dimensionality reduction is driven explicitly by the clustering performance.

Chapter 6

Recommender Systems and Collaborative Filtering

6.1 Recommender Systems

A recommender system is a software that, as the name suggests, gives recommendations to the user. Notable examples include Book recommendations by Amazon, and film recommendations by Netflix. The two main approaches to recommender systems include *content filtering* and *collaborative filtering*.

6.2 Content Filtering

In content filtering, the system is assume to have some backround information on the user (say, because he logged in), and uses this information to give him recommendations. The recommendation in this case, is approached as a supervised learning problem: the system learns to predict a product's rating (y) based on the user's features (x). It then computes the rating for many candidate products and recommends a set with high predicted ratings.

6.3 Collaborative Filtering

Unlike content filtering, in *collaborative filtering*, there is no external information on the user or the products, besides the ratings of other users. The term collaborative filtering, was coined by the authors of the first such system– Tapestry [Goldberg et al., 1992].

Collaborative filtering can be approached as a supervised learning problem, or as an unsupervised learning problem. This is because it is neither.

It is essentially a *missing data* problem. To see this consider a matrix of rankings, $\mathcal{R}$ where the i, j'th entry, $r_{i,j}$, is the ranking of user i movie j. Predicting $r_{i,j'}$, i.e., the ranking of a currently unseen movie, is essentially an imputation of a missing value.

Missing Data

The two main approaches to collaborative filtering include *neighbourhood methods*, and *latent factor models* Koren et al. [2009].

Neighbourhood Methods

The neighbourhood methods to collaborative filtering rest on the assumption that similar individuals have similar tastes. If someone similar to individual i has seen movie j', then i should have a similar opinion.

The notion of using the neighbourhood of a data point is not a new one. We have seen it being used for supervised learning in kernel regression (§3.2.2) and KNN (§3.2.1).

Neighbourhood methods for collaborative filtering, or missing data imputation in general, can thus be seen as supervised learning problems, and solved in the same way.

Remark 6.3.1 (Collaborative Filtering and Other Supervised Learning Methods)**.** If you are wondering, why only neighbourhood methods for supervised learning apply to collaborative filtering, you are right. Any supervised learning method can be applied to impute entries in $\mathcal{R}$. Neighbourhood methods are merely the most popular.

Latent Factor Models

The latent factor approach to collaborative filtering rests on the assumption that the rankings are a function of some latent user attributes and latent movie attributes. This idea is not a new one, as we have seen it in the context of unsupervised learning in factor analysis (FA) (§5.6.1), independent component analysis (ICA) (§5.6.2), and other latent space generative models. From the relation between FA and PCA, it should also come as no surprise that collaborative filtering can also be viewed as a matrix approximation problem. When approximating the rankings matrix ($\mathcal{R}$) with some simple manifold $\mathcal{M}$, en-passant, we also impute the missing entries in $\mathcal{R}$.

We now see that collaborative filtering, and missing data imputation in general, can also be approached as an unsupervised learning problem.

Appendix A

The Relation Between Supervised and Unsupervised Learning

It may be surprising that collaborative filtering can be seen as both an unsupervised and a supervised learning problem. But these are not mutually exclusive problems. In fact, the relation has already been implied in the introduction to the unsupervised learning section (§5), and we now make it explicit.

In unsupervised learning we try to learn the joint distribution of x, i.e., try to learn the relationship between any variable in x to the rest, we may see it as several supervised learning problems. In each, a different variable in x plays the role of y.

Many unsupervised learning methods can be seen in this light. We, however, will not be exploring this avenue right now.

[TODO: autoencoders].

Appendix B

The Kernel Trick and Reproducing Kernel Hilbert Spaces (RKHS)

In the context of supervised learning the *kernel trick* is a mathematical device that allows to learn very complicated predictors (f) in a computationally efficient manner. More generally, in the context of unsupervised learning, the kernel tricks allow to learn complicated non-linear mappings of the original features (and not only predictor functions).

Not all predictors and not all problem admit this trick. Then again, many do. Methods for which it applies include: SVM's (§3.1.7), principal components analysis (§5.4.1), canonical correlation analysis (§3.3.5), ridge regression (§3.1.3), spectral clustering (§5.8.5), Gaussian processes[1], and more[2].

We now give an exposition of the method in the context of supervised learning.

Think of smoothing splines (§3.1.12); It was quite magical that without constraining the hypothesis class $\mathcal{F}$, the ERM problem in Eq.(3.16) has a finite dimensional closed form solution. The property of an infinite dimensional problem having a solution in a finite dimensional space is known as the *kernel property*. We with to generalize this observation and ask- which problems have the kernel property? Stating the general optimization problem:

Kernel Property

$$\underset{f}{argmin} \left\{ \frac{1}{n} \sum_i l(y_i, f(x_i)) + \lambda J(f) \right\} \tag{B.1}$$

[1] See the Bayesian interpretation below to see why they apply to Gaussian Processes.

[2] This partial list is taken from Wikipedia: `http://en.wikipedia.org/wiki/Kernel_method`

The problem is then- what type of penalties $J(f)$ will return simple solutions to Eq.(B.1). The answer is: function that belong to *Reproducing Kernel Hilbert Space* function spaces. RKHS's are denoted by $\mathcal{H}_\mathcal{K}$. They include many functions, but they are a rather "small" subset of the space of all possible functions. These spaces, and the functions therein, are defined by another function called a *Kernel* denoted by $\mathcal{K}$. Choosing a particular kernel defines the space and the functions therein. Choosing a particular kernel, also defines the form of J in Eq.(B.1). Put differently: for any choice of a kernel $\mathcal{K}$, there is a particular $J(f)$ for which the solution of Eq.(B.1) will be a function in $\mathcal{H}_\mathcal{K}$ and will be easily computable.

B.1 Mathematics of RKHS

We now show how choosing a kernel $\mathcal{K}$ defines a space $\mathcal{H}_\mathcal{K}$, and a penalty $J(f)$.

A kernel is a non-negative symmetric function of two arguments: $\mathcal{K}(x, y) : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}_+$. By fixing y, $\mathcal{K}(x, y)$ is a function with a single argument $x \mapsto \mathcal{K}(x, y)$. $\mathcal{H}_\mathcal{K}$ is merely the space of functions of x, spanned at given y's:

$$f(x) : \sum_m \alpha_m \mathcal{K}(x, y_m) \tag{B.2}$$

From linear algebra, you may know that positive definite matrices be diagonalized. This analogy carries to $\mathcal{K}$, which admits an eigen-expansion:

$$\mathcal{K}(x, y) = \sum_{i=1}^{\infty} \gamma_i \phi(x) \phi(y) \tag{B.3}$$

Using Eqs.(B.3) and (B.2) we can thus expand elements f of $\mathcal{H}_\mathcal{K}$:

$$f(x) = \sum_{i=1}^{\infty} c_i \phi(x) \tag{B.4}$$

where $c_i = \gamma_i \sum_m \alpha_m \phi(y)$. We also define a norm $\|f\|^2_{\mathcal{H}_\mathcal{K}}$ in this space, which is induced by $\mathcal{K}$:

$$\|f\|^2_{\mathcal{H}_\mathcal{K}} := \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i} \tag{B.5}$$

The penalty $J(f)$ in Eq.(B.1), is simply be $\|f\|^2_{\mathcal{H}_\mathcal{K}}$. The f's that solve Eq.(B.1) are guaranteed to have a simple form. They reside in an n dimen-

sional linear function space [Wahba, 1990]:

$$f(x) = \sum_{i=1}^{n} \alpha_i \mathcal{K}(x, x_i) \tag{B.6}$$

The functions $\mathcal{K}(x, x_i)$ can be seen as a basis to the solution space. The good news continue! Being only n dimensional, the norms of these f's, do not require integration but rather only finite summation:

$$\|f\|^2_{\mathcal{H}_\mathcal{K}} = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathcal{K}(x_i, x_j)\alpha_i\alpha_j := \alpha' K \alpha. \tag{B.7}$$

Adding the above results, we can restate Eq.(B.1) and say that when fixing $\mathcal{K}$ and using the appropriate J, we only need to solve:

$$\underset{\alpha}{argmin} \left\{ \frac{1}{n} \sum_i l(y_i - K_i\alpha) + \lambda \alpha' K \alpha \right\} \tag{B.8}$$

which is a quadratic programming problem over an n dimensional linear space, easily solvable with numeric routines.

B.2 The Bayesian View of RKHS

Just as the ridge regression (§3.1.3) has a Bayesian interpretation, so does the kernel trick. Informally, the functions solving Eq.(B.1) can be seen as the posterior mode if our prior beliefs postulate that the function we are trying to recover is a Gaussian zero-mean process with covariance given by $\mathcal{K}$. This view suggests the intuition that the regularization introduced by $J(f)$ shrinks the estimated f towards a smoother function. At an extreme, where $\lambda \to \infty$, we will recover a constant function, since the the mode of our Gaussian process prior is at the origin of $\mathcal{H}_\mathcal{K}$.

B.3 Kernel Generalization of Other Methods

[TODO: Sec 18.5.2]

Appendix C

The Spectral Trick

[TODO]

Appendix D

Generative Models

By *generative model* we mean that we specify the whole data distribution. This is particularly relevant to supervised learning where many methods only assume the distribution of $P(y|x)$ without stating the distribution of $P(x)$. Assuming only $P(y|x)$ is known as a *discriminative model*, or *discriminative analysis*. In a generative model, in contrast, we assume the whole $P(y, x)$.

Descriminative Model

For the mere purpose of making a prediction, we do not need to learn $P(y, x)$. Knowing this distribution, however, does permit to make predictions, via Bayes Theorem: $P(y|x) = \frac{P(y,x)}{\int P(y,x)dy}$. Generative models make use of this relation to make predictions.

To gain some intuition, consider a supervised learning problem where the data has an equal number of samples per class. Learning the distribution of x withing each class, allows to a simple classification of a given x to the class with highest probability. LDA (§3.4.1), QDA (§3.4.1), and Naïve Bayes (§3.4.3) follow this exact same rational.

Appendix E

Dimensionality Reduction

Dimensionality reduction is a useful concept for both supervised and unsupervised learning. It allows to represent high dimensional data in a lower dimension. This allows the visualization of the data in a human-tractable dimension, the application of low-dimensional algorithms, and the reduction of computational burden when using the data for supervised learning.

The fundamental idea behind dimensionality reduction is that while $\mathcal{X}$ may be high dimensional, thus $P(x)$ hard to learn, there is hope that $\mathbf{x}$ does not really vary in the whole space. If the mass of $P(x)$ is concentrated around some low dimensional manifold $\mathcal{M}$, then the original problem might be approximated to learning the distribution of the projection $P(X \hookrightarrow \mathcal{M})$ on $\mathcal{M}$. If $\mathcal{M}$ is fairly low dimensional, we may hope to visualize and understand $P(X \hookrightarrow \mathcal{M})$ with fairly simple tools. Dimensionality reduction also reduces the memory required to represent the data. It is thus intimately related to *lossy compression* in information theory.

Lossy Compression

A similar reasoning justifies dimensionality reduction in supervised learning. While $P(x)$ might vary in the whole $\mathcal{X}$, but there might be only few directions which carry information on y. Learning $P(y|x)$ can thus be well approximated by $P(y|x \hookrightarrow \mathcal{M})$.

As was first observed in the context of PCA (§5.4.1), for many types of embeddings, i.e., for many target manifolds and reconstruction errors, we do not really need the original data X, but rather only a graph of similarities between data points ($\mathfrak{S}$). This allow the dimensionality reduction theory to borrow from *graph embedding* and *graph drawing* literature.

The different dimensionality methods can be stratified along these lines: We can further stratify dimensionality reduction methods along these lines:

Linear-Space vs. Non-Linear-Space Embeddings When reducing the dimension of X, it can be mapped (embedded) into a linear subspace, $\mathcal{M} \subset \mathcal{X}$, or a non-linear $\mathcal{M}$.

Linear vs. Non-Linear Space Embeddings Not to be confused with the previous item. The dimensionality reducing mapping, $X \hookrightarrow \mathcal{M}$, can be a linear operation on the data or a non-linear one.

Learning an Embedding vs. Learning an Embedding Function When learning a mapping to a lower dimensional space, we can map the original data points (an embedding), or learn a mapping of the whole data space (an embedding function).

E.1 Dimensionality Reduction in Supervised Learning

Dimensionality reduction is often performed before supervised learning to keep computational complexity low. It is sometimes performed on X while ignoring y (e.g. PCA Regression in §3.3.3), and sometimes as part of the supervised learning (e.g. PLS in §3.3.4).

From a statistical view-point, it is preferable to solve the supervised learning and dimensionality reduction simultaneously. This is because the subspace $\mathcal{M}$, which approximates $P(x)$ may differ than the one that approximates $P(y|x)$. From a computational view-point, however, it may be preferable to decouple the stages.

E.2 Graph Drawing

[TODO]

Appendix F

Latent Variables

[TODO]

Appendix G

Information Theory

Definition G.1 (Entropy)**.** [TODO]

Definition G.2 (Mutual Information)**.** [TODO]

Definition G.3 (Kullback–Leibler Divergence)**.** [TODO]

Appendix H

Notation

In this text we use the following notation conventions:

x A vector (or scalar). It is typically a column vector, but this should typically be implied from the text.

$\mathbf{1}$ A vector of 1's.

$\mathbf{x}$ A vector (or scalar) valued random variable.

X A matrix.

$\mathbf{X}$ A matrix valued random variable (a random matrix).

X' The matrix transpose of X.

$\|x\|_2$ The l_2 norm of x: $\sqrt{\sum_j x_j^2}$.

$\|x\|_1$ The l_1 norm of x: $\sum_j |x_j|$

$\|X\|_{Frob}$ The Frobenius matrix norm of X: $\|X\|_{Frob}^2 = \sum_{ij} x_{ij}^2$

$\mathbb{O}$ The space of orthogonal matrices.

$\langle x, y \rangle$ The scalar product of two vectors x and y.

$\mathcal{S}$ A data sample.

$\mathbf{E}[\mathbf{x}]$ The expectation of $\mathbf{x}$.

$\mathbb{E}[x]$ The empirical expectation (average) of the vector x.

$\mathbf{C}ov[\mathbf{x}]$ The covariance matrix of $\mathbf{x}$: $\mathbf{E}[(\mathbf{x} - \mathbf{E}[\mathbf{x}])(\mathbf{x} - \mathbf{E}[\mathbf{x}])']$.

$\mathbb{C}ov[x]$ The empirical covariance matrix of x: $\mathbb{E}[(x - \mathbb{E}[x])(x - \mathbb{E}[x])']$.

$\rho(\mathbf{x}, \mathbf{y})$ The correlation coefficient.

$F_{\mathbf{x}}(t)$ The CDF of $\mathbf{x}$ at t.

$F_{\mathbf{x}}^{-1}(\alpha)$ The inverse CDF at α (the quantile function).

$\mathbb{F}_x(t)$ The empirical CDF of data vector x.

$\mathbb{F}_x^{-1}(\alpha)$ The empirical α quantile of the data vector x.

$\mathbf{x} \sim P$ The random variable $\mathbf{x}$ is P distributed.

$p(x)$ The density function of P at x.

$\mathcal{N}(\mu, \sigma^2)$ The univariate Gaussian distribution with mean μ and variance σ^2.

$\mathcal{N}(\mu, \Sigma)$ The multivariate Gaussian distribution with mean vector μ and covariance matrix Σ.

$\mathcal{L}(\theta)$ The likelihood function at θ.

$L(\theta)$ The log likelihood function at θ.

$l(x, \theta)$ The loss function of θ at x.

$R(\theta)$ The risk at θ.

$\mathbb{R}(\theta)$ The empirical risk at θ.

$f(x)$ A prediction (hypothesis) at x.

$\mathcal{F}$ The class of all hypotheses f.

$\mathbb{L}$ A hyperplane.

$\mathcal{G}$ A set of categories.

$[t]_+$ The positive part of t: $\max\{0, t\}$.

$\mathcal{K}(x, y)$ A kernel function evaluated at (x, y).

$I_{\{A\}}$ The indicator function of the set A.

$\mathcal{M}$ A manifold.

$\hookrightarrow$ A projection operator.

s_{ij} A similarity measure between observations i and j.

d_{ij} A dissimilarity (i.e., distance) measure between observations i and j.

$\mathfrak{S}$ A weighted graph (i.e. network, or matrix) of similarities between observations.

$\mathfrak{D}$ A weighted graph (i.e. network, or matrix) of dissimilarities between observations.

$D_{KL}(\mathbf{x}\|\mathbf{y})$ Kullbeck-Leibler divergence between random variable $\mathbf{x}$ to $\mathbf{y}$.

$H(\mathbf{x})$ The entropy of random variable $\mathbf{x}$.

$I(\mathbf{x};\mathbf{y})$ The mutual information between $\mathbf{x}$ and $\mathbf{y}$.

Bibliography

F. Abramovich and Y. Ritov. *Statistical theory: a concise introduction*. 2013. ISBN 9781439851845 1439851840.

R. Agraval and R. Srikant. 'Fast Algorithms for Mining Association Rules in Large Data Bases. In *20th Inter-national Conference on Very Large Databases, Santiagom*, 1994.

T. W. Anderson. *An Introduction to Multivariate Statistical Analysis*. Wiley-Interscience, Hoboken, NJ, 3 edition edition, July 2003. ISBN 9780471360919.

I. Borg and P. J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, New York, 2nd edition edition, Aug. 2005. ISBN 9780387251509.

G. Claeskens and N. L. Hjort. *Model Selection and Model Averaging*. Cambridge University Press, Cambridge ; New York, 1 edition edition, July 2008. ISBN 9780521852258.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, New York, softcover reprint of the original 1st ed. 1993 edition edition, May 1994. ISBN 9780412042317.

D. P. Foster and R. A. Stine. Variable Selection in Data Mining. *Journal of the American Statistical Association*, 99(466):303–313, June 2004. ISSN 0162-1459. doi: 10.1198/016214504000000287. URL `http://dx.doi.org/10.1198/016214504000000287`.

Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, Aug. 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL `http://www.sciencedirect.com/science/article/pii/S002200009791504X`.

J. Friedman and J. Tukey. A Projection Pursuit Algorithm for Exploratory Data Analysis. *IEEE Transactions on Computers*, C-23(9):881–890, Sept. 1974. ISSN 0018-9340. doi: 10.1109/T-C.1974.224051.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337–407, Apr. 2000. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1016218223. URL `http://projecteuclid.org/euclid.aos/1016218223`.

M. Gashler, C. Giraud-Carrier, and T. Martinez. Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous. In *Seventh International Conference on Machine Learning and Applications, 2008. ICMLA '08*, pages 900–905, Dec. 2008. doi: 10.1109/ICMLA.2008.154.

D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM*, 35(12): 61–70, Dec. 1992. ISSN 0001-0782. doi: 10.1145/138859.138867. URL `http://doi.acm.org/10.1145/138859.138867`.

A. Goldenberg, A. Zheng, S. Fienberg, and E. Airoldi. A Survey of Statistical Network Models. *Found. Trends Mach. Learn.*, 2(2):129–233, Feb. 2010. ISSN 1935-8237. doi: 10.1561/2200000005. URL `http://dx.doi.org/10.1561/2200000005`.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, July 2003. ISBN 0387952845.

H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933. ISSN 1939-2176(Electronic);0022-0663(Print). doi: 10.1037/h0071325.

A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4–5):411–430, June 2000. ISSN 0893-6080. doi: 10.1016/S0893-6080(00)00026-5. URL `http://www.sciencedirect.com/science/article/pii/S0893608000000265`.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 1st ed. 2013. corr. 4th printing 2014 edition edition, Aug. 2013. ISBN 9781461471370.

W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemp. Math.*, pages 189–206.

Amer. Math. Soc., Providence, RI, 1984. URL http://www.ams.org/mathscinet-getitem?mr=737400.

I. T. Jolliffe. *Principal Component Analysis*. Springer, New York, 2nd edition edition, Oct. 2002. ISBN 9780387954424.

R. E. Kalman. *Contributions to the Theory of Optimal Control*. 1960.

Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, Aug. 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263.

P. McCullagh and J. A. Nelder. *Generalized Linear Models, Second Edition*. Chapman and Hall/CRC, Boca Raton, 2 edition edition, Aug. 1989. ISBN 9780412317606.

C. D. Meyer. *Matrix Analysis and Applied Linear Algebra Book and Solutions Manual*. SIAM: Society for Industrial and Applied Mathematics, Philadelphia, har/cdr edition edition, Feb. 2001. ISBN 9780898714548.

K. Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00116037. URL http://link.springer.com/article/10.1007/BF00116037.

J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117, Jan. 2015. ISSN 08936080. doi: 10.1016/j.neunet.2014.09.003. URL http://arxiv.org/abs/1404.7828. arXiv: 1404.7828.

S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, May 2014. ISBN 9781107057135.

V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, 1 edition edition, Sept. 1998. ISBN 9780471030034.

G. Wahba. *Spline Models for Observational Data*. SIAM, 1990. ISBN 9781611970128.

L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer, New York, 2004. ISBN 0387402721 9780387402727.