Contents

1 First Lecture

1.1 Rationale

- We are interested in quantities Y (*"response"*), but we cannot measure Y directly
- We have access to quantities X (*"features"*), which are related to Y.
- Both X, Y are generally (huge) vectors/matrices
- We see a mapping $f(X) = Y$; but no analytical function $f(\cdot)$ is known
- we therefore use a generic model $f(X|\theta) = Y$, where θ are free parameters that tailor the model to the application
- mapping can be uncertain:
  - intrinsic uncertainty (eg. radioactive decay, chaotic systems)
  - insufficient information in X
- $\Rightarrow$ probabilistic model *"posterior probability"* $p(y|x)$.

1.2 Goal

Select best possible model & parameters for any given application given a set of training data

1.3 Kinds of variables

- $X, Y \in \mathbb{R} \rightarrow$ *numerical*
- $X, Y \in \{A, B, C, ...\} \rightarrow$ *ordinal* if there is an order $A < B < C$ (e.g. "warm", "cold", "hot") or *categorical* otherwise

1.4 Kinds of training data

- supervised learning: we can in principle measure Y, but not in a routine manner
  - we need control conditions & expensive equipment
  - measurement is destructive (eg. crash test)
  - asking an expert is expensive (eg. radiologist)
  - Y is only known in hindsight

  $\Rightarrow$ we have training pairs $(X_i, Y_i)_{i=1,...,N}$ that describe the mapping $f(X_i) = Y_i$ (generating these pairs is called *"labeling / annotation"*)

  batch training use entire training set

  online training update the model as data pairs arrive one at a time

active learning learning algorithm actively prepares X_i whose Y_i would be very instructive

- unsupervised learning: Y_i are unknown, only $(X_i)_i$ are known (also called *"data mining"*)
  - determine useful categories
  - find interesting features and drop uninteresting (aka feature selection, dimension reduction), fight the *"curse of dimensionality"*
  - group data by similarity
  - detect outliers
  - estimate the probability distribution $p(x)$ (aka *"feature density / evidence"*)
- weakly supervised learning: reduce the effort to create training data
  - coarse annotation: "this picture contains an elephant", instead of "this pixel is part of an elephant"
  - delayed annotation: "this sequence of moves lead to a loss" instead of "this is a bad move"
  - Semi supervised learning: combine $(X_i, Y_i)_{i=1,...,N}$ with $(X_j)_{j=N+1,...,M}$.

Summarizing

		response	
		numeric	categorical
training data	supervised	regression	classification
	unsupervised	feature reduction, metric learning	clustering

Second Lesson

Plans for the future

- **Chapter 1** classification
- **Chapter 2** regression and the mathematical theory of ML
- **Chapter 3** How to evaluate the quality of our solution!
- **Chapter 4** unsupervised learning
- **Chapter 5** advances classification methods (support vector machines, random forest & boosting ensemble methods, neural networks)
- **Chapter 6** find project (own topic)

1.5 Notation

- X = features, $X \in \mathbb{R}^D$, X_j a particular feature, $j = 1, ..., D$
- Y = responses, $Y \in \mathbb{R}$, $Y \in \{A, B, ...\}$; if Y is categorical, C is the number of classes, k is the class index, $k = 1, ..., C$ or for $C = 2$: $k \in \{0, 1\}$ or $k \in \{-1, 1\}$
- training instances $i = 1, ..., N$
- arrange training features as a matrix (rows are instances, columns are features)
- arrange training response in a vector
- $f = 1, ..., T$, $l = 1, ..., L$ universal order

1.6 Models

modeling the full posterior $p(Y|X)$ is usually intractable

$\Rightarrow$ naive independence assumptions st $p(Y|X)$ decomposes into the product of simpler probabilities:

$$p(Y|X) = p_A(Y_A|X_A)p_B(Y_B|X_B)p_C(Y_C|X_C)$$

where $X_A \cup X_B \cup X_C = X$, $Y_A \cup Y_B \cup Y_C = Y$.

Fundamental factorization / decomposition: training data (& real data) consist of a set of individual events where we assume:

- all events are independent
- probabilities controlling the events do not change over time ($\Rightarrow$ events identically distributed)

aka *"independent, identically distributed"* data (iid), st

$$p(\{A, B, C\}) = p(A)p(B)p(C)$$

examples for noniid:

- written exam: student looks at neighbor's solutions
- pixels in an image: p(pixel$_i \in$ elephant) is much bigger when neighbors are also in the elephant

$\Rightarrow$ need structured models (eg Markov random fields)

1.7 All models are uncertain!

- intrinsic uncertainty
- modeling error: a model **is not** reality!
- finding optimal model parameters θ may be intractable or difficult
- trade off between these two
- training dataset is finite (no free-lunch theorem)
- features X are noisy

$\Rightarrow$ error analysis is an important part of ML

2 Second Lecture: 21.10.2015

First Lesson

2.1 Classification

Classification: response $Y \in \{0, 1, ..., C-1\}$ no ordering and C classes.
When we have a classifier $\hat{f}(x) = \hat{Y}$, determine the error rate on a test set $(X_i, Y_i)_{i=1,...,N}$.
The classifier predicts $\hat{Y}_i$ for given X_i, giving us the following table

AC/PC	1	2	...	C
1				
2				
⋮				
C				

Table 1: Confusion Matrix

AC: Actual Class, PC: Predicted Class

The error rate is then $\frac{\#\text{errors}}{\#\text{total}}$. Considering the special case of $c = 2$ we call $Y = 0$ "negative" and $Y = 1$ "positive" (eg $Y = 0$ new drug doesn't work, $Y = 1$ new drug does work). We have the following possible outcomes: True Negatives (TN), False Negatives (FN), False Positives (FT), True Positives (TP), giving us for the error rate:

$$\frac{\#FN + \#FP}{\#TN + \#FN + \#FP + \#TP}$$

Possible Questions

1. How bad can a classifier perform:
   - we have no features or features are unrelated to response
   - but we know how often each class occurs on average, also called the *prior probability* aka π_i or $p(Y)$.

   Case 1 All classes occur equally: $\pi_i = 1/C$ Then we have $p(correct) = 1/C$ and $p(error) = (C-1)/C$ as an upper bound for classification

   Case 2 Priors are different: define $i_{\max} = \arg\max_i \pi_i$ most frequent class, giving us an error rate of $p(error) = 1 - \pi_{i_{\max}}$

   general upper bound $p(error) \leq 1 - \max_i \pi_i$

2. How well can we perform:
   - we have features, but they are not perfect $\Rightarrow$ we have intrinsic error
   - ignore all other error sources $\Rightarrow$ lower bound on the error
   - We know the correct posterior $p(y|x) = \hat{p}(y|x)$
   - we are required to make a hard decision $\hat{f}(x)$. best possible decision function $\hat{f}$ minimizes expected error.

     $$p(error) = \mathbb{E}_x[p(error|x)] = \int p(\hat{f}(x) \neq y|X)p(x)dx$$

     The integral is minimized if $p(\hat{f}(x) \neq y|x)$ is minimized for every x.
   - For $c = 2$: we have $\hat{f}(x) = 0 \begin{cases} \text{true negative}, & p(y=0|x) \\ \text{false negative}, & p(y=1|x) \end{cases}$ and similarly for $\hat{f}(x) = 1$.

     $$p(error|x) = \begin{cases} p(y=0|x) & if\ \hat{f}(x) = 1 \\ p(y=1|x) & if\ \hat{f}(x) = 0 \end{cases}$$

     is minimized when $\hat{f}(x) = \arg\max_Y p(y|x)$. This rule is called the *Bayes decision rule*. We get that $p(error) = \int(1 - \max_y p(y|x))p(x)dx$[1]

If X is continuous, we can define:

- decision regions: maximal connected regions in X-space where $\hat{f}(x) = \text{const.}$
- decision boundaries: between decision regions[2]

Bayes decision boundaries are optimal.

In general we have two different kinds of models:

discriminative models learn $\hat{p}(y|x)$ directly (nearest-neighbor classifier, decision tree)

[1] Those two are obviously valid in general not just in our two-case example
[2] on this boundary we have to decide arbitrary

generative models expand using Bayes formula

$$p(Y|X) = \frac{p(X|Y) \cdot P(Y)}{P(X)}$$

we call $p(Y|X)$ the *posterior*, $p(X|Y)$ *likelihood*, $p(Y)$ *prior* and $p(X)$ *evidence* or *feature density*. A generative model learns likelihood, prior & feature density. Thus we can generate more data from the same distribution. Note that for classification, we don't need the evidence since $Y_i = \arg\max_Y P(Y|X) \Leftrightarrow \frac{P(Y=i|X)}{P(Y=j|X)} > 1 \ \forall j \neq i$.[3]

In case of equal sized priors we get the likelihood ratio: $\frac{p(X|Y=i)}{p(X|Y=j} > 1 \ \ \forall j \neq i$

example 1: pens

Second Lesson

$Y \in \{\text{red, black}\}$, $X \in \{\text{ballpen, fineliner}\}$, $p(Y) = \frac{1}{2}$, $p(X) = \frac{1}{2}$. $p(bp|red) = \frac{2}{7}$, $p(fl|red) = \frac{5}{7}$, $p(bp|balck) = \frac{5}{7}$, $p(fl|black) = \frac{2}{7}$. Bayes gives us $p(red|bp) = \frac{2}{7}$ and $p(black|bp) = \frac{5}{7}$.

example 2

One continuous feature: $X \in [0,1], Y \in \{0,1\}$, $p(y=0) = p(y=1) = \frac{1}{2}$[4].
Likelihoods:

$$p(x|y=0) = 2 - 2x \text{ and } p(x|y=1) = 2x$$

Feature density:

$$p(x) = p(x|y=0)p(y=0) + p(x|y=1)p(y=1) = 1$$

and finally the posteriors are:

$$p(y=0|x) = \frac{(2-2x)\frac{1}{2}}{1} = 1 - x \text{ and } p(y=1|x) = \frac{2x \cdot \frac{1}{2}}{1} = x$$

This gives us the following decision strategy:

- Select threshold $t \in [0,1]$.
- rule a: $\hat{f}(x) = 0 \Leftrightarrow x \leq t$
- rule b: $\hat{f}(x) = 1 \Leftrightarrow x \leq t$

We have the following errors:

$$p(error|x \leq t) = \begin{cases} p(y=1|x) & \text{if rule } a \\ p(y=0|x) & \text{if rule } b \end{cases} \text{ and } p(error|x > t) = \begin{cases} p(y=0|x), & \text{if } a \\ p(y=1|x), & \text{if } b \end{cases}$$

[3] This ratio is also called the *odds*, or the *log-odds* if you take the logarithms of this
[4] giving us a so called *uninformative prior*

$$\begin{aligned} p_a(error) &= \mathbb{E}_x[p(error|x)] = \int_0^\top p(y=1|x)p(x)dx + \int_t^1 p(y=0|xp(x)dx \\ &= \int_0^\top (1-x)dx + \int_t^1 xdx = \frac{1}{4} + \left(t - \frac{1}{2}\right)^2 \end{aligned}$$

The expected error for rule b (working likewise) $p_b(error) = \frac{3}{4} - (t - \frac{1}{2})^2$. Such that we get $p_b(error|t) = 1 - p_a(error|t)$

The error is minimal for $p_a(error|t = 1/2) = 1/4$, which is called the *"Bayes error"*. The corresponding rule is called *"Bayes decision rule"*. Always place the decision boundary where the posteriors intersect, in this case 1/2.

3 Third Lecture: 23.10.2015

First Lesson

- worst case error: pure guessing $p(error) \leq \frac{c-1}{c}$ (c number of classes)
- best case error: ignore all error sources except intrinsic error: Bayes error

  $$p(error) = \int (1 - \max_y p(y|x))p(x)dx$$

  This[5] is also known as the maximum a posteriori decision (MAP).
- Bayes rule

  $$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

- if $p(y) = \frac{1}{c}$ we have an uninformative prior then we can equivalently use the maximum likelihood decision (ML) giving us $p(error) = 1 - \frac{1}{c}\int \max_y p(x|y)dx$

Nearest-Neighbor Classifier

- Intuitive idea: in an unknown situation, act as you did in the most similar situation in the past.
  - "most similar": minimum distance in feature space
  - "act as you did": copy the class label Y of the nearest training instance

  Let $(X_i, Y_i)_{i=1,\ldots,N}$ be the training data, then:

  $$\hat{f}(X) = Y_k | k = \arg\min_j d(X, X_j)$$

  is our rule. The effect is that we split the feature space according to the Voronoi tessellation:

  $$\text{neighbors}(X_k) = \{X | d(X, X_k) < d(X, X_j) \ \ \forall j \neq k\}$$

  $\Rightarrow$ decision boundaries: bisectors between neighboring training points of different classes

[5] the first term in the integral

3.0.1 How well does NN work?

Second Lesson

- The NN classifier memorizes all training instances ⇒ error on the training set (TS error): 0

  ⇒ only the out of training set error (OTS) error is interesting, or iid error (TS + OTS)

- finite training set:
  - only few general results exist: OTS error can be arbitrary high *("no free lunch theorem"[6])*
  - in simple cases analytic expressions exist
  - example: $p(x|y=0) = 2-2x$, $p(x|y=1) = 2x$, $p(y) = \frac{1}{2}$, $p(y=0|x) = 1-x$, $p(y=1|x) = x$. $p^*(error) = \frac{1}{4}$

    assumption. One prototype for each class, $(x_0, y_0 = 0)$, $(x_1, y_1 = 1)$. Then we have the bisector $t = \frac{x_0+x_1}{2}$. We have
    - rule a: $x_0 < x_1$
    - rule b: $x_0 > x_1$

    Then

$$p(error) = \int_0^1 p(error|rule\ a, t) \cdot p(rule\ a, t|x_0, x_1)dt + \int_0^1 p(error|rule\ b, t)p(rule\ b, t|x_0, x_1)dt$$

  - to get threshold t and rule (a/b), we must have $x_0 = t(-/+)x'$, $x_1 = t(+/-)x'$, $x' \geq 0$.

$$\Rightarrow \quad p(rule\ a, t|x') = 2 \cdot p(x_0 = t - x'|y=0) \cdot p(x_1 = t + x'|y=1)$$

    Integration boundaries: $0 \leq x' \leq \begin{cases} t & \text{if } t \leq \frac{1}{2} \\ 1-t & \text{if } t \geq \frac{1}{2} \end{cases}$

    [*...and so on...*]

    We get an error of 35%.

- asymptotic analysis for $N \to \infty$
  - Definition: a classification rule is called consistent if it converges towards the Bayes rule as $N \to \infty$.
  - (Derivation: Duda, Hart, Starl. Chapter 4.5)
  - Result: $p^* \leq p_\infty \leq p^*(2 - \frac{c}{c-1}p^*)$ where p^* is the Bayes error, p_∞ NN class $\mathbb{N} \to \infty$

[6] topic of a later lecture

  - e.g. $p^* = 0$, then $p_\infty = 0$,

    $p^* = \frac{c-1}{c}$, then $p_\infty = \frac{c-1}{c}$

    $p^* = \varepsilon$, then $p_\infty = \varepsilon(2 - \frac{c}{c-1}\varepsilon) \approx 2\varepsilon$

- empirical/experimental error analysis
  - TS error is meaningless (=0) ⇒ need an OTS error
  - $p(OTS\ error) = h(TS\ error) + \omega$, where $h(TS\ error)$ is called the empirical error (or TS error), and ω is the "model optimism".
  - A model with big optimism "overfits" the training data

    ⇒ split the database into TS and OTS, where you train on the TS and measure the error on the OTS. This split needs to be done at random; flip roles, repeat and compute the average error of both trials (called 2-fold cross validation)
  - better average (+ a variance) of the error: more repetitions: k-fold cross-validation
    1. Split data D into K groups (at random)
    2. For each $j \in \{1, ..., K\}$: Train on D_{-j} and evaluate on D_j.
    3. Average the error results and variance

    Typical values are $k = 5, 10$ or $k = N$ aka leave one out cv (LOOCV)

4 Fourth Lecture

First Lesson

Nearest-Neighbor Classifier

- advantages:
  - simple & intuitive concept
  - easy to implement
  - often works well in practice, esp. if one has a lot of data
- problems:
  - NN classifier is *not* consistent [$p_\infty \approx 2p^*$] ⇒ use k-NN method
    - find the k nearest training instances to the query
    - predict the majority vote of those k instances
    - usually smaller error than 1-NN
    - consistent when $k \to \infty$, $\frac{k}{N} \to 0$ (as $N \to \infty$), ex: $k = \log N$, $k = \sqrt{N}$
  - speed problem: naive implementation $\mathcal{O}(d \cdot N)$, where d = #features, N = #instances (sequential search for NN)
    - reduce d:

  - · feature selection
  - · dimension reduction
- ∗ Reduce N:
  - · remove unnecessary instances
  - · drop instances whose neighbors have all the same class ⇒ requires exact computation of the Voronoi tessellation $\mathcal{O}(N^{d/2})$ ⇒ too expensive when d big
- ∗ approximate algorithm:
  - · maintain an active set of instance (initially empty)
  - · for all instances: compute NN prediction w.r.t. current active set and if correct ⇒ drop training point, otherwise add it to the active set
- ∗ between these extremes:
  - · clustering algorithms, vector quantization ⇒ later
- ∗ speed-up the algorithm
  - · use an optimal data structure instead of sequential search:

    "k-D tree" generalizes binary search tree to k dimensions

    search complexity: $\mathcal{O}(d : \min(N, \log N \cdot 2^d))$ [7]
  - · use early stopping in distance computation
  - · use approximate NN search: since measurements are noisy insisting on exact NN is pointless
- ∗ locality preserving hashing
- ∗ minimize the hidden constant in $\mathcal{O}(d \cdot N)$
- ∗ use 64-bit hash functions ⇒ comparison takes only 1 CPU clock cycle ⇒ sequential search is very fast

– The NN depends on the distance function: units matter so always standardize (and center) your units

– use domain knowledge to design a good distance ⇒ later

– use metric learning

4.1 Quadratic and Linear Discriminant Analysis (QDA, LDA)

Second Lesson

- minimum number of training instance for NN: 1 per class

  ⇒ Use average of each class: $m_k = \frac{1}{N_k} \sum_{j:Y_j=k} X_j$, where $N_k = \#$ samples in class k, giving us a nearest mean classifier, which works well for spherical clusters [8]

[7] only use for $d \leq 20$

[8] the lecture contains some sketches here

  ⇒ we need to consider the cluster shape, and thus generalize the sphere to the ellipsoid. In terms of degrees of freedom we have for a sphere $d+1 \in \mathcal{O}(d)$ (center + radius) and for the ellipsoid $d + \frac{d}{2}(d+1) \in \mathcal{O}(d^2)$ (center + orientation & shape)

  ⇒ choose one multi-variate Gaussian per class
  - – ellipsoid shaped clusters
  - – easy to train
  - – CLT: superposition of infinitely many independent random events has Gaussian distribution

4.1.1 QDA

- assumption: likelihood $p(x|y = k)$ is (approximately) Gaussian for each class k
- prediction rule: for query point X
  - – compute posterior $p(y = k|x) = \frac{p(x|y=k)p(y=k)}{p(x)}$ and predict class with maximum posterior[9]

Form of the Gaussian distribution[10]

$$p(x|\mu, \Sigma) = (2\pi)^{-d/2}|\Sigma|^{-1/2} \exp\left(-\tfrac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

How to fit a Gaussian to training data? The most common method is the maximum likelihood method

- assume that the training data are representative (i.e. very typical) for the true distribution

  ⇒ training set has high probability under the distribution

  ⇒ look for the distribution (among the allowed choices, i.e. Gaussians) that maximizes the probability of the data

5 Fifth Lecture

First Lesson

- generative method: $p(Y|X) = \frac{P(X|Y)p(Y)}{P(X)}$
- parametric method: fix type of distribution (e.g. multivariate Gaussian) and only learn its parameters
- multivariate Gaussian:

$$p(X|m, S) = (\det(2\pi S))^{-1/2} \exp\left(-\tfrac{1}{2}(x-m)^\top S^{-1}(x-m)\right)$$

[9] we assume $p(y = k) = \frac{N_k}{N}$

[10] not the notation of the lecture

- maximum likelihood principle: choose parameters such that the likelihood of the observed data is maximized.
- Let $\{x_i\}_{i=1,..,N_k}$[11] be the training instances with $Y_i = k$.

$$p(x_1, ..., x_n) \overset{\text{iid}}{=} p(x_1) \cdots p(x_N) = p(x_1|m, S) \cdots p(x_N|m, S)$$

- we often work with the log-likelihood instead:

$$\begin{aligned} L = \log p(x_1, ..., x_n) &= \log \prod_i p(x_i|m, S) \\ &= \sum_i \log p(x_i|m, S) \\ &= \sum_i \left[\log \frac{1}{\sqrt{\det(2\pi S)}} - \frac{1}{2}(x_i - m)^\top S^{-1}(x_i - m) \right] \\ &= -\frac{N}{2} \log \det(2\pi S) - \frac{1}{2} \sum_i (x_i - m)^\top S^{-1}(x_i - m) \end{aligned}$$

Maximizing this log-likelihood we get:

$$\begin{aligned} 0 = \frac{\partial}{\partial m} L &= \sum S^{-1}(x_i - m) \\ \Rightarrow \hat{m} &= \frac{1}{N} \sum_i x_i \qquad \text{(aka sample average)} \end{aligned}$$

and for the variance matrix we get:

$$\hat{S} = \frac{1}{N} \sum_i (x_i - m)(x_i - m)^\top \qquad \text{(aka sample covariance)}$$

[for the rest of this computation see any old ML book (for example Bishop's PRML)]

- Some matrix calculus rules that are useful to derive this result:

$$\frac{\partial}{\partial A} \log \det A = (A^\top)^{-1}$$
$$\frac{\partial}{\partial A} \text{tr}(A \cdot B) = B^\top$$

5.1 QDA Learning Algorithm

1. for each class $k = 0, ..., C - 1$ [12] compute the class mean $\hat{m}_k = \frac{1}{N_k}$
   a) compute the class mean $\hat{m}_k$ and variance $\hat{S}_k = \frac{1}{N_k} \sum_{i:y_i=k} (x_i - \hat{m}_k)(x_i - \hat{m}_k)^\top$
   b) define likelihood $p(x|y = k) = p(x|\hat{m}_k, \hat{S}_k)$
   c) compute prior $p(y = k) = \frac{N_k}{N}$

[11] to simplify notation N_k is sometimes abbreviated to N
[12] C:=# of classes

5.2 QDA prediction

- new sample x :

$$\begin{aligned} \Rightarrow \hat{y} = \arg\max_K \frac{p(x|y = k)p(y = k)}{p(x)} &\Leftrightarrow \arg\max_K p(X|Y = k)p(y = k) \\ &\Leftrightarrow \left[\arg\max_k \log p(X|Y = k) + \log p(Y = k) \right] \end{aligned}$$

$$\begin{aligned} &\Leftrightarrow \arg\max_k \left[-\frac{1}{2} \log \det(2\pi \hat{S}_k) - \frac{1}{2}(X - m_k)^\top \hat{S}_k^{-1}(X - m_k) + \log p(y = k) \right] \\ &\Leftrightarrow \arg\max_k \left[-\frac{1}{2}(X - \hat{m}_k)^\top \hat{S}_k^{-1}(x - \hat{m}_k) - b_k \right] \\ &\Leftrightarrow \arg\min \left[\frac{1}{2}(x - \hat{m}_k)^\top \hat{S}_k^{-1}(x - \hat{m}_k) + b_k \right] \end{aligned}$$

where $-b_k = -\frac{1}{2} \log \det(2\pi \hat{S}_k) + \log p(y = k)$

Special case: $S = \mathbb{1}\sigma^2$, then QDA : $\arg\min_k (x - \hat{m}_k)^\top (x - \hat{m}_k) + b'_k$ where the first product is the squared Euclidean distance $d^2(X, \hat{m}_k)$. If the sample sizes are also the same, (i.e. $\frac{N_k}{N} = \frac{1}{C}$) we get the nearest mean classifier.

$$(X - m)^\top S^{-1}(X - m) = d_S^2(X, m)$$

is called the *"Mahalanobis distance "*, it is a generalization of the euclidean distance.
$\Rightarrow$ QDA is the nearest mean classifier under Mahalanobis distance. [13]

5.3 Linear Discriminant Analysis (LDA)

- Suppose clusters for all classes have the same shape $\Rightarrow \forall k : S_k = S_w$ (aka. "within class covariance")

  QDA: $\arg\max_k \left[-\frac{1}{2}(x - \hat{m}_k)^\top \hat{S}_w^{-1}(x - \hat{m}_k) - b_k \right]$

  expanding we get

$$-\frac{1}{2} \left(x^\top \hat{S}_w^{-1} x - \hat{m}_k^\top \hat{S}_w^{-1} x - x^\top \hat{S}_w^{-1} \hat{m}_k + \hat{m}_k^\top \hat{S}_w^{-1} \hat{m}_k \right)$$

  giving us for the LDA:

$$\arg\max_k \left[x^\top \hat{S}_w^{-1} \hat{m}_k + b'_k \right] = \arg\max_k \left[w_k^\top x + b'_k \right]$$

  where $w_k = \hat{S}_w^{-1} \hat{m}_k$ giving us a weighted sum of the features in the first summand.[14]

[13] while Euclid treats all axis the same, Mahalanobis gives different weights to the different axis
[14] since b'_k is derived from theory it often won't be appropriate, so we estimate it practically (through CV)

We have

$$\hat{S}_w = \frac{1}{N}\sum_i (X_i - \hat{m}_{Y_i})(X_i - \hat{m}_{Y_i})^\top$$

i.e. X_i is "compared" to the local class mean and not the global one.

Concentrating on only two classes ($C = 2$) we have:

$$\begin{aligned}\arg\max\left(w_0^\top X + b_0, w_0^\top x + b_1\right) \Leftrightarrow \text{ decide } \hat{Y} = 1 \quad \text{if } (w_1^\top x + b_1) - (w_0^\top x + b_0) > 0 \\ \Leftrightarrow (w_1^\top - w_0^\top)x + (b_1 - b_0) > 0 \\ \Leftrightarrow w^\top x + b > 0\end{aligned}$$

Decide $\hat{y} = 0$ if $w^\top x + b < 0$, where $w = \hat{S}_w^{-1}(\hat{m}_1 - \hat{m}_0)$

6 Sixth Lecture

First Lesson

6.1 Linear Discriminant Analysis (LDA)

- two class case: decision rule

$$\hat{y} = \begin{cases} 1, & \text{if } w^\top x + b \geq 0 \\ 0, & \text{if } w^\top x + b < 0 \end{cases}$$

  where $w = S_w^{-1}(m_1 - m_0)$ and $b =$ threshold [15]

- properties:
  - $w' = \tau w$, $b' = \tau b \Rightarrow$ decision doesn't change
  - let $\mu_k = w^\top m_k + b$ be the projected means, $\sigma^2 = w^\top S_w w$ projected covariance[16]
  - $w = S_w^{-1}(m_1 - m_0)$ maximizes separability $\frac{(\mu_1 - \mu_0)^2}{\sigma^2}$
  - 2-class LDA also follows from least-squares optimization

$$\hat{w} = \arg\min_w \sum_{i=1}^N (\underbrace{w^\top x_i + b}_{\text{prediction}} - \underbrace{z_i}_{\text{truth}})^2$$

    if the classes are balanced ($N_0 = N_1$): $z_i = \begin{cases} 1, & \text{if } y_i = 1 \\ -1 & \text{if } y_i = 0 \end{cases}$, if they aren't we set $z_i = \begin{cases} \frac{N}{N_1}, & \text{if } y_i = 1 \\ -\frac{N}{N_0}, & \text{if } y_i = 0 \end{cases}$.

[15] choose via CV

[16] a one-dimensional example picture follows here

Set derivatives with respect to w and b to zero and solve for w and b.

$$\left(NS_w + \frac{N_0 N_1}{N} S_B\right) w = N(m_1 - m_0)$$

where $S_B = (m_1 - m_0)(m_1 - m_0)^\top$ is called the *between-class covariance* giving us

$$NS_w w + \underbrace{\frac{N_0 N_1}{N}(m_1 - m_0)\left((m_1 - m_0)^\top w\right)}_{\tau(m_1 - m_0),} = N(m_1 - m_0)$$

$$\begin{aligned} NS_w w &= \tau'(m_1 - m_0) \\ S_w w &= \tau''(m_1 - m_0) \\ w &= \tau'' S_w^{-1}(m_1 - m_0) \end{aligned}$$

6.2 Logistic Regression

Second Lesson

- misnomer, actually it is classification
- LDA and QDA are generative models (learn likelihood and prior = RHS of Bayes)
- logistic regression makes the same assumptions as LDA (Gaussian likelihoods, common covariance S_w) but as a discriminative model (learn posterior = LHS of Bayes)
- stick with two class case $\Rightarrow$ we only have to learn <u>one</u> posterior, because $p(y = 0|x) = 1 - p(y = 1|x)$
- for simplicity: balanced data, $N_0 = N_1$ and we center the data $x_i' = x_i - m$, where $m = \frac{1}{N}\sum_i x_i$. Then $\frac{1}{N}\sum_i x_i' = 0$ and $m_0 + m_1 = 0$ or $m_0 = -m_1$

$$\begin{aligned} p(y = 1|x) &= \frac{p(x|y = 1)p(y = 1)}{p(x|y = 0)p(y = 0) + p(x|y = 1)p(y = 1)} \\ &= \frac{p(x|y = 1)}{p(x|y = 0) + p(x|y = 1)} \qquad \text{since } N_0 = N_1 \\ &= \frac{(\det(2\pi S_w))^{-1/2}\exp\left(-\frac{1}{2}(x - m_1)^\top S_w^{-1}(x - m_1)\right)}{(\det(2\pi S_w))^{-1/2}\exp\left(-\frac{1}{2}(x - m_0)^\top S_w^{-1}(x - m_0)\right) + \ldots} \\ &= \frac{\exp\left(-\frac{1}{2}(x - m_1)^\top S_w^{-1}(x - m_1)\right)}{\exp\left(-\frac{1}{2}(x + m_1)^\top S_w^{-1}(x + m_1)\right) + \exp\left(-\frac{1}{2}(x - m_1)^\top S_w^{-1}(x - m_1)\right)} \end{aligned}$$

Using $\exp\left(-\frac{1}{2}(x-m_1)^\top S_w^{-1}(x-m_1)\right) = \exp(-\frac{1}{2}x^\top S_w^{-1}x)\exp(m_1^\top S_w^{-1}x)\exp(-\frac{1}{2}m_1^\top S_w^{-1}m_1)$ we get

$$\begin{aligned} p(y=1|x) &= \frac{\exp(m_1^\top S_w^{-1}x)}{\exp(m_1^\top S_w^{-1}x) + \exp(-m_1^\top S_w^{-1}x)} \\ &= \frac{1}{1+\exp(-2\underbrace{m_1^\top S_w^{-1}}_{w^\top}x)} \end{aligned}$$

which is known as the logistic regression equation[17].

- learn optimal w by the maximum likelihood rule on the posterior and choose w that maximizes the probability of the training labels under the model

$$\begin{aligned} p(y_1,...,y_N) &=^{iid} \prod_{i:y_i=1} p(y=1|x_i) \prod_{i:y_i=0} p(y=0|x_i) \\ &= \prod_{i:y_i=1} p(y=1|x_i) \prod_{i:y_i=0} (1-p(y=1|x_i)) \\ &= \prod_i p(y=1|x_i)^{y_i}(1-p(y=1|x_i))^{1-y_i} \end{aligned}$$

  Computing the log-likelihood simplifies this to:

$$\begin{aligned} LL = \log p(y_1,...,y_N) &= \sum_{i=1}^N \Big(y_i \log p(y=1|x_i) + (1-y_i)\log(1-p(y=1|x_i)) \Big) \\ &= \sum_{i=1}^N \Big(y_i \log \sigma(w^\top x_i) + (1-y_i)\log(1-\sigma(w^\top x_i)) \Big) \end{aligned}$$

  Learning logistic regression model:

$$\hat{w} = \arg\min_w -LL$$

  compared to LDA learning:

$$\hat{w} = \arg\min_w \sum_{i=1}^N (w^\top x_i - z_i)^2$$

  in both cases: decision $\hat{y}=1 \Leftrightarrow w^\top x(+b) \geq 0$ **but the optimal** w **differ**, because the objective functions of the learning algorithm differ[18].

- Solution to the LR learning problem: no closed form expression $\Rightarrow$ iterative algorithm

[17] remember the logistic sigmoid function: $\sigma(a) = \frac{1}{1+exp(-a)}$

[18] we will later learn about SVMs which provide again a similar decision rule but optimize yet another objective function.

- properties of logistic function:
  - $\sigma(a) = \frac{1}{1+e^{-a}} = (1+e^{-a})^{-1}$
  - $1-\sigma(a) = \frac{e^{-a}}{1+e^{-a}}$
  - $\frac{d}{da}\sigma(a) = \sigma(a)(1-\sigma(a))$
- we have $\frac{\partial}{\partial w} - y_i \log \sigma(w^\top x_i) = \frac{-y_i}{\sigma(w^\top x_i)}\sigma'(w^\top x_i)x_i^\top = -y_i(1-\sigma(w^\top x_i))x_i^\top$ for the first term and $\frac{\partial}{\partial w} -)1-y_i)\log(1-\sigma(w^\top x_i)) = (1-y_i)\sigma(w^\top x_i)x_i^\top$ and finally get

$$\frac{\partial}{\partial w}\text{LR-Objective} = \sum_{i=1}^N (y_i - \sigma(w^\top x_i))x_i^\top = 0$$

- consider 1D, let $x_i > 0$ with $\sigma(wx_i)$ as the predicted label and y_i the desired label. Then $(y_i - \sigma(wx_i))$ is positive when the prediction is too low and negative when it is too high.

Solution by stochastic gradient descent

1. choose initial w (e.g. $w=0 \Rightarrow \sigma(w^\top x) = \frac{1}{2}$) and learning rate κ
2. for $t = 0,...,t_{\max}$ or until convergence (also known as outer iterations or "epochs")
   a) order the data randomly (e.g. by calling numpy.random.shuffle() on index array) [19]
   b) for all i in shuffled order: $w = w + \kappa\sigma(w^\top x_i)x_i$
   c) (optional) update the learning rate: $\kappa = \kappa\frac{t}{t+1}$

7 Seventh Lecture

First Lesson

7.1 Logistic Regression

- Learn the posterior of an LDA-like model
- Stochastic gradient ascent: good for Big Data
- many alternatives:

  first order (plain) gradient ascent, conjugate gradient ascent

  second order Newton, quasi-Newton (BFGS)

  dual, primal-dual

  For more on those see the chapter on Support Vector Machines

[19] IMPORTANT: otherwise you might get biased results

- Intuition about Newton: Iterative Re-weighted Least-Squares (IRLS)

  alternating optimization: given an initial guess $w^{(0)}$ repeat until convergence:

  1. given $w^{(t)}$ determine weights λ_i for all instances, such that instances near decision boundary get high λ_i[20].

     $$\lambda_i = \sigma'(a)\Big|_{w^{(t)\top} x_i}$$

  2. given $\lambda_1^{(t)}$ compute $w^{(t+1)}$ by a weighted linear least squares fit of *the straight part of* $\sigma(a)$

- compare to LDA:
  - both use the same mathematical for the decision function $\hat{y} = \text{sign}(w^\top x + b)$
  - but the optimal w differ because we optimize against different objectives:

    $$LDA: \quad \arg\min_w \sum_i (w^\top x_i - y_i)^2$$

    $$LR: \quad \arg\min_w \sum_i y_i \log \sigma(w^\top x_i) + (1 - y_i) \log(1 - \sigma(w^\top x_i))$$

  - in practice: LDA better when data are (approx) Gaussian, LR otherwise

7.2 Lecture so far

	generative	discriminative
parametric	QDA / LDA	Logistic regression
non-parametric	Histograms; Density Trees	KNN

7.3 Histograms and Density Trees

first look a the 1-dimensional case

- when features X are discrete: histogram trivial: just count the frequency of each feature. and then for example

$$p(x|y=k) = \begin{cases} N_{low}/N \\ N_{medium}/N \\ N_{high}/N \end{cases}$$

[20] the lecture contains an example graph here

[20] quick reminder:

generative learn RHS of Bayes formula (likelihood & prior)

discriminative learn LHS of Bayes (posterior) or just learn the decision boundaries (easier: when we decide $\hat{y} = k$, we don't care if the probability was 0.8 or 0.9

parametric fix the type of distribution (i.e. Gaussian) and learn its parameters (i.e. m, S)

non-parametric don't fix the distribution

- when feature X is continuous: discretize giving us the following:

$$x_l = x_0 + l \cdot \underbrace{\Delta x}_{\text{bin width}}$$
$$bin_l = [x_l, x_{l+1})$$

  where $x_0 \leq x_{\min}$.
  - optimal bin height given x_0 and Δx
  - minimize

$$\int_{-\infty}^{\infty} (\underbrace{p(x)}_{\text{truth}} - \underbrace{\hat{p}(x)}_{\text{hist}})^2 dx = \int p(x)^2 dx - 2 \int p(x)\hat{p}(x) dx + \int \hat{p}(x)^2 dx$$

    where

$$\hat{p}(x) = \sum_{l=0}^{L-1} \underbrace{p_l}_{\text{height of bin } l} \cdot \mathbb{1}[x_l \leq x \leq x_{l+1}]$$

    Note that:

$$\mathbb{1}[x \in bin_l] \cdot \mathbb{1}[x \in bin_{l'}] = \begin{cases} \mathbb{1}[x \in bin_l], & \text{if } l = l' \\ 0, & \text{if } l \neq l' \end{cases}$$

    Giving us:

$$\begin{aligned} \int p(x)\hat{p}(x) &= \int p(x) \sum_l p_l \mathbb{1}[...] dx \\ &= \sum_l p_l \int p(x) \mathbb{1}[...] dx \\ &= \sum p_l \underbrace{\int_{x_l}^{x_{l+1}} p(x) dx}_{\approx \frac{N_l}{N}} \end{aligned}$$

$$\begin{aligned} \int \hat{p}(x)^2 dx &= \int \left(\sum_l p_l \mathbb{1}[x \in bin_l] \right)^2 dx \\ &= \int \sum_l p_l^2 \mathbb{1}[x \in bin_l] dx \\ &= \sum_l p_l^2 \underbrace{\int \mathbb{1}[x \in bin_l] dx}_{\Delta x} \end{aligned}$$

so that finally we get:

$$L(p_0, ..., p_{l-1}) := \min_{\hat{p}} \int (p(x) - \hat{p}(x))^2 \, dx$$
$$= \min_{\hat{p}} \int p(x)^2 dx - 2 \sum_l p_l \frac{N_l}{N} + \sum_l p_l^2 \Delta x$$

Then

$$0 = \frac{\partial}{\partial p_l} L(p_0, ..., p_{l-1}) = -2\frac{N_l}{N} + 2p_l \Delta x$$

giving us the optimal bin height of $\hat{p}_l = \frac{N_l}{N\Delta x}$

Second Lesson

optimal Δx:

* if X is Gaussian distributed[21]: Scott's rule: $\Delta x = \frac{3.5\sigma}{\sqrt[3]{N}}$
* Freedman- Diaconis rule for arbitrary distributions: $\Delta x = \frac{2IQR(x_1,...,x_N)}{\sqrt[3]{N}}$ where IQR is the inter-quartile range. To compute it: sort $x_1, ..., x_N$ giving you $x_{[1]}, ..., x_{[N]}$. Then $N_{1/4} = \frac{N}{4}$ and $N_{3/4} = \frac{3N}{4}$.[22] Then

  $$IQR : X_{[N_{3/4}]} - X_{[N_{1/4}]}$$

* Shimazaki / Shinomoto method: use candidate set $\{\Delta x_1, ..., \Delta x_R\}$ and build a histogram with each Δx_r, choose the one that minimizes

  $$\frac{2m_r - v_r}{(\Delta x_r)^2}$$

  where m_r and v_R are the mean and variance of bin height in histogram r
* cross-validation: candidate set $\{\Delta x_1, ..., \Delta x_R\}$, test data $\frac{M_l}{M}$: the fraction of test points in bin l. Choose Δx that minimizes $\sum_l \frac{1}{\Delta x_r} \left(\frac{N_l}{N} - \frac{M_l}{M}\right)$ with $\frac{M_l}{M} \approx \Delta x_l \cdot p(x \in bin_l)$

Now X is d-dimensional, leading to a d-dimensional histogram. Using L bins per dimension we get for the total number of bins $L_{tot} = L^d$.

If we have $N < L^d$

- then most bins are empty $\Rightarrow$ no density estimate in these bins.
- most remaining bins have few instances: $\Rightarrow$ estimate is very inaccurate

$\Rightarrow$ The approach won't work

8 Eight Lecture

First Lesson

Generative Nonparametric Classification

- 1-D case: learn one histogram per class: bin width: $\Delta x = \frac{2IQR(X)}{\sqrt[3]{N}}$, bin height/density: $\hat{p}_l = \frac{N_l}{N\Delta x}$
- d-D case: histograms fail when $d \geq 5$ because there are too many empty bins

Solution 1: naive Bayes method assume that the features are independent, given the class:

$$p(x_1, ..., x_d | Y = k) = \prod_{j=1}^{d} p_j(x_j | Y = k)$$

$\Rightarrow$ we only learn d 1-dimensional likelihoods per class

- $p_j(x_j|y) \sim \mathcal{N}(x_j | m_j, \sigma_j^2) \Rightarrow$ naive Bayes $\hat{=}$ QDA with diagonal covariance [23]
- $p_j(x_j|y)$ is a 1-dimensional histogram
- Prediction:
  - determine likelihood for each feature and class from the individual histograms p_{ljk}[24]
  - full posterior for class k: $p(y = k) = \prod_{j=1}^{d} p_{ljk} \cdot p(y = k)$
  - decide $\hat{y} = \arg\max_k p(y = k|x)$
  - numerically better: compute $\sum_{j=1}^{d} \log p_{ljk} + \log p(y = k)$

Solution 2: density tree

- idea: determine bins adaptively for the joint distribution of the data
- simple greedy algorithm: recursive subdivision
- Consider one class at a time
  1. put all instances into the root node of the tree; place the root node on a stack
  2. while stack not empty:
     a) take top node n from stack
     b) check termination condition on n
     c) if true: turn n into a terminal node (compute $\hat{p}_l = \frac{N_l}{NV_l}$, where V_l is the volume of bin l): else: turn n into an interior node

[21] If you know it's Gaussian, better fit a Gaussian than a histogram
[22] each rounded down (but this doesn't really matter)
[23] i.e. off-diagonals are all 0
[24] l = bin, j = feature, k = class

- find the optimal split of n's box in feature space
- Assign n's instances to two children according to the split
- place the children on the stack

- termination condition: stop splitting when:
  1. maximum tree depth is reached (user parameter)
  2. minimum number of instances in the root (user parameter)
  3. minimum density is reached (user parameter): $p_l \geq p_{min}$
  4. improvement of the error $\int (p(x) - \hat{p}(x))^2 dx$ is too small
- split selection: restrict to axis-aligned splits $\Rightarrow$ the split only depends on one feature and is cheap to compute during prediction.

  $$x \in \text{left child } : x_j \leq t_l \quad x \in \text{right child } : x_j > t_l$$

  each interior node stores its feature index j and threshold t_l
- greedy exhaustive search for best split: try all possible splits on all features and choose the best one.

  $$\text{Loss} : \int (p(x) - \hat{p}(x))^2 dx \text{ is minimized by } \hat{p}(x) = \sum_l \underbrace{p_l}_{=N_l/(NV_l)} \mathbb{1}(x \in bin_l)$$

  $$= \int p(x)^2 - \sum_l p_l^2 V_l \qquad \Rightarrow \text{minimize}$$

  for fixed binning.
- contribution of bin l to error reduction: $-Loss_l = p_l^2 V_l \Rightarrow$ maximize

  after split $-Loss_l = p_{l_{left}}^2 V_{l_{left}} + p_{l_{right}}^2 V_{l_{right}} = \left(\frac{N_{l_{left}}}{N_l}\right)^2 \frac{1}{V_{l_{left}}} + \left(\frac{N_{l_{right}}}{N_l}\right)^2 \frac{1}{V_{l_{right}}}$

  $\Rightarrow$ maximize by the split
- simplify: X is 1-d, $x \in [0,1]$ $V_l = 1, V_{l,left} = t, V_{l,right} = 1 - t$, $\frac{N_{l,left}}{N_l} =: q$, $\frac{N_{l,right}}{N_l} = 1 - q$, $t \in [0,1], q \in [0,1]$

  $\Rightarrow \text{Loss}_l = \frac{q^2}{t} + \frac{(1-q)^2}{(1-t)^2} \Rightarrow$ maximize

  Since Loss_t is convex in $t \in (0,1)$ it has exactly one minimum at $t = q$ but we want to maximize and so for fixed q and any interval $t \in [t_0, t_1]$ the maximum at t_0 or t_1
- Split selection algorithm:
  - for each feature j :
    - sort the N_l instances i according to feature X_{ij}[25].
    - Check all $t \in \{x_{ij} \pm \varepsilon | j \text{ fixed}, i \in [1, ..., N_l]\}$ $2N_l - 2$ possible splits
    - compute the accuracy gain (i.e. $-$Loss)
  - select the best (j,t) among the $d(2N_l - 2)$ possibilities

[25] $X_{[1]j}, ..., X_{[N_l]j}$

Remarks

- The same algorithm can be applied as a discriminative model $\Rightarrow$ decision tree: known since the 1960s (density trees: 2010). There you need a different gain/loss criteria
- density trees tend to overfit:
  1. improvement: pruning
     - for each subtree: estimate the optimism (held out test set) or use analytical upper bounds
     - replace the subtree by a single terminal node when optimism is too big
  2. improvement: ensemble learning
     - randomize the training algorithm (work on random subsets of instances and features, when selecting splits)
     - train T different trees
     - take the average of the T trees

9 Lecture 10/11

9.1 Generative non-parametric models

- naive Bayes method
- density tree
- density tree $\rightarrow$ density forest (later: random forest)
- kernel density estimation: place one PDF at each instance and take their superposition;

  Problem: how to determine the width of these PDFs?
- Gaussian mixture model $\Rightarrow$ later: EM algorithm
- nested one - class SVMs

9.2 Combine density trees and naive Bayes

- preliminary: consider data standardization

  $$X'_j = \frac{X_j}{\sigma_j}$$

  where σ_j represents the standard deviation of X_j. And instead of $p(X_j)$ we get $\tilde{p}(X'_j)$. As usual

  $$\int p(X_j)dX_j = 1 \qquad \int \tilde{p}(X'_j)dX'_j = 1$$

  giving us the transformation

  $$dX'_j = \frac{dX_j}{\sigma_j} \Rightarrow \int \tilde{p}(\frac{X_j}{\sigma_j})\frac{dX_j}{\sigma_j} = 1$$

- generalize: $u_j = f_j(x_j)$, where f_j : is an arbitrary monotonically increasing transformation, for each of the d features.

  We then get with $du_1...du_d = f'_1(x_1)dx_1....f_d(x_d)dx_d$ that

  $$\int \tilde{p}(u_1, ..., u_d)du_1...du_d = 1 = \int \tilde{p}(f_1(x_1), ..., f_d(x_d)) \underbrace{f'_1(x_1)...f'_d(x_d)}_{\text{Jacobian Factor}} dx_1...dx_d$$
  $$= \int p(x_1, ..., x_d)dx_1...dx_d$$

- data standardization identity[26]:

  $$p(x_1, ..., x_d) = \tilde{(}f_1(x_1), ..., f_d(x_d))f'_1(x_1)...f_d(x_d)'$$

9.3 Rank order standardization

consider feature j, sort instances according to features $j \to [1], ..., [N]$ sorted order

$$u_{ij} = \frac{[i]_j}{N+1}$$

	x_1	x_2	u_1	u_2
1	2	6	3/4	2/4
2	5	1	3/4	1/4
3	3	7	2/4	3/4

The lecture contains a plot of the data of the table at this point to clarify. We see that the relative positioning of the data remains the same, only the absolute position in space changes (get's squeezed together)

[26] We require the equality of the integration formula above to hold for all subsets of the Domain also and from this follows the equality

Continuous case: define cumulative distribution function CDF: $F(x_j) = Prob(x' \leq x_j) = \int_{-\infty}^{x_j} p_j(x')dx'$, With

$$u_j = F_j(x_j)$$

we get that

$$du_j = F'_j(x_j)dx_j = p_j(x_j)dx_j$$
$$\Rightarrow p(x_1, ..., x_d) = \overbrace{\underbrace{\tilde{p}(u_1 = F_1(x_1), ..., u_d = F_d(x_d))}_{\text{Copula density}}}^{\text{density tree}} \overbrace{\underbrace{p_1(x_1)....p_d(x_d)}_{\text{naive Bayes model}}}^{d \text{ 1-dim histograms}}$$

The naive Bayes model contains all of the information that is contained in the individual features, and the copula contains the interactions between the features.[27]

9.4 Training algorithm

repeat for each class $k = 1, ..., C$

1. train a 1-D histogram for each feature
2. map data to the copula using rank order transform

   $$u_{ij} = \frac{[i]_j}{N+1} \quad \text{assuming 1-based indexing (otherwise use } [i]_j + 1)$$

3. train a density tree on u_i.
4. Compute the CDF of each histogram from step 1: $F_j(x_j)$

9.5 Prediction

repeat for all classes $k = 1, ..., C$

1. compute density according to naive Bayes
2. compute $\underline{u} = (F_1(x_1), ..., F_d(x_d))^\top$
3. compute copula density for $\underline{u}$ and multiply with **1.**
4. multiply **3.** with $p(y = k)$ (i.e. the prior)

Classify $\hat{y} = k$ for k that maximizes **4.**

[27] In general one can of course use other methods for these estimations, but we will use density trees and 1-dim histograms in the exercise

9.6 Generate new instances of class k

1. traverse down the copula density tree go left with prob p, right with prob q

$$p_L = p_{\text{left child}} V_{\text{left child}} \qquad p_R = p_{\text{right child}} V_{\text{right child}}$$
$$p = \frac{p_L}{p_L + p_R} \qquad q = \frac{p_R}{p_L + p_R}$$

2. In the leaf node: sample the position uniformly at random from the leaf node region
3. Map the resulting u to x according $x_j = F_j^{-1}(u_j)$

10 Lecture 12/11

10.1 Regression

The task is to learn a function $y = f(x)$, mapping $\mathbb{R}^d \to \mathbb{R}^{d'}$ (i.e. from feature space to response space, and in our case $d' \equiv 1$). (in contrast: classification $y \in \mathbb{N}, \{0, ..., C-1\}$), from training data $\{(X_i, Y_i)\}_{i=1,...,N}$ assumed *iid*. Matrix notation[28] $X \in \mathbb{R}^{N \times d}$

We have the following order:

What is the Function class $f(x)$?

linear What is the noise?

other

simple parametric noise model (e.g. Poisson) [29]

Yes ML-estimator

Outliers

Robust regression

Outlier detection and elimination

additive Gaussian Is only Y noisy?

yes Is the linear system underdetermined? ($d > N$ or $d \leq N$ and ill conditioned)

no Do all Y_i have the same noise?

yes ordinary least squares

no weighted least squares

yes regularized / constrained least squares

no (X and Y are noisy) total least squares

[28] instances = rows

[29] somewhere in this part of the tree: transform to Gaussian noise (Anscombe transform)

non-linear What is the noise?

additive Gauss

non-linear least squares Levenberg-Marquardt-Algorithm

other

many non-linear regression methods regression forest/tree

10.2 Linear Regression

- model: $Y = \underbrace{X\beta}_{\text{lin model}} + \varepsilon$ (where ε is additive Gaussian noise); X is noise-free, Y is noisy.
- $Y_i = X_i \cdot \beta + \varepsilon_i$, $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$
- task: determine β that maximizes the likelihood of the training data (ML principle)

  $r_i := Y_i - X_i\beta$ called the residual for ith training instance

  choose β such that $r_i = \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$

$$p(\text{training data}) = \prod_{i=1}^{N} \nu \exp\left(-\frac{(Y_i - X_i\beta)^2}{2\sigma^2}\right)$$

  we are looking at the negative log likelihood:

$$-\log p(\text{training data}) = \sum_{i=1}^{N} \frac{(Y_i - X_i\beta)^2}{2\sigma^2} + N \log \nu$$

- Least Squares objective:

$$\min_{\beta} \sum_{i=1}^{N} (Y_i - X_i\beta)^2 = \sum_{i=1}^{N} r_i^2 \qquad \text{invented by Gauss}^{30}$$

- example: fit a line in 2D, model: $y = ax + b$,

  [*The lecture contains an example plot here*]

$$\min_{a,b} \sum_i (y_i - ax_i - b)^2 = Loss((x_i, y_i)|a, b)$$

$$\frac{\partial Loss}{\partial b} = -2 \sum_i (y_i - ax_i - b) = 0$$
$$\Rightarrow \quad \sum y_i - a \sum x_i = \sum b = nb$$
$$\Rightarrow \quad b = \bar{y} - a\bar{x}$$

Therefore the data centroid is alway on the regression line/hyper plane[31] $\Rightarrow$ centralize the data $y \to y - \bar{y}$, $X \to X - \bar{X} \Rightarrow b = 0$.

Alternatively: augment the matrix X: $X \to [X, 1]$

- arbitrary dimensions: matrix notation:

$$\min_\beta ||X\beta - Y||_2^2 = \min_\beta \underbrace{(X\beta - y)^\top (X\beta - y)}_{\text{Loss}}$$

$$\frac{\partial L}{\partial \beta} = 2X^\top (X\beta - y) = 0$$

Giving us the so called normal equation(s)[32]

$$X^\top X\beta = X^\top y$$

$X^\top X$ is called the scatter matrix and ($S = \frac{1}{N} X^\top X$ is the covariance, when the data are centralized). Centralization uses the magnitude of the elements of $X^\top X$ $\Rightarrow$ numerically much better

- formal solution: $\hat{\beta}(X^\top X)^{-1} X^\top y$ where $(X^\top X)^{-1} X^\top =: X^\dagger$ is known as the Moore-Penrose pseudo-inverse (generalization of inverse to rectangular matrices $X \in N \times d, N \geq d$, full rank)

- **Solution 1**: Let $R^\top R = X^\top X$ be the Cholesky factorization of $X^\top X$ where R is an upper triangular matrix .

  Theorem:

  - *Every positive definite symmetric(PDS) matrix has a Cholesky factorization*
  - $X^\top X$ is PDS, when X is full rank

$$\begin{aligned} X^\top X\beta &= X^\top y \\ R^\top R\beta &= X^\top y && \text{set } z = R\beta \\ R^\top z &= X^\top y = \tilde{y} && \text{solve for } z \text{ by forward substitution}[33] \end{aligned}$$

And finally solve $z = R\beta$ by backward substitution

[31] in higher dimensions

[32] since the solution for β is normal on it

[33] solving this then becomes very easy. In the first row you have $z_1 = \tilde{y}_i / R_{11}$, in the second $z_2 = (\tilde{y}_2 - R_{12} z_1)/R_{22}$ and so on

Brief interlude on how to compute $X^\top X$ efficiently

How to compute $X^\top X$ without ever creating X.

- naive algorithm when X exists: matrix multiplication

```
for k = 1 ... d
    for l = 1 ... d
        Q_{kl} = 0
        for i = 1 ... N
            Q_{kl} += X_{ik}*X_{il}
```

- improved algorithm: access X one row at a time

```
Q_{kl} = 0                  #for all k and l
for i = 1 ... N
    for k = 1 ... d
         for l = 1 ... d
             Q_{kl} += X_{ik}*X_{il}
```

- **Solution 2;** Singular value decomposition (SVD) of X[34]

$$X = V\Lambda U^\top$$

where Λ is diagonal and $N \times d$, V is orthogonal and $N \times N$ and U is orthogonal and $d \times d$

$$\begin{aligned} X^\top &= U\Lambda^\top V^\top \\ X^\top X &= U\Lambda^\top V^\top V\Lambda U^\top = U\Lambda^2 U^\top \\ \Rightarrow \quad (X^\top X)^{-1} &= U\left(\Lambda^2\right)^{-1} U^\top \\ \Rightarrow \beta &= (X^\top X)^{-1} X^\top y = U\Lambda^{-2} U^\top U\Lambda^\top V^\top \\ &= U\Lambda^{-1} V^\top y \end{aligned}$$

11 Lecture 19/11

11.1 Ordinary Least Squares

- $\min_\beta ||X\beta - Y||_2^2 = (X\beta - Y)^\top (X\beta - Y) = Loss(X\beta - Y)$ with the formal solution of $\beta = (X^\top X)^{-1} X^\top Y$ aka the pseudo-inverse

- Solution **1.** Cholesky: $X^\top X = R^\top R$, where R is upper triangular

[34] this is the numerically most stable algorithm, and also sometimes works when the original X is not of full rank

- Solution **2.** SVD (singular value decomposition) $X = U\Lambda V^\top$ with the solution of $\beta = V\Lambda^{-1}U^\top Y$
- Advantages of **1.**: simple, matrix $X^\top X$ $(d \times d)$ can be constructed without ever creating X $(N \times d)$
- Advantages of **2.**: numerically most stable method.

  Reason: condition umber of matrix X : $\kappa(X) = ||X||||X^\dagger|| = \frac{\max_j(|\lambda_j|)}{\min_j(|\lambda_j|)}$. Rule of thumb for the relative precision of solution of a linear system.

  $$\varepsilon = \text{ machine precision} = \begin{cases} 10^{-7} & \text{float32} \\ 10^{-16} & \text{float 64 (double)} \end{cases}$$

  rel error$(\beta) \approx \kappa\varepsilon$, e.g. if $\kappa(X) = 10^7$ then $\kappa\varepsilon = 1$ in float 32 and $= 10^{-9}$ in float 64.

  $$\kappa(X^\top X) = \frac{\lambda^2_{\max}}{\lambda^2_{\min}} = \kappa(X)^2 (= 10^{14} \text{ in the example}$$

  $\kappa(X)$ is what we deal with in the case of SVD, whereas in Cholesky we deal with $\kappa(X^\top X)$[35].
- Solution **3.**: compromise between **1.** and **2.**: QR-decomposition:[36] $X = Q \cdot R$, where Q is $N \times N$ orthogonal, and R is $N \times d$ upper triangular. We again only deal with $\kappa(X)$.

  $$\beta = (X^\top X)^{-1}X^\top Y = (R^\top Q^\top QR)^{-1}R^\top Q^\top Y = R^{-1}(R^{-T}R^\top Q^\top Y)$$
  $$= R^{-1}Q^\top Y$$

  Solve $R\beta = Q^\top Y$ (again by backward substitution)

  Algorithm sketch: Construct R one column at a time
  - initialize $R^{(0)} = X$, $Z^{(0)} = Y$
  - for $j = 1, ..., d$:
    - construct an elementary orthogonal matrix $Q^{(j)}$ via Householder transform such that $R^{(j)} = (Q^{(j)})^\top R^{(j-1)}$ makes column j of $R^{(j-1)}$ triangular (i.e. $R^{(j)}_{ij} = 0 \quad \forall i > j$).
    - $R^{(j)} = (Q^{(j)})^\top R^{(j-1)}$
    - $Z^{(j)} = (Q^{(j)})^\top Z^{(j-1)}$

[35] the downside being, that SVD is more expensive

[36] today the standard

  - solve $R^{(d)}\beta = Z^{(d)}$ via backward substitution.
- formally: $Q = Q^{(d)} \cdot Q^{(d-1)} \cdot ... \cdot Q^{(1)}$, $R = R^{(d)} (= \underbrace{(Q^{(1)} \cdot ... \cdot Q^{(d)}}_{R^{(d)}} \underbrace{(Q^{(d)})^\top \cdots \underbrace{(Q^{(1)})^\top X}_{R^{(1)}})}$

  and $Z^{(d)} = (Q^{(d)})^\top \cdots \underbrace{(Q^{(1)})^\top Y}_{Z^{(1)}}$
- Solution **4.** when matrix X is too big to be constructed, and $X^\top X$ is "too dangerous": LSQR algorithm (Paige and Saunders, 1982)
  - variant of conjugate gradient algorithm, adapted to least squares (makes it numerically more stable)
    - big matrices usually have special structure (e.g. sparse, stored on hard disk in a special format, computed on the fly,...)
    - never access X directly, but only via matrix-vector products Xu or/and $X^\top v \Rightarrow$ user has to provide efficient subroutines ("functor") for Xu and/or $X^\top v$

  Algorithm sketch:
  - for $t = 1, ..., t_{\max}$
    - construct decomposition $X = U^{(t)}B^{(t)}(V^{(t)})^\top$, $U^{(t)}$ is $N \times t$ orthogonal, $V^{(t)}$ is $t \times d$ orthogonal and $B^{(t)}$ is $t \times t$ upper bidiagonal by extending $U^{(t-1)}$, $B^{(t-1)}$, $V^{(t-1)}$ with a new row/column
    - update the RHS $Z^{(t)}$ and solution $\beta^{(t)}$ accordingly.
    - trick: never store any $U^{(t)}$, $B^{(t)}$, $V^{(t)}$ entirely. Only columns / rows $(t-1)$ and t are needed in each iteration
    - in theory: convergence after $t = d$ in practice: terminate when residual small enough or negligible improvement
    - other solutions: Gauss-Seidel iterations, gradient descent, stochastic gradient descent (active area of research, make this fast for very big data)

Example

Computed tomography[37]. *Note: There is a plot here detailing the working of tomography*

$$I_{Y_1} = I_0 \exp\left(\int_{\text{ray}} \mu(x,y)dxdy\right)$$

where I_0 is the source intensity, $\mu(x,y)$ the absorption coefficient at x, y, leading to

$$Y = -\log\frac{I}{I_0} = \int_{\text{ray}} \mu(x,y)dxdy$$

[37] will probably be the next exercise

- goal: find $\mu(x,y)$ from many directions
- discretize the problem

  [there are a lot of plots in this example]

  flatten the μ matrix $\beta_j \leftrightarrow \mu(k,l)$ $j = k + K \cdot l + 1$, $Y_i \leftrightarrow Z_{i'}^{(t)}, i = i' + tu$
- discretize integral: take $ray_{i',t}$ ray detected by $Z_{i'}$ at angle t

  $$Z_{i'}^{(t)} = \int_{\text{ray}_{i',t}} \mu(x,y)dxdy = \sum_{k,l} \text{contr of pixel (k,l) to the absorption along ray}_{i',t}$$
- model: contribution of pixel k,l is a simple function of the distance between the center of pixel and ray: e.g. triangle function $w_{k,l} = \bigwedge(\theta) = \begin{cases} 1-|\theta|, & |\theta| \le 1 \\ 0, & \text{else} \end{cases}$

  $X = (X_{ij})$ X_{ij} is the influence of pixel $j(= k + Kl + 1)$ on sensor element $i = i' + nt + 1$

  $Y_i = \int \mu(x,y)dxdy \approx \sum_j X_{ij}\beta_j$

12 Lecture 24/11

The goal of the next couple of lectures is to talk about different variations of OLS[38].

12.1 Weighted least Squares

$Y_i = X_i\beta^* + \varepsilon_i$, $\varepsilon_i \in \mathcal{N}(0, \sigma_i^2)$

- simplest case: σ_i are known (but different) $Y_i = (\hat{Y}_i + \varepsilon_i)$ e.g. by error analysis of the measurement
- Optimization problem:

  $$\min_\beta \sum_i \underbrace{\overbrace{\frac{(Y_i - X_i\beta)^2}{\sigma_i^2}}^{\text{expected value } \sigma_i^2}}_{\frac{r_i^2}{\sigma_i^2} \text{ has expected value 1}}$$
- matrix notation: $S = \text{diag}(\sigma_1^2, ..., \sigma_n^2)$ gives us the following optimization problem

  $$\min_\beta (Y - X\beta)^\top S^{-1}(Y - X\beta) = \min_\beta ||Y - X\beta||_S^2$$

  $$\Rightarrow \hat{\beta} = \underbrace{(X^\top S^{-1} X)^{-1} X^\top S^{-1} Y}_{\text{weighted pseudo-inverse}}$$

[38] which was $\min_\beta ||Y - X\beta||_2^2$

- in practice: replace $\tilde{X}_i = \frac{X_i}{\sigma_i}$ and $\tilde{Y}_i = \frac{Y_i}{\sigma_i}$ and solve

  $$\min_\beta (\tilde{Y} - \tilde{X}\beta)^2$$
- this also works if the error between different Y_i's are correlated. $\Rightarrow$ S is a spd[39] matrix. $\Rightarrow S^{-1}$ exists.
  - Cholesky factorization $S^{-1} = R^\top R$

    $$\tilde{X} = RX \qquad \tilde{Y} = RY \quad \Rightarrow \text{ OLS in } \tilde{X} \text{ and } \tilde{Y}$$
- If σ_i are unknown: we must make assumptions about the structure of S
  - most common assumption: S is diagonal
  - Iteratively Re-weighted Least-Squares(IRLS) algorithm
    - initialize $\sigma_i^{(0)} = 1$
    - for $t = 1, ..., t_{\max}$ (or until convergence)
      - solve $\beta^{(t)} = \arg\min_\beta \sum_i \frac{(Y_i - X_i\beta)^2}{(\sigma_i^{(t-1)})^2}$
      - compute residuals: $r_i^{(t)} = Y_i - X_i\beta^{(t)}$
      - set $\sigma_i^{(t)} = \sigma_i^{(t-1)} r_i^{(t)}$
  - theory: asymptotically $t_{\max} = 2$ is sufficient for $N \to \infty$
  - for finite N, more iterations are usually beneficial

12.2 Total Least Squares (TLS)

OLS: $Y_i = X_i\beta^* + \varepsilon_i$, $\hat{Y}_i = Y_i - \varepsilon_i$

TLS both Y and X are noisy

OLS: alternative (but equivalent) formulation:

$$\min_{\beta, e_i} \sum_i e_i^2, \qquad \text{s.t } Y_i + e_i = X_i\beta$$

TLS:

$$\min_{\beta, e_i, \gamma_i} ||[\gamma e]||_2^2 \quad \text{s.t. } Y_i + e_i = (X_i + \gamma_i)\beta$$

- solution:
  - $\tilde{X}, \tilde{Y}$ centralization of X and Y
  - form concatenated matrix $[\tilde{X}\ \tilde{Y}]$ $N \times (d+1)$

[39] symmetric, positive definite

- compute SVD:

$$[\tilde{X}\ \tilde{Y}] = U\Lambda V^\top$$

-

$$\min_{\beta,e,\gamma} ||[\gamma e]||2 = \min_j \lambda_j^2$$

i.e. the smalles singular value

- $\hat{\beta}$ can be constructed from the singular vectors. Let V_j be the column of V that corresponds to the smallest SV $V_j = [z\ \alpha]^\top \Rightarrow \hat{\beta} = -\frac{z}{\alpha}$
- application $2D$ line filtering with orthogonal coordinations:

$$S = [\tilde{X}\ \tilde{Y}]^\top [\tilde{X}\ \tilde{Y}] \qquad \text{combined scatter matrix}^{40}$$

$\Rightarrow$ SVD creates an approximating ellipse. SV $|\lambda_1|$ and $|\lambda_2|$ are the radii of th ellipse V_1 and V_2 ar the axis

12.3 Regularized Least Squares

- when system is underdetermined($d >> N$) or has bad condition ($\kappa(X) >> 1$) OLS doesn't work $\Rightarrow$ we need additional information
- Bias-Variance-Trade-Off
  - Define the (unknown) data distribution $(X_i, Y_i) \sim p(X, Y)$
  - Let T_N be the set of all training sets of size N

$$p(T_N) = (p(X,Y))^N \qquad \text{due to iid assumption}$$

- $\hat{\beta}$ is an estimate of β^* (the truth) from $T'_N \in T_N$ $\mathbb{E}_{T_N}[\hat{\beta}_N]$ average of $\hat{\beta}_N$ over all training sets of size N

$$\begin{aligned} MSE &= \mathbb{E}_{T_N}[(\hat{\beta}_N - \beta^*)^2] \\ &= \underbrace{\mathbb{E}_{T_N}[(\hat{\beta}_N - \mathbb{E}_{T_N}[\hat{\beta}_N])^2]}_{\text{variance of } \hat{\beta}_N} + \underbrace{\mathbb{E}_{T_N}[(\mathbb{E}[\hat{\beta}_N] - \beta^*)^2]}_{\text{bias}} \end{aligned}$$

variance "how much does $\hat{\beta}$ differ when we get several training sets?"

bias "Does our algorithm estimate the truth on average?

- application to OLS:

$$\mathbb{E}\hat{\beta} = ... = \hat{\beta}$$

[40] known in physics as the tensor of inertia)

- Since the bias is zero we get that OLS is unbiased
- variance (co-variance matrix):

$$\mathbb{E}[(\hat{\beta} - \mathbb{E}\hat{\beta})(\hat{\beta} - \mathbb{E}\hat{\beta})^\top] = \mathbb{E}[(\hat{\beta} - \beta^*)(\hat{\beta} - \beta^*)^\top]$$

We have that

$$\hat{\beta} - \beta^* = (X^\top X)^{-1} X^\top (Y - Y + \varepsilon) = (X^\top X)^{-1} X^\top \varepsilon$$

giving us

$$\begin{aligned} \mathbb{E}\left[(\hat{\beta} - \beta^*)(\hat{\beta} - \beta^*)^\top\right] &= \mathbb{E}\left[(X^\top X)^{-1} X^\top \varepsilon\varepsilon^\top X (X^\top X)^{-1}\right] \\ &= (X^\top X)^{-1}\sigma^2 \end{aligned}$$

as the covariance of the OLS estimate

- consequence: if X has bad condition $\kappa(X) >> 1$, $(X^\top X)^{-1}$ explodes $\Rightarrow$ the OLS variance becomes huge
- what happens in detail?
  - simple case: X has two almost equal columns (up to scaling factors) (=almost linearly dependent), say columns 1 and 2

    $\Rightarrow X_1\beta_1 + X_2\beta_2 \approx 0$, but $|\beta_1| >> 1$ and $|\beta_2| >> 1$ but in an independent test set, this delicate cancellation doesn't work.

    $\Rightarrow$ the huge coefficients β_1 and β_2 have catastrophic effects
  - usually, more than two columns are involved, but the cancellation effect is the same

    $\Rightarrow$ we must penalize very big coefficients. The estimate of $\hat{\beta}$ is then no longer unbiased, but a small increase in bias may cause a big decrease in variance

    $\Rightarrow$ The totale MSE error decreases (which is known as the Bias-Variance Tradeoff)[41]

12.4 Ridge Regression

IMPORTANT: STANDARDIZE DATA BEFOREHAND

$$\min_\beta ||Y - X\beta||_2^2 \qquad \text{s.t.} ||\beta||_2^2 \leq t$$

- penalization only has an effect if $t < ||\hat{|}||_2^2$, where $\hat{\beta}$ is the solution of the OLS problem

[41] the lecture contains the usual bias variance plot

- if penalization is active it forces $|\beta_1|$ to be small (shrinking) and $|\beta_{j_1}| \approx |\beta_{j_2}|$ (equalization)
- equivalent optimization problem using Lagrange multiplier $\tau > 0$:

$$\min_\beta ||Y - X\beta||_2^2 + \tau||\beta||_2^2$$

giving us $\hat{\beta} = (X^\top X + \tau \mathbb{1})^{-1} X^\top Y$

- solution 1: QR decomposition $\tilde{X} = [X\ \tau^{1/2}\mathbb{1}]^\top$, $\tilde{Y} = [Y\ 0]^\top$ $\tilde{X} = QR$

$$(\tilde{Y} - \tilde{X}\beta)^\top(\tilde{Y} - \tilde{X}\beta) = (Y - X\beta)^\top(Y - X\beta) + \tau\beta^\top\beta$$
$$\tilde{X} = QR \Rightarrow R\hat{\beta} = Q\tilde{Y} \quad \text{solve by backward substitution}$$

- solution 2: via SVD: $X = U\Lambda V^T$ *after some computations we get*

$$\hat{\beta} = V(\Lambda^2 + \tau\mathbb{1})^{-1}\Lambda U Y$$

12.5 Lasso regression ("least absolute shrinkage and selection operator")

IMPORTANT: STANDARDIZE DATA BEFOREHAND

- when X is underdetermined, features are highly redundant $\Rightarrow$ select the important features
- ridge regression shrinks coefficients to small values, but usually not to zero. $\Rightarrow$ meant to solve feature selection problem

$$\min_\beta ||Y - X\beta||_2^2 \quad \text{such that } ||\beta||_0 \leq d' < d$$

with $||\beta||_0 = \sum_j \mathbb{1}[\beta_j \neq 0]$, unfortunately, this is NP hard

- it turnes out, that L_1 regularization is almost as good

$$\min_\beta ||Y - X\beta||_2^2 \quad \text{such that } ||\beta||_1 \leq t$$

LASSO ist still a convex optimization problem $\Rightarrow$ unique solution + efficent algorithms

- Intuition: why does LASSO select variables?
  - 1-D case: Loss is a parabola [42]
  - 2-D case: *again a plot*

[42] there is a plot here

12.6 Orthogonal matching pursuit

Greedy approximation for L_0 norm.

- maintain an active set B_s of coefficients $\neq 0$
- initialize $B_s = \emptyset$, add one index per iteration, $r^{(0)} = Y$
- while $s \leq s_{\max}$:
  - find column $j_s \neq B_{s-1}$ of X that has maximum correlation with the residual $r^{(s-1)}$
  - add j_s to B_{s-1}: $B_s = B_{s-1} \cup \{j_s\}$
  - solve the OLS problem with only the columns in B_S to get $\hat{\beta}^{(s)}$
  - compute $r^{(s)} = Y - X\beta^{(s)}$

13 Lecture 26/11

- Orthogonal matching pursuit using QR decomposition
- goal: find the d' most important elements of $\beta \in \mathbb{R}^d$ ($d > d'$) and set all other elements to zero ("sparse regression")
- principle:
  - maintain an active set $B_s = \{j : \hat{\beta}^{(s)} \neq 0\}$ of non-zero indices
  - add one index to B_s per iteration, until $\#B_s = d'$
- initialize $B_s = \emptyset, r^{(0)} = Y, X^{(0)} = X$
- for $s = 1, ..., d'$
  - find $j_s \geq s$, the inde of the column $X^{(s-1)}$ with highest correlation to the residual $r^{(s-1)}$, $j_s = \arg\max_{j \geq s} \left| \left(r^{(s-1)}\right)^\top - X_j^{(s-1)} \right|$
  - swap the columns s and j_s of $X^{(s-1)} \Rightarrow X'^{(s)}$
  - perform one iteration of QR-decomposition on $X'^{(s)} \Rightarrow X^{(s)}$
    - make column $X_s'^{(s)}$ triangular using Householder transformation and update $Y^{(s)}$
    - compute $\beta^{(s)}$ (Least-squares solution for B_s) by backward substitution on $R^{(s)}\beta = Y^{(s)}$[43]
    - compute residual $r^{(s)} = Y - X\beta^{(s)}$
  - undo the column permutations on $\beta^{(d')}$ and return $\hat{\beta}$

[43] $R^{(s)}$ is the upper left $s \times s$ submatrix of $X^{(s)}$ (=triangular part)

- OMP gives an approximate solution[44] for $\min_\beta ||Y - X\beta||_2^2$ s.t. $||\beta||_0 = d'$
- LASSO problem can be solved exactly $\min_\beta ||Y - X\beta||_2^2$ s.t. $||\beta||_1 \leq t$ by Least Angle Regression (LARS), a variation of OMP
- when we express $\hat{\beta}$ as a function of t, we get the solution path $\hat{\beta}(t)$
  - $\hat{\beta}(t)$ is a piecewise linear function: it consists of "knots" $\hat{\beta}^{(s)} = \hat{\beta}(t^{(s)})$ where the path bends, and linear segments in between ($\hat{\beta}(t) = \alpha\hat{\beta}^{(s)} + (1-\alpha)\hat{\beta}^{(s+1)}$ for all $t = \alpha t^{(s)} + (1-\alpha)t^{(s+1)}$)
  - at each $t^{(s)}$ the active set B_s changes (variable enters or leaves[45])
  - init $B^{(0)} = \emptyset, X^{(0)} = X, Y^{(0)} = Y = r^{(0)}$
  - for $s = 1, ..., \underbrace{d(= \#B_s)}_{\text{OLS-Solution = end of solution path}}$
    - find $j_s'' \geq s$ that has maximum correlation with $r^{(s-1)}$
    - find $j_s' < s$ where $\beta_{j'}^{(s-1)}$ and $\hat{\beta}_{j'}^{(s)}$ and the corresponding α is biggest
    - set j_s to j_s' or j_s'', whatever event occurs first on path
      - if $j_s = j_s'$ (a variable leaves B_s), create $X'^{(s)}$ by swapping j_s and k ($= \#B_{s-1}$, aka. last active column) and remove the new k from B_s and repair triangular shape of the first $(k-1)$ columns
      - otherwise (column j_s enters B_s): swap j_s and $k+1$ and repair triangular part (first $(k+1)$ columns of X)
    - compute current solution
  - return the untangled $\hat{\beta}^{(s)}$ (after inverse permutation)
- application: topic discovery
- rows of X correspond to words ("trees", "leaf", "deer", "key"), columns of X correspond to topics ("biology", "algorithms")
- X_{ij}: probability that word i occurs in a document about topic j (exclusively)
- Y is a particular document of interest
- solve $\min_\beta ||Y - X\beta||_2^2$, such that $||\beta||_0 \leq t_{\max}$ (max number of topics allowed per document), $||\beta||_1 \leq t$
- perform non-negative OMP (i.e. only positive coefficients are allowed)
- $\hat{\beta}_J \neq 0$ means Y talks about topic j

[44] due to it's greedy nature it is not exact. This could only be achieved by checking all combinations → NP-hard

[45] the latter was not possible in OMP

- for big data: use iterative algorithms (similar to CG instead of LARS or OMP)
- variations:
  - non-negative LASSO ($\hat{\beta} \geq 0$)
  - elastic net: combination of ridge regression and LASSO $\min ||....||_2^2 + \tau_1||\beta||_2^2 + \tau_2||\beta||_1$
  - group LASSO: penalize groups of variables together
- dictionary learning: also learn the matrix X from training data ⇒ Machine Learning II

13.1 Non-linear Regression

- Task: find f s.t. $Y = f(X) \quad X \in \mathbb{R}^d, Y \in \mathbb{R}$ but f is non-linear
- given training data $\{(X_i, Y_i)\}_{i=1}^N$
- non-linear least squares

  $$\min_{f \in \mathcal{H}} ||Y - f(X)||_2^2$$

  - parametric approach: function type of f is fixed, only some free parameters to be determined: $f(X, \beta)$

    ⇒ Levenberg-Marquardt algorithm
  - non-parametric: e.g. f is piecewise constant ⇒ regression tree / forest
- solve non-linear regression by means of linear least squares:
  - map X into an augmented feature space $\tilde{X}$ (new columns from non-linear functions of the old columns)
  - kernel trick

13.1.1 Levenberg-Marquardt-Algorithm

- $f(x)$ is parametric: $f(X, \beta)$, model: $Y = f(X, \beta^*) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- linearize $f(X, \beta)$ by 1^{st} order Taylor expansion in β

$$f(X, \beta_0 + \Delta) \approx f(X, \beta_0) + \left.\frac{\partial f(X, \beta)}{\partial \beta}\right|_{\beta_0} \Delta$$

$$\min_\beta ||Y - f(X;\beta)||_2^2 \Leftrightarrow \min_\Delta ||\underbrace{Y - f(X, \beta_0)}_{\tilde{Y}} - \left.\frac{\partial f}{\partial \beta}\right|_{\beta_0} \Delta||_2^2 \text{ iff } ||\beta_0 - \hat{\beta}|| \text{ "small"}$$

$$= \min_\Delta ||\tilde{Y} - \tilde{X}\Delta||_2^2 \quad \text{OLS problem}$$

- Gauss-Newton algorithm
  - define initial guess $\beta^{(0)}$
  - repeat until convergence ($ort = 1, ..., t_{\max}$)

$$\min_{\Delta} \sum_{i=1}^{N} \left(\tilde{Y}_i^{(t-1)} - \tilde{X}_i^{(t-1)} \Delta \right)^2$$

$$\tilde{Y}_i^{(t-1)} = \tilde{Y}_i - f(X_i, \beta^{(t-1)}) \qquad \tilde{X}_i^{(t-1)} = \frac{\partial f(X, \beta)}{\partial \beta)} \Big|_{X=X_i, \beta=\beta^{(t-1)}}$$

  - solve by normal equations

$$(\tilde{X}^\top \tilde{X})\Delta = \tilde{X}^\top \tilde{Y}$$

  - $\beta^{(t)} = \beta^{(t-1)} + \Delta$

14 Lecture 01/12

14.1 parametric non-linear least-squares

- model: $Y = f(X, \beta^*) + \varepsilon$ parametric function , where β^* are the true parameters
- task: $\hat{\beta} = \arg\min_\beta \sum_i (Y_i - f(X_i, \beta))^2$
- Gauss-Newton iteration: choose initial guess $\beta^{(0)}$; update $\beta^{(t)} = \beta^{(t-1)} + (\tilde{X}^\top \tilde{X})^{-1} \tilde{X} \tilde{Y}$, $\tilde{Y} = \left[Y_i - f(X_i, \beta^{(t-1)}) \right]$, $X = \left[\frac{\partial (X_i, \beta^{(t-1)})}{\partial \beta} \right]$ (aka. the Jacobian)

  fast, but may not converge
- alternative: gradient descent:

$$\frac{\partial}{\partial \beta} (Y - f(X, \beta))^2 = -2 \underbrace{\frac{\partial f(x, \beta)}{\partial \beta}}_{\tilde{X}} \underbrace{(Y - f(x, \beta))}_{\tilde{Y}}$$

  Update: $\beta^{(t)} = \beta^{(t-1)} + \tilde{X}^\top \tilde{Y}$

  slow, but converges to the nearest local minimum
- Levenberg's idea: combine the advantages of GN and GD

$$\beta^{(t)} = \beta^{(t-1)} + (\tilde{X}^\top \tilde{X} + \tau \mathbb{1})^{-1} \tilde{X} \tilde{Y} \qquad \text{(ridge regression)}$$

- effect: if τ small: almost GN $\Rightarrow$ fast

  if τ big: $\tilde{X}^\top \tilde{X}$ doesn't matter - almost GD $\Rightarrow$ robust
- adaptive damping (τ damping parameter)

  - init: $\beta^{(0)}, \tau = \tau_0 (= 0, 1), \kappa (= \sqrt{2})$ damping update
  - iterate:
    - perform one update step
    - compute residual
    - if residual decreased:
      - accept update
      - set $\tau = \tau / \kappa$ (less damping)
    - otherwise
      - reject update
      - $\tau = \kappa \tau$ (retry with more damping)
- Marquardt's idea:
  - in normal ridge regression, we standardize X (to column-wise unit norm) beforehand to ensure fair penalization
  - doesn't work here: $\tilde{X}$ must be the Jacobian

    $\Rightarrow$ we must adjust τ for each feature using $\text{diag}(\tilde{X}^\top \tilde{X})$
- Levenberg-Marquardt-update:

$$\beta^{(t)} = \beta^{(t-1)} + (\tilde{X}^\top \tilde{X} + \text{diag}(\tilde{X}^\top \tilde{X})\tau)^{-1} \tilde{X}^\top \tilde{Y}$$

[46]

14.2 Non-parametric non-linear least-squares

- regression tree (like density tree, but for regression)

  model: piecewise constant approximation

$$\hat{f}(x) = \sum_{l=1}^{L} f_l \mathbb{1}(x \in bin_l)$$

  task: when binning is given:

$$\min_{f_l} \mathbb{E}[(f(x) - \hat{f}(x))^2] = \int (f(x) - \hat{f}(x))^2 p(x) dx = Loss$$

$$Loss = c - 2 \sum_l f_l \int_{x \in bin_l} f(x) p(x) dx + \sum_l f_l^2 \int_{x \in bin_l} p(x) dx$$

$$\frac{\partial Loss}{\partial f_l} = -2 \int_{x \in bin_l} f(x) p(x) dx + 2 f_l \int_{x \in bin_l} p(x) dx =^! 0$$

$$f_l = \frac{\int f(x) p(x) dx}{\int p(x) dx} \approx \frac{1}{N_l} \sum_{i: X_i \in bin_l} f(X_i)$$

[46] See the CERES library in C++, for more on this topic

- algorithm:
  - put all instances in a root node and place root node on a stack
  - while stack not empty:
    - take top node from stack
    - if node satisfies termination criterion:
      - compute f_l and create leaf node
    - otherwise: determine optimal split and put the two child nodes on stack
  - split selection: exhaustive search over all axes and all split points to finde the split that maximally reduces loss
    - consider one node:

$$loss(bin_l) = \sum_{i:X_i \in bin_l} (f_l - Y_i)^2$$

    - after split:

$$loss(left) = \sum_{i \in left} (f_{left} - Y_i)^2 \qquad loss(right) = ...$$

      $\Rightarrow$ choose split that minimizes $loss(left) + loss(right)$
    - termination: stop when $N_l < N_{min}$
  - important: avoid overfitting
  - traditional: pruning:
    - estimate optimism of each subtree
    - replace with a single node if too big
  - better: ensemble of regression trees
    - train T trees with randomization (each tree uses a random subset of the training data, each split optimization only considers a random subset of the possible splits)
    - return the average response of all trees
    - use a bootstrap sample for the subset of the data (explained later)

14.3 Reducing non-linear regression to linear regression

- idea: transform the data X into new variables $\tilde{X}$ where linear regression works using a non-linear mapping ("change of variables")

$$\tilde{X}_j = \phi_j(X)$$

- example: least squares fit of a circle [47]

  task:

$$\min_{c,r} \sum_{i=1}^{N} (\sqrt{(X_i - c)^2} - r)^2$$

  this non-linear problem can be solved using Levenberg-Marquardt
- change of variables:
  - define an "algebraic loss"

$$\min_{c,r} \sum_i ((x_i - l)^2 - r^2)^2 = Loss_a$$

$$(x_i - c)^2 - r^2 = x_i^2 - 2x_i c + c^2 - r^2$$

    define $\tilde{X} = [X1] \quad \tilde{Y} = ||x_i||_2^2 \quad \beta = [2c_1, 2c_2, r^2 - ||c||_2^2]$
  -

$$\sum_i ((X_i - c)^2 - r^2)^2 = \sum_i (\tilde{Y} - \tilde{X}_i \beta)^2$$

  - solve for β using OLS
  - compute solution: $c_1 = \beta_1/2$, $c_2 = \hat{\beta}_2/2$, $r = \sqrt{c_1^2 + c_2^2 + \hat{\beta}_3}$
  - better numerics after data centering.
- advantage. better and fast, robust against outliers
- disadvantage: changes error metric; doesn't work well for incomplete circles

15 Lecture 3/12

Reduce non-linear regression to linear least squares

- general idea: non-linear mapping $X \mapsto \tilde{X}$ ($\tilde{d}$ dimensional feature space, in general $\tilde{d} > d$)
- solve: $\min_\beta (\tilde{Y} - \tilde{X}\beta)^2$
- traditionally: hand-crafted feature space $\tilde{X}, \tilde{Y}$ taylored to the application by an expert
- machine learning: use generic transformations that work in many applications
- caveat: if $\tilde{d}$ is big, we may overfit $\Rightarrow$ use ridge regression

[47] the lecture contains a plot here

Dual optimization problem of ridge regression

RR: $\min_\beta \tilde{\beta}^2 + (Y - X\beta)^2 \Leftrightarrow \min_{\beta,\xi_i} \tau\beta^2 + \sum_i \xi_i^2$ such that $Y_i - X_i\beta = \xi_i \quad \forall i$

$$(Primal)\ Loss(\beta, \xi_i) = \tau\beta^2 + \sum_i \xi_i^2$$
$$Lagrangian(\beta, \xi_i, \alpha_i) = \tau\beta^2 + \sum_i \xi_i^2 + 2\tau \sum_i \alpha_i(Y_i - X_i\beta - \xi_i)$$
$$Dual\ Loss(\alpha_i) = \min_{\beta,\xi_i} Lagrangian(\beta, \xi_i, \alpha_i)$$

- important property of Dual Loss:
  - Let $\hat{\beta}, \hat{\xi}_i$ be the optimal primal solutions $Loss(\hat{\beta}, \hat{\xi}_i) = \min$
  - for all α: Lagrangian$(\hat{\beta}, \hat{\xi}_i, \alpha_i) = Loss(\hat{\beta}, \hat{\xi}_i)$
    $\Rightarrow$ Dual Loss $(\alpha_i) \leq Loss(\hat{\beta}, \hat{\xi}_i)$
  - any dual solution provides a lower bound for $Loss(\hat{\beta}, \hat{\xi}_i)$
    $\Rightarrow$ best possible lower bound: $\max_{\alpha_i} Dual\ Loss(\alpha_i) \Rightarrow \hat{\alpha}_i$
    here: $Loss(\hat{\beta}, \hat{\xi}_i) = DualLoss(\hat{\alpha}_i) \Rightarrow$ exercise

$$\frac{\partial Lagrangian}{\partial \beta} = 2\tau\beta - 2\tau \sum_i \alpha_i X_i = 0$$
$$\beta = \sum_i \alpha_i X_i$$
$$\frac{\partial Lagrangian}{\partial \xi_i} = 2\xi_i - 2\tau\alpha_i = 0$$
$$\xi_i = \tau\alpha_i$$

$$DualLoss(\alpha_i) = -\tau \sum_{i_1} \sum_{i_2} \alpha_{i_1}\alpha_{i_2} X_{i_1} X_{i_2}^\top - \tau^2 \sum_i \alpha_i^2 + 2\tau \sum_i \alpha_i X_i$$
$$DualLoss(\underline{\alpha}_i) = \tau \left[-\underline{\alpha}^\top (XX^\top + \tau\mathbb{1})\underline{\alpha} + 2\underline{\alpha}^\top Y \right]$$

- $XX^\top$: Gram matrix, kernel matrix, $K_{i_1,i_2} = X_{i_1} X_{i_2}^\top$: skalar product between instances i_1, i_2[48]
- dual solution:

$$\frac{\partial DualLoss(\alpha}{\partial \alpha} = -\tau\,(2(K + \tau\mathbb{1})\alpha + 2Y) = 0$$
$$\hat{\alpha} = (K + \tau\mathbb{1})^{-1} Y$$

[48] compare to normal equations of primal $\underbrace{X^\top X}_{\text{scatter matrix}} \beta = X^\top Y$,

- prediction for new instance X_{new}:

$$\beta = \sum_i \alpha_i X_i = X^\top \alpha$$
$$Y_{new} = X_{new}\beta = X_{new} X^\top \alpha$$
$$k = X_{new} X^\top$$ [49]
$$Y_{new} = \underbrace{Y^\top (K + \tau\mathbb{1})^{-1}}_{\text{precomputed}} k^\top$$

- compute k for each new instance and predict
- key observation: Let $\tilde{X} = \varphi(X)$,, $\tilde{K} = \tilde{X}\tilde{X}^\top$
  - for certain mappings φ, we can compute $\tilde{K}$ without ever computing $\tilde{X}$ (likewise for $\tilde{k}$)
    $\Rightarrow$ **we can solve the dual problem without computing $\tilde{X}$ "kernel trick"**
- example: $d = 2, X_i = [X_{i1}, X_{i2}]$; $\tilde{d} = 5, \tilde{X}_i = [X_{i1}^2, X_{i2}^2, \sqrt{2}X_{i1}X_{i2}, \sqrt{2}X_{i1}, \sqrt{2}X_{i2}, 1]$[50]

$$K_{i_1,i_2} = \tilde{X}_{i_1}\tilde{X}_{i_2}^\top = \ldots$$
$$= (X_{i_1} X_{i_2}^\top + \mathbb{1})^2$$

compute kernel matrix directly from the old features

Theorem(Mercer)

Let $K(X, X) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. $K(X, X)$ is a kernel function iff the kernel matrix

$$[K_{i_1 i_2} = K(X_{i_1}, X_{i_2})]$$

is positive semidefinite symmetric for all possible training sets $\{x_i\}_{i=1}^N (\forall N)$

- Kernel ridge regression:
  - choose a Mercer kernel
  - compute a kernel matrix
  - solve the dual problem $\Rightarrow \hat{\alpha}$
  - predict: compute k, return $\hat{\alpha}k^\top$
- popular kernel functions:
  - polynomial kernels : $K(X_{i_1}, X_{i_2}) = (X_{i_1} X_{i_2}^\top + 1)^p$

[49] effort: $\mathcal{O}(ND)$
[50] there follows a long computation which won't be repeated here...

  - Gaussian kernel: $K(X_{i_1}, X_{i_2}) = e^{\frac{||X_{i_1} - X_{i_2}||^2}{2h^2}}$ (with h as the bandwidth parameter) [=ε-nearest-neighbor interpolation] $\Rightarrow$ exercise]

- advantage: can handle arbitrary nonlinear problems
- disadvantages:
  - prediction may be expensive $\mathcal{O}(Nd)$
  - choose τ (and h) carefully to avoid overfitting
- simplified version: kernel regression

$$\hat{f} = \frac{\sum_i Y_i k_i}{\sum_i k_i} \qquad \left(\alpha_i = \frac{Y_i}{\sum_i k_i}\right)$$
$$= \sum_i w_i k_i \qquad \left(w_i = \frac{k_i}{\sum_i k_i} \text{ weights}\right)$$
$$= \text{ weighted average over neighboring training points}$$

"neighboring": large value of $k(X_{new}, X_i) = k_i$

- speed-up for Gaussian kernel
  - easy version: find the X_i that are near X_{new}, via quick (approximate) nearest neighbor search
  - better version: Fast Gaussian Transform: approximate the combined influence of the far neighbors by mean field approximation

Robust Regression

- robust: regression still works, when the assumptions are not exactly fulfilled
- two important violations:
  - heavier tails of the error distribution, especially outliers
  - the data contain several instances of the model (e.g. inliers of one model are the outliers of the others)
- two main approaches:
  - outlier detections & removal, followed by normal regression, followed by normal regression (dangerous, e.g. ozone hole)
  - robust regression: outliers are implicitly down-weighted by the algorithm in a controlled way

16 Lecture 8/12

RANSAC (Random sample consensus)

- very popular in image analysis
  - when inlayer fraction is small ($< 50\%$, even $< 10\%$)
  - when data contains sever al model instances
  - when the model has few free parameters, but has many local optima.
- simple algorithm: N data points, d free parameters
  - suppose, we need at least D data points to fit a model (e.g. lin 2D: $D = 2$, circle in 2d: $D = 3$, plain in $\mathbb{R}^d$: $D = d$, stitching of 2 images (fundamental matrix) $D = 8$)
  - define "inlier threshold" κ.

    X_i is inlier $\Leftrightarrow d(X_i, f) \leq \kappa$

    (for linear models. $\kappa = 3\sigma$)[51]
- algorithm:
  - repeat T times:
    - sample D points uniformly at random
    - fit $\hat{f}$ using these points
    - find inlier set $\{X_i : d(X_i, \hat{f}) \leq \kappa\}$
  - return biggest inlier set and corresponding $\hat{f}$
- optional:
  - use normal fit from all inliers
  - remove inliers from data set & repeat: find more model instances
- How big should T (number of iterations) be?
  - assumption: when we pick D inliers, we get a good model (not true, but deviations in both directions)
  - define $\gamma = \frac{N_{in}}{N}$ inlier fraction

$$p(\text{choose } D \text{ inliers}) = \gamma^D$$
$$p(\text{choose at least one outlier in } D \text{ picks}) = 1 - \gamma^D = p(\text{bad model})$$
$$p(\text{only bad models in } T \text{ iterations}) = (1 - \gamma^D)^\top$$
$$= 1 - p(\text{at least one good model in } T \text{ iterations})$$

[51] σ std deviation of true inliers (estimated in pilot experiment)

– define α: probability of RANSAC success (user defined)

$$\Rightarrow \quad (1-\gamma^D)^\top = 1-\alpha$$
$$T \geq \frac{\log(1-\alpha)}{\log(1-\gamma^D)}$$

– examples[52]:

D/r	90%	50%	30%
2	3	17	49
3	4	35	169
8	9	1172	70128

Basics of Robust Regression

- first consider the simplest model: define "center " of data (1-D)
  traditional estimator: mean $\overline{x} = \frac{1}{N}\sum_i X_i$
- define simple contaminated distribution: Gaussian mixture model[53]

$$p(X|\varepsilon,\sigma^2,\tau^2) = (1-\varepsilon)\mathcal{N}(0,\sigma^2) + \sigma\mathcal{N}(0,\tau^2), \quad \tau^2 \geq \sigma^2$$
$$var(Mean) = \frac{\sigma^2}{N}\left(1-\varepsilon+\varepsilon\frac{\tau^2}{\sigma^2}\right)$$

- traditional robust estimator: median

$$Med(x) = t \qquad s.t. \quad \#\{X_1 < t\} = \#\{X_i > t\}$$

  (not unique when N is even: $t = \frac{X_{N/2}+X_{N/2+1}}{2}$)
-

$$var(Median) = \frac{sigma^2}{N}\frac{\pi}{2(1-\varepsilon+\varepsilon\frac{\sigma}{\tau})^2}$$

- numbers: uncontaminated ($\varepsilon = 0$ or $\tau^2 = \sigma^2$)

$$\frac{var(median)}{var(mean)} = \frac{\pi}{2} = 157\%$$

  $\Rightarrow$ median of a Gaussian needs 57% mode data for same accuracy as mean
  contaminated: $\varepsilon = 5\% : \frac{v_{\text{med}}}{v_{\text{mean}}} = 1\ (\tau = 4\sigma), 0.29(tau = 10\sigma)$,
  $\varepsilon = 10\% : \frac{v_{\text{med}}}{v_{\text{mean}}} = 1\ (\tau = 3\sigma)$, $0.17(\tau = 10\sigma)$
- generalize in terms of loss functions ρ

[52] probably calculated with $\alpha = 99\%$
[53] ε is called the mixture coefficient

– estimation task: $\min_t \sum_i \rho(X_i,t) \Leftrightarrow \sum \psi(X_i,t) =^! 0$
  where t is the estimate for "outer" and $\psi(X_i,t) = \frac{\partial}{\partial t}\rho(X_i,t)$ as the "influence function"
– typical choice for ρ: ML estimator $\rho = -\log(p(X_1,...,X_N|t)$

$$p(x) \sim \mathcal{N}(0,\sigma^2) \Rightarrow \quad \rho(X_i,t) = \frac{1}{\sigma^2}(X_i-t)^2 \qquad \text{squared loss } (L_2)$$
$$p(x) \sim e^{-\frac{|x|}{\tau}} \Rightarrow \quad \rho(X_i,tI = \frac{1}{\tau}|X_i-t| \qquad \text{absolute loss } (L_1)$$

Squared loss gives us $t = \frac{1}{N}\sum_i X_i$, and L_1 gives us $t = median$

- Huber loss: combine the advantages of mean and median

$$\rho_\kappa(X_i,t) = \begin{cases}(X_i-t)^2, & \text{if } |x_i-t| \leq \kappa \\ 2\kappa|X_i-t| - \kappa^2, & \text{otherwise}\end{cases}$$
$$\psi_\kappa(X_i,t) = \begin{cases}X_i-t, & \text{if } |_i-t| \leq \kappa \\ \kappa sign(X_i-t), & \text{otherwise}\end{cases}$$

[The lecture contains a plot of Huber here]

- for Gaussian distributions: $\kappa = 1.37\sigma$

$$\frac{var(\text{Huber-center})}{var(mean)} = 1.05 \text{ for } \varepsilon = 0 \text{ (uncontaminated)}$$
$$= 0.21\ (\varepsilon = 5\%, \tau = 10\sigma)$$
$$= 0.13\ (\varepsilon = 10\%, \tau = 10\sigma)$$

- biweight loss:

$$\rho_\kappa(x,t) = \begin{cases}1-\left[1-\left(\frac{X-t}{\kappa}\right)\right]^3, & |x-t| \leq \kappa \\ 1, & \text{else}\end{cases}$$
$$\psi_\kappa(x,t) = \begin{cases}(x-t)\left(1-\left(\frac{x-t}{\kappa}\right)^2\right)^2, & |x-t| \leq \kappa \\ 0, & \text{else}\end{cases}$$

non-convex $\Rightarrow$ hard to solve

17 Lecture 10/12

application of robust loss functions in regression

- model: $Y = X\beta + \varepsilon$, estimate $\hat{\beta}$

- residual: $r_i(\beta) = Y_i - X_i\beta$
- regression problem[54]:

$$\hat{\beta} = \arg\min_{\beta} \sum_i Loss(r_i(\beta)) \Leftrightarrow \sum_i \psi(r_i(\beta)) =^! 0$$

- OLS: $Loss(r) = r^2$
- robust regression: $Loss(r) = \rho_\kappa(r)$ Huber function
  - Huber weight function

$$w_\kappa(r) = \frac{\rho_\kappa(r)}{r^2} = \begin{cases} 1, & |r| \le \kappa \\ \frac{\kappa}{|r|}, & |r| > \kappa \end{cases}$$

    $\hat{\beta} = \arg\min_\beta \sum_i w_\kappa(r_i) r_i^2$
  - iterative solution by iterated Re-weighted Least Squares
    - define initial guess $\beta^{(0)}$ and compute $r_i^{(0)}$
    - repeat until convergence, $t = 1, .., t_{\max}$
      - $\hat{\beta}^{(t)} = \arg\min_\beta \sum_i w_\kappa(r_i^{(t-1)})(Y_i - X_i\beta)^2$ by weighted least squares
- Least absolute deviation regression: $Loss(r) = |r|$
  - $w(r) \left[= \frac{Loss(r)}{r^2}\right] = \frac{1}{|r|+\varepsilon}$ gives an approximate solution in IRLS
  - exact solution: reduce LAD regression to a linear program and solve with Simplex algorithm

    introduce variables u_i to get the following problem[55]:

$$\min_{\beta, u_i} \sum_i u_i \text{ subject to } u_i \ge 0 \quad \forall i$$
$$u_i \ge Y_i - X_i\beta$$
$$u_i \ge -(Y_i - X_i\beta)$$

  - at the minimum $u : i = |Y_i - X_i\hat{\beta}| = |r_i|$
  - use standard solvers (library)
  - caveat: the solution is not unique
  - solution: solve OLS problem restricted to the set of optimal LAD solutions

[54] $\psi = \frac{\partial Loss(x)}{dx}$ is the influence function

[55] The second and third constraints are equal to $u_i \ge |Y_i - X_i\beta|$ which wouldn't be linear and is therefore not allowed.

Risk, Loss, and Optimism

or: model assessment and selection

- what is risk?
  - not the probability of undesirable events, but: the probability of undesirable events weighted by the their consequences as measured by a "loss" (loss of money, lives, opportunities, ...)
- the risk of a rule (decision, regression, prediction, ...) $\hat{Y} = \hat{f}(X)$ is the expected loss of this rule for the given loss function $L(\hat{f}(X), Y^*)$ (where Y^* represents the truth).

$$risk(f) = \mathbb{E}_{(X,Y)\sim D}[Loss\ f(X), Y)] = \int p(x,y) Loss\ f(X), Y) dxdy$$

- optimal rule $\hat{f}$ minimizes expected risk ($\hat{f}$).
- canonical example: access control (e.g. fingerprint scanner) cases
  - $Y = 1, f(X) = 1 \Rightarrow \quad TP \Rightarrow Loss = 0$
  - $Y = 0, f(X) = 0 \Rightarrow \quad TN \Rightarrow Loss = 0$
  - $Y = 1, f(X) = 0 \Rightarrow FN \Rightarrow Loss = L_{FN}$
  - $Y = 0, f(X) = 1 \Rightarrow FP \Rightarrow Loss = L_{FP}$
- Assume $p(Y|X)$ is the Bayesian posterior

$$\text{Bayesian risk: } R(\text{granting access}|X) = p(Y = 0|X) \cdot L_{FP}$$
$$R(\text{deny access}|X) = p(Y = 1|X) \cdot L_{FN}$$

  minimize Bayesian risk: grant access $\Leftrightarrow R(grant|X) < R(deny|X)$

$$\frac{R(grant|X)}{R(deny|X)} < 1 \Rightarrow \frac{p(Y=1|X)}{p(Y=0|X)} > \frac{L_{FP}}{L_{NP}} = t$$

- we recover minimum error prediction, when $Loss$= 0-1-loss: $L(\hat{f}(X), Y) = \mathbb{1}(\hat{f}(X) \neq Y)$, $L_{FP} = L_{FN} = 1$
- different definitions of Loss just shift the decision threshold $\Rightarrow$ application dependent
- expand by Bayesian formula $p(Y|X) = p(X|Y)p(Y)/(pX)$

$$\text{decide } y = 1 \Leftrightarrow \frac{p(X|Y=1)}{p(X|Y=0)} > \frac{L_{FP}}{L_{FN}} \frac{p(Y=0)}{p(Y=1)}$$

  $\Rightarrow$ the loss ratio adjusts the ratio of the priors

- The user can in principle choose an (application specific) loss after training and just change decision threshold
- but: probability estimates may be inaccurate away from the threshold where model was trained on

  $\Rightarrow$ retrain, whenever the threshold (Loss) changes
- in order to allow the user to choose the appropriate threshold, we must report the performance of our algorithm at different thresholds

  $\Rightarrow$ Receiver Operating Characteristic (ROC curve)
- Let N_0, N_1 be the number of negative/positive test instances.
  - TP, FN, FP, FN the number of the respective outcomes
  - "true positive rate" ("sensitivity", "recall"):

    $$h_{TP}(t) = \frac{TP(f)}{N_1}$$

    "false positive rate"

    $$h_{FP}(t) = \frac{FP(t)}{N_0}$$

  - ROC curve: parametric plot $h_{TP}(t)$ vs. $h_{FP}(t)$ for $t \in [0, \infty)$
    - limit cases:
      - $t = 0$ (everything classified $\hat{y} = 1$): $h_{TP}(0) = 1$, $h_{FP}(0) = 1$
      - $t = \infty$ (never classify $\hat{y} = 1$): $h_{FP}(0) = h_{FP} = 0$
      - perfect classifier $\exists \hat{t}, h_{TP}(\hat{t}) = 1, h_{FP}(\hat{t}) = 0$
      - guessing: $\forall t : h_{TP}(t) = h_{FP}(t)$ (if $N_0 = N_1$)

18 Lecture 15 / 10

18.1 How to report performance as a function of decision threshold

ROC curve: draw parametric curve

$$\begin{pmatrix} h_{FP}(t) \\ h_{TP}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

[The lecture contains the usual ROC plot here]

- use $h_{TP}(t)$ and $h_{FP}(t)$ to compute application specific risk (t)
- choose $\hat{t} = \arg\min_t risk(t) = \mathbb{E}loss(t)$.

- variation: draw $\begin{pmatrix} h_{TN}(t) \\ h_{TP}(t) \end{pmatrix}$ $\Rightarrow$ mirrored diagram (around $x = 1/2$ axis), because $h_{TN} = 1 - h_{FP}$
- AUC $= \int_0^1$ ROC-curve $dx \in [1/2, 1]$ if $N_0 = N_1$

  $\Rightarrow$ best classifier maximizes AUC (actually, the representative score is minimize $\log(1 - AUC)$)
- when we have very unbalanced problems ($N_1 << N_0$) (for example: similarity search in a database)

  Use Precision-Recall-Curve instead of ROC. Recall $= h_{TP}(t)$
  - Recall(t) $= h_{TP}(t)$ *("How many % of the relevant were found")*
  - Precision(t) $= \frac{\#TP(t)}{\#TP(t)+\#FP(t)}$ *("how many % of the answers are relevant")*

  $\Rightarrow$ draw parametric curve $\begin{pmatrix} Precision(t) \\ Recall(t) \end{pmatrix}$
- $F1$-Score: $F_1(t) = \frac{2\text{Precision(t)Recall(t)}}{\text{Prec(t)+Rec(t)}}$ harmonic mean

18.2 How to estimate optimism ? (= test error)

- test error $\mathbb{E}_{(X',Y')\sim D}[Loss(Y', \hat{f}(X')]$ (model is fixed as a function of the training set $\{(X_i, Y_i)\}_{i=1}^N$
- to make analytical computations, it is easier to average over all possible training sets (=possible models)

  $$\text{expected test error } \mathbb{E}_{TS\sim D^N}\left[\mathbb{E}_{(X',Y')\sim D}[Loss(Y', \tilde{f}(X'|TS))]\right]$$
- even easier:
  - in sample test error
  - keep X_i fixed and only draw new Y_i

  $$Err_{IN} = \frac{1}{N}\sum_i \mathbb{E}_{Y_i\sim p(Y|X_i)}\left[Loss(Y_i, \hat{f}(X_i))\right]$$

  (often X_i are deterministic (e.g. tomography)) then $Err = Err_{IN}$, otherwise Err_{IN} is an approximation of Err[56]
- expected in-sample test error

  $$\mathbb{E}_{TS}(Err_{IN}) \qquad \text{for analytical computations}$$

[56] ($Err_{IN} \to Err$ as $N \to \infty$?)

- optimism: $opt = Err - err$[57]
- expected opt: $\mathbb{E}_{TS}(opt) = \mathbb{E}_{TS}(Err) - \mathbb{E}_{TS}(err)$
- in-sample optimism: $opt_{IN} = Err_{IN} - err$
- expected in-sample optimism: $\mathbb{E}_{TS}(opt_{IN}) = ...$

be clear about which optimism you are talking about.

2 Reasons to estimate / measure optimism:

1. model assessment: report to the user how well your model works ⇒ estimate should be correct
2. model selection: among candidate models choose, the one where $err + \hat{opt}$ is minimal ⇒ ranking should be correct

3 approaches

- empirical estimation (cross-validation)
- analytical formula (Mallow's Cp criterion)
- combination of both (covariance penalty with Rademacher sampling).

18.3 Empirical estimates

- cross validation:
  - split the training data into k groups ("folds") with $\approx \frac{N}{K}$ instances each
  - for $l = 1, ..., K$
    - train on $TS \backslash Fold_l$ to get $\hat{f}_l$
    - estimate $\hat{Err}_l$ on $Fold_l$ using $\hat{f}_l$
  - return $\frac{1}{K}\sum_l \hat{Err}_l = \hat{Err}$
- popular $K \in \{2, 5, 10, N\}$ the last is called leave-one-out CV (aka LOOCV)
- advantages: easy and general (works for all models)
- disadvantages: (K repetitions may be too expensive)[58], high variance
- variants to reduce variance
  - repeat CV: repeat K-fold CV T times and return average (choose $T \cdot K \approx 50...200$)

[57] err = training error
[58] mostly a problem of the past

  - reversed CV:
    - train on $Fold_l$
    - test on $TS \backslash Fold_l$

    lower variance, but higher overfitting ⇒ penalizes complex models
  - corrected CV: estimates optimism instead of test error
    - for $l = 1, ..., K$
      - train on $TS \backslash Fold_l$ to get $\hat{f}_l$
      - $\hat{Err}_l$ test error of $Fold_l$, $\hat{err}_l$ total error of $\hat{f}_l$ on <u>all</u> data

$$= \frac{1}{K}\hat{Err}_l + \frac{K-1}{K} err_l$$
$$\hat{opt}_l = \hat{Err}_l - \hat{err}_l$$

    - compute average: $\hat{opt} = \frac{1}{K}\sum_l \hat{opt}_l$
    - train $\hat{f}$ on all data, return: $\hat{Err} = err + \hat{opt}$

- <u>Bootstrapping and the Out-of-Bag Error</u>
  - bootstrap sampling = take a second order sample;

    define new training data $S' = \{(X'_i, Y'_i)\}_{i=1}^N$, by $(X'_i, Y'_i) = \text{Select}((X_1, Y_1), ..., (X_N, Y_N))$ uniformly at random (sampling with replacement)
  - (X_i, Y_i) has certain probability to be <u>not drawn</u> in any of the N repetitions: "out-of-bag"

$$p(\text{chosen}) = \frac{1}{N}$$
$$p(\text{not chosen}) = 1 - \frac{1}{N}$$
$$p(\text{never chosen}) = \left(1 - \frac{1}{N}\right)^N$$

  - about 37% are OOB, 63% are in bag (since $1/e \approx 0.368$)
  - OOB error:
    - for $t = 1, ..., T$
      - draw S'_t and train $\hat{f}_t(X|S'_t)$
      - compute $r_{it} = Loss(Y_i, \hat{f}_t(X_i|S'_t))$
    - $\hat{r}_i = \frac{\sum_t r_{it}\mathbb{1}(i \notin S'_t)}{\sum_t \mathbb{1}(i \notin S'_t)}$
    - $\hat{Err}_{OOB} = \frac{1}{N}\sum_i \hat{r}_i$
    - OOB is essentially free for an ensemble method trained on bootstrap samples (random forest → later)
    - disadvantage: slightly biased upwards (pessimistic)

18.4 (Semi-) Analytical methods

- covariance penalty:

$$\hat{Err}_{IN} = err + \frac{2}{N}\sum_i \text{cov}_{TS}(Y_i, \hat{Y}_i)$$

$$\text{cov}_{TS}(Y_i, \hat{Y}_i) = \mathbb{E}_{Y_i \sim p(Y|X_i)}(Y_i - \mu_i)(\hat{Y}_i - \hat{\mu}_i)$$

$\mu_i = \mathbb{E}(y_i)$

- can be computed analytically for OLS, ridge regression, LASSO
- determine empirically by sampling methods (Rademacher sampling, parametric bootstrap)

19 Lecture 07/01

19.1 Unsupervised Learning

- find structure in *unlabeled* data (no expert annotations)
- rational: labeled data is *expensive* while unlabeled data is almost free

  $\Rightarrow$ usually, we have lots of unlabeled data $\Rightarrow$ make best use of it
- data mining: find interesting regularities (worth further investigation) in huge amounts of data for research & development
- improve the feature space to make some tasks simpler
  - remove uninformative features
  - to merge correlated features into one pseudofeature (noise reduction)
  - create (compute) new features to simplify decision boundary (explicitly by mapping into augmented feature space, implicitly by kernel trick)
- group instances automatically
  - clustering
  - low-dimensional embedding in a high-dimensional ambient space
  - blind signal separation

19.2 Principal Component Analysis

- simplest kind of linear feature space transform $\tilde{X} = X \cdot W$ [59]
- define W according to some application-dependent optimality criterion.

  PCA: minimize the squared approximation error between $\tilde{X}$ and X

[59] $\dim X = N \times d$, $\dim W = d \times d'$, usually $d' \leq d$

- basic version: replace X with a single feature $\tilde{v} = \tilde{X}(N \times 1)$ such that the kernel matrix changes as little as possible $XX^\top \approx \tilde{v}\tilde{v}^\top$
- important: X must be centered beforehand
- $\min_{\tilde{v}} ||XX^\top - \tilde{v}\tilde{v}^\top||_F^2$ [60] $\Leftrightarrow \min_{v,\lambda} ||XX^\top - \lambda vv^\top||_F^2$ s.t. $\tilde{v}^\top v = 1$

  [Trick: $||B||_F^2 = \text{tr}B^2$*]*

  $\Leftrightarrow \min_{v,\lambda} \text{tr}(XX^\top)^2 - 2\lambda\text{tr}(XX^\top vv^\top) + \lambda^2\text{tr}(vv^\top)^2$

$$\frac{\partial}{\partial v}(...) = -2\lambda 2XX^\top v + \lambda^2 4(v^\top v)v =^! 0$$

$\Rightarrow XX^\top v = \lambda v \Rightarrow v$ is an eigenvector of $XX^\top$ with eigenvalue λ. Which eigenvalue? Insert into Loss

$$\begin{aligned}\min_{\lambda,v} Loss(v,\lambda) &= -2\lambda\text{tr}(XX^\top vv^\top) + \lambda^2\text{tr}(vv^\top vv^\top) \\ &= -2\lambda^2\text{tr}(vv^\top) + \lambda^2\text{tr}(vv^\top) = -2\lambda^2 + \lambda^2 = -\lambda^2\end{aligned}$$

$\Rightarrow \lambda$ should be as big as possible (recall that all eigenvalues of $XX^\top$ are non-negative)

$\Rightarrow v$ is the eigenvector of $XX^\top$ with the biggest eigenvalue λ.

- how to compute the new feature $\tilde{v} = \lambda v$ without constructing $XX^\top$ (which is huge)? Especially for new data (test data)

$$\begin{aligned} XX^\top v &= \lambda v \\ X^\top X(X^\top v) &= \lambda(X^\top v) && u := X^\top v \\ X^\top X u &= \lambda u \\ XX^\top \underbrace{(Xu)}_{\sqrt{\lambda}v} &= \lambda(Xu) && (\text{if } u^\top u = 1)\end{aligned}$$

- result: best 1D feature is $\tilde{v}_i = \lambda v_i = X_i u$. with u the eigenvector of $X^\top X$ with maximal eigenvalue
- this can be replicated on the residual matrix $XX^\top - \tilde{v}\tilde{v}^\top$

  $\Rightarrow$ the second best feature is the eigenvector of $XX^\top$ with the second biggest eigenvalue
- PCA algorithm:
  - center X

[60] Frobenius norm: $||B|_F = \sum_{ij} B_{ij}^2$

- compute scatter matrix $X^\top X$ and its eigensystem $\{u_j, \lambda_j\}_{j=1}^d$, sort such that $\lambda_j \geq \lambda_{j+1}$ (alternatively, compute SVD of $X = U\Lambda V^\top$)
- choose $d' \leq d$
- compute $\tilde{X} (\in [N \times d'])$ according to $\tilde{X}_{ij} = X_i u_j$

- alternative interpretations of PCA
  - rotate the coordinate system such that the columns of $\tilde{X}$ (new features) become uncorrelated ($\text{Cov}(\tilde{X})$ is diagonal)

    $\Rightarrow$ The appropriate new coordinate axes are exactly the eigenvectors of $X^\top X$
  - intuitively we approximate the data by an ellipsoid whose axes are u_j. When $d' < d$, we keep only the axes with largest variance which are the most informative dimensions in absence of other criteria.

- whitening: scale each column to $\tilde{X}$ to unit variance (equivalently: define $\tilde{X}_{ij} = X_j \frac{u_j}{\sqrt{\lambda_j}}$)

  $\Rightarrow$ ellipse approximation becomes a sphere, all axes have "equal units"

  preprocessing for more complicated analysis

- alternative interpretation: embed $\tilde{X}$ into the space spanned by X (usually $\mathbb{R}^d$)

$$\tilde{X} = XU$$
$$\hat{X} = \tilde{X}U^\top$$

  PCA minimizes the squared reconstruction error $||X - \hat{X}||_F^2$ (since we keep the u_j with the biggest λ_j)

19.3 application: eigenvaces (face recognition)

- compute the average face and subtract (centering)
- X_i are the flattened (centered) images
- compute PCA of tX and construct $\tilde{X}$ ($d' \in [20, 100]$) $\tilde{X} = X_i U$
- classify a new face: center and project ($\tilde{X}_{\text{new}} = X_{\text{new}} U$) and use nearest-neighbor rule in $\tilde{X}$ space

20 Lecture 12 / 01

20.1 Linear Dimension Reduction

- $\tilde{X} = X \cdot \tilde{U}$ $X = N \times d$, $\tilde{X} = N \times d'$, $d' \leq d$

- fundamental algorithm: PCA

  require that the columns $\tilde{X}_j$ are uncorrelated

  $\Rightarrow \tilde{U}$ are the first d' columns of the eigenvector matrix $X^\top X = U\Lambda U^\top$, ($U$ is sorted by decreasing eigenvalues)

- prerequisite X has been centered (i.e. column mean $\frac{1}{N}\mathbb{1}^\top X = \bar{X}_c = 0$ (with $\mathbb{1} = (1, ..., 1)^\top \in \mathbb{R}^n$)

- modifications of PCA: all the usual tricks can be applied
  - change the objective of the optimization i.e. different requirements of $\tilde{X}$.
    - ICA (Independent component analysis): columns $\tilde{X}_j$ are statistically independent
    - NMF (Non-negative Matrix Factorization): require $X \geq 0$, $\tilde{X} \geq 0$, $\tilde{U} \geq 0$
    - ...
  - make the dimension reduction non-linear
    - kPCA (kernel PCA): apply the kernel trick
    - LLE (local linear embedding): piecewise linear mapping
  - if X contains outliers
    - Robust PCA

20.2 Kernel PCA

- define non-linear mapping $\phi(X)$ to a high-dimensional feature space
- kernel matrix $K = \phi(X)\phi(X)^\top$ Kernel function $k_{i_1,i_2} = K(X_{i_1}, X_{i-2})$
- apply PCA to K in the augmented feature space without explicit computation of $\phi(X)$

step 1 : center K without computing $\overline{\phi(X)}$

$$K = (\phi(X) - \mathbb{1}\overline{\phi(X)})(\phi(X) - \mathbb{1}\overline{\phi(X)})^\top$$
$$\overline{\phi(X)} = \frac{1}{N}\mathbb{1}^\top \phi(X)$$
$$\Rightarrow K' = \left(\phi(X) - \frac{1}{N}\mathbb{1}\mathbb{1}^\top\phi(X)\right)\left(\phi(X) - \frac{1}{N}\mathbb{1}\mathbb{1}^\top\phi(X)\right)^\top$$
$$= \underbrace{\left(I - \frac{1}{N}\mathbb{1}\mathbb{1}^\top\right)}_{=M}\underbrace{\phi(X)\phi(X)^\top}_{=K}\left(I - \frac{1}{N}\mathbb{1}\mathbb{1}^\top\right)$$
$$K' = MKM^\top = K - \mathbb{1}\underbrace{\left(\frac{1}{N}\mathbb{1}^\top K\right)}_{\text{col avg}} - \underbrace{\left(\frac{1}{N}K\mathbb{1}\right)}_{\text{row avg}}\mathbb{1}^\top + \mathbb{1}\underbrace{\left(\frac{1}{N^2}\mathbb{1}^\top K\mathbb{1}\right)}_{\text{avg of } K}\mathbb{1}^\top$$

$\Rightarrow$ we don't need $\phi(X)$ where M is called the centering matrix

step 2 : Compute eigendecomposition of $K' = U\Lambda U^\top$, define $\tilde{U}$ by the first d' columns of (sorted) U [may be expensive because K' is $[N \times N]$]

$\Rightarrow$ use algorithm that only computes the first d' eigenvectors of K]

$$\tilde{X} = \tilde{U}\tilde{\Lambda}^{1/2}$$

step 3: express $\tilde{X}$ in terms of the kernel matrix.

find matrix A such taht $\tilde{X} = K'A =^! \tilde{U}\tilde{\Lambda}^{1/2}$

$$(U\Lambda U^\top)A =^! U\Lambda^{1/2}$$
$$U^\top U\Lambda U^\top A = U^\top U\Lambda^{1/2}$$

$\Rightarrow A = \tilde{U}\tilde{\Lambda}^{-1/2}$

$\Rightarrow \tilde{X}_j = K'A_j$ with $A_j = \frac{\tilde{U}_j}{\sqrt{\lambda_j}}$, where $\tilde{U}_j$ is the j's eigenvector of K' and λ_j is the j's eigenvalue of K'

step 4 : mapping of new data instance $X_{\text{new}} \in [1 \times d]$

- compute the kernel vector of X_{new}

$$(k_{\text{new}})_i = K(X_{\text{new}}, X_i)$$

- center k_{new}

$$\tilde{k}'_{\text{new}} = k_{\text{new}} - \frac{1}{N}K\mathbb{1} - 1\left(\frac{1}{N}\mathbb{1}^\top k_{\text{new}}\right) + \mathbb{1}\left(\frac{1}{N}^2\mathbb{1}^\top K\mathbb{1}\right)$$

- map $\tilde{X}_{\text{new}} = k'_{\text{new}}A$

20.3 Local Linear Embedding (LLE)

- piecewise linear dimension reduction (linear dimension reduction in a local neighborhood)
- training algorithm
  - for each point $i = 1, ..., N$
    - $*$ define the neighborhood $\mathcal{N}(i) = \{i' : d(X_i, X_{i'})$is among k nearest neighbors $\}$
    - $*$ express X_i as a linear combination of its neighbors

$$\min_{w_i}(X_i - w_iX)^2$$

$$W = (w_i, ..., w_N)^\top \in \mathbb{R}^{N\times N},\ W_i = \begin{cases} w_{ii'}, & \text{for } i' \in \mathcal{N}(i) \\ 0 & \text{else} \end{cases},\ \sum_{i'} W_{ii'} = 1$$

[

- $*$ define $C^{(i)} = \sum_{i'\in\mathcal{N}(i)}(X_{i'} - X_i)^\top(X_{i'} - X_i)$
- $*$ find w'_i by solving $C^{(i)}w'_i = 1$
- $*$ normalize $w''_i = w_i/||w'_i||$
- $*$ insert w''_i into w_i at the appropriate positions

]

- given W find $\tilde{X}$ so that $\tilde{X}_i \approx W_i\tilde{X}$ (local reconstruction property is preserved) (we constructed W such that $X_i \approx W_iX$)

$$\min_{\tilde{X}} \sum_i ||\tilde{X}_i - W_i\tilde{X}||^2$$

equivalent to

$$\min_{\tilde{X}} \text{tr}((\tilde{X} - W\tilde{X})^\top(\tilde{X} - W\tilde{X})) = \text{tr}[\tilde{X}^\top \underbrace{(I-W)^\top(I-W)}_{=M}\tilde{X}]$$

$$\min_{\tilde{X}} \text{tr}\tilde{X}^\top M\tilde{X}$$

Let $M = U\Lambda U^\top$ be the eigendecomposition if M. Sort U by eigenvalues, let $\tilde{U}$ be the d' trailing eigenvectors, except the last ev, because it is trivial

$$M\mathbb{1} = 0\mathbb{1}$$

$\tilde{X} = \tilde{U}\tilde{\Lambda}^{-1/2}$

- mapping of new data X_{new}
  - find the k nearest neighbors of X_{new} in the training data
  - find w_{new} by optimizing the reconstruction error

$$w_{\text{new}} = \arg\min_w ||X_{\text{new}} - wX||_2^2 \qquad \sum_{i'} w_{\text{new}i'} = 1$$

  - map $\tilde{X}_{\text{new}} = w_{\text{new}}\tilde{X}$ (reconstruction from neighbors in $\tilde{X}$ space)

20.4 Robust PCA

- standard PCA assumes that the ignored dimensions $(d'+1, ..., d)$ in U contain only Gaussian noise
- what if this is not true?
- model: decompose X into $X = \tilde{X}\tilde{U}^\top + S$, with the first term being a low rank matrix that isn't sparse and S is assumed to be a full rank sparse matrix (instead of rank d-d' Gaussian noise)
- Candes & Li, 2011: the decomposition can be obtained exactly & "easily"

21 Lecture 12/01

21.1 Alternative objective functions in linear dimension reduction

- PCA :$\tilde{X} = X\tilde{U}$ $\tilde{U}$: $d \times d'$ orthonormal, $\tilde{X}$: $N \times d'$ with uncorrelated columns
  $\Rightarrow \tilde{U}$: first d' eigenvectors of $X^\top X$
- NMF non-negative matrix factorization:

  $$X = \tilde{X}W^\top (\Rightarrow \tilde{X} = XW^+)$$

  where W^+ is the pseudo-inverse of W.
  $X \geq 0$ given, $\tilde{X}, W \geq 0$
- application: document topic discovery (recall LASSO regression)

  X is a matrix of word counts
  X_i is a document
  X_{ij} is the count of word j in document i (d words considered)
  $(W^\top)_{i'}$ word counts of a prototypical document for topic i', d' topics
  $\tilde{X}_i$ the "loadings" of document i with respect to the topics
  $\tilde{X}_{ii'}$ how much does document i talk about topic i'

  $\Rightarrow$ negative numbers in $X, \tilde{X}, W$ would make no sense
  - construct objective function (Lee and Seung, 1999)
    assumption: X_{ij} result from a counting process
    $\Rightarrow$ noise is Poisson distributed
    $\Rightarrow$ true count = mean of the Poisson distribution $\mu_{ij} = (\tilde{X}W^\top)_{ij}$
    $\Rightarrow p(X_{ij} = c) = \frac{\mu_{ij}^c}{c!} e^{-\mu_{ij}}$
  - idea. ML approach: find $\tilde{X}$ and $W^\top$ that maximizes the log-likelihood of the observed data X.

    $$\max_{\tilde{X},W} \sum_{i,j} X_{ij} \log(\tilde{X}W^\top)_{ij} - (\tilde{X}W^\top)_{ij}$$

  - this model is non-convex, so has local optima (algorithm will not generally find global optimum)
  - the global optimum may not be unique
  - algorithm:
    * fix $d' \leq \max(N, d)$
    * form an initial guess $\tilde{X}^{(0)}, W^{(0)}$

    * alternating optimization until convergence $t = 1, ..., T$
      · for all $k = 1, ..., d'$

      $$\tilde{X}_{ik}^{(t)} = \tilde{X}_{ik}^{(t-1)} \frac{\left(\sum_{j=1}^d W_{jk}^{(t-1)} X_{ij}\right) / (\tilde{X}W^\top)_{ij}^{(t-1)}}{\sum_{j=1}^d W_{jk}^{(t-1)}}$$

      $$W_{jk} = W_{jk}^{(t-1)} \frac{\left(\sum_{i=1}^N \tilde{X}_{ik}^{(t-1)} X_{ij}\right) / (\tilde{X}W^\top)_{ij}^{(t-1)}}{\sum_{i=1}^N \tilde{X}_{ik}^{(t-1)}}$$

  - interesting: multiplicative update rule
  - converges to good local minima when the initial guess was good $\Rightarrow$ active area of research
    * classical (Lee & Seang): random uniform in $[0, 1]$
    * better: find rows of $(W^\top)^{(0)}$ (topics) by k-means clustering of the rows of X ($\Rightarrow$ next week)
    * random Acol initialization: form rows of $(W^\top)^{(0)}$ as average of randomly selected rows of X
  - mapping of new data instance X_{new}, given W
    $\Rightarrow$ construct $\tilde{X}_{\text{new}}$ by $\min_{\tilde{X}_{\text{new}}} ||X_{\text{new}} - \tilde{X}_{\text{new}}W^\top||_2^2$ by non-negative least squares or non-negative LASSO[61]
- comparison with eigenfaces
  - face recognition using PCA or NMF
  - PCA represents each face as a superposition of prototypical faces
  - NMF tends to be sparse, i.e. represents each face as a superposition of parts
- Independent Component Analysis (ICA)

  $$X_j = (\tilde{X}W^\top)_j + \varepsilon_j$$
  $$X = \tilde{X}W^\top + \varepsilon$$
  $$\Rightarrow \tilde{X} = XW^+$$
  $$W^+ = W(W^\top W)^{-1}$$

  $\Rightarrow$ goal : determine W^+, but now require that the columns of $\tilde{X}$ are statistically independent, not just uncorrelated as in PCA
- application: cocktail party problem
  - $k = 1, ..., d'$ are people talking (columns of $\tilde{X}$ are the voltage over time of a loudspeaker formalizing the mouth)

[61] ensures a sparse solution (most topics do not occur in the given X_{new}

- $j = 1, ..., d$ are microphones (columns of X are the voltages measured in each microphone over time)
- goal: recover what each person said, given the recordings of the microphones

  $X_j = (\tilde{X}W^\top)_j + \text{noise} \Rightarrow$ each microphone hears a superposition of all people talking; the rows of $W^\top$ determine the weight of each person in the observed X_j
- problem: PCA does not work
- statistical independence works better
- objective function: $p(\tilde{X}) = \prod_{k=1}^{d} p(\tilde{X}_k)$

  $\Leftrightarrow$ entropy of $\tilde{X}$ equals the sum of the entropies of the columns $\tilde{X}_k$. $H(\tilde{X}) \approx \sum_k H(\tilde{X}_k)$

  $\Leftrightarrow$ Kullback-Leibler divergence, vanishes:

$$KL(\tilde{X}_k|\tilde{X}) = \sum_k H(\tilde{X}_k) - H(\tilde{X}) \qquad \rightarrow \text{ minimize}$$

$\Leftrightarrow KL(\tilde{X}_k|X) = \sum_k H(\tilde{X}_k) - H(X) - \underbrace{\log\det W^+}_{=0}$

- since columns $\tilde{X}_k$ are independent, the covariance of $\tilde{X}$ is diagonal (uncorrelated)
- preprocess X by whitening
  - X''': given matrix
  - X'': centering of X''', that is $X'' = X''' - \mathbb{1}\overline{X}'''$
  - X': PCA of X'' (without dropping features)
  - X: X' with columns normalized to unit variance ($X_j = X'_j/\sqrt{\lambda_j}$, where λ_j is j's eigenvalue of $(X'')^\top X''$)

  $\Rightarrow$ columns of X (after whitening) are uncorrelated unit variance $\Rightarrow$ covariance of X is $\mathbb{1}$ (unit matrix)
- $\tilde{X} = XW^+$ where $\text{cov}(\tilde{X})$ is diagonal and $\text{cov}(X) = \mathbb{1} \Rightarrow W^+$ must be orthogonal
- W can only be recovered up to scaling (amplitude) $\Rightarrow$ scale W^+ arbitrarily such that is becomes orthonormal) $\det W^+ = 1$, $\log\det W^+ = 0$

$$\min_{\tilde{X}} \sum_k H(\tilde{X}_k) - H(X)$$

22 Lecture 19/01

22.1 Independent component analysis (ICA)

- model: $X = \tilde{X} \cdot W^\top + \varepsilon$, where W is an orthonormal matrix (when X was whitened beforehand) and $\tilde{X}$ has statistically independent columns, non-Gaussian. Same model as PCA.
- objective: determine $\tilde{X}$

$$\min_{\tilde{X}} \sum_j H(\tilde{X}_j) - H(X)$$

  with H being the entropy.[62]

22.1.1 Approximate Solution via FastICA (Hyrarinen and Oja

- algorithm of forward stagewise type:
  - determine W^+, the pseudoinverse of $W^\top$ ($\tilde{X} = X \cdot W^+$) one column at a time
  - in iteration t: greedily find $(W^+)_t$ conditioned on $(W^+)_1, ..., (W^+)_{t-1}$
- objective: $\sum_k H(\tilde{X}_k)$ (since H(X) is independent of solution)
- among all distributions with given (fixed) variance, the Gaussian has the highest entropy.

  $\Rightarrow$ neg-entropy $J(\tilde{X}_j) = H(Z_k) - H(\tilde{X}_k) \rightarrow$ maximize[63]
- approximate neg-entropy by expectations

$$J(\tilde{X}_j) \approx (\mathbb{E}[G(Z_k)] - \mathbb{E}[G(\tilde{X}_k)])^2$$

  G is a non-quadratic loss function, e.g. $G(r) = \log(\cosh(r))$ (similar to Huber function); $G(r) = -\exp(-r^2/2)$ (similar to bigweight function -1)
- FastICA algo:
  - for $k = 1, ..., d'$ ($d' \leq d$ feature space dimensions)
    - init $(W^+)_k^{(0)}$ randomly
    - repeat until convergence ($t = 1, ..., T$)

[62] PCA: $\min_{\tilde{X}, W^\top} ||X - \tilde{X}W^\top||_F^2$

[63] with Z_k being an auxiliary Gaussian variable with $\text{var}(Z_k) = \text{var}(\tilde{X}_k)$

· optimize the k's column of $\tilde{W}^+$ via Newton iterations

$$v = \frac{1}{N}\sum_i X_i^\top G'(X_i(W^+)_k^{(t-1)}) - \frac{1}{N}\sum_i (W_k^+)^{-1} G''(X_i(W^+)_k^{(t-1)})$$

$$v' = v - \sum_{l=1}^{k-1}(V(W^+)_l)(W^*)_l \quad \text{orthogonalization}$$

$$(W^+)_k^{(t)} = \frac{v'}{||v'||} \quad \text{normalization}$$

· return $W^+ = [(W^+)_1, ..., (W^+)_{d'}]$, compute $\tilde{X} = XW^+$

22.2 Clustering

- dimension reduction reduces the number of columns (features) of X
- clustering reduces the number of rows (instances)
  - group related / similar instances into clusters
  - replace each cluster by a representative
  - notation: C number of clusters $C \leq N$, k: cluster index, $k \in \{1, ..., C\}$; C_k: kth cluster; N_k: size of cluster k; $\sum_k N_k = N$
  - typical representatives:
    * mean $\overline{X}_k = \frac{1}{N_k}\sum_{X_i \in C_k} X_i$
    * medoid: $\overline{X}_k = \arg\min_{X_i \in C_k} \sum_{i' \in C_k} \delta(X_i, X_{i'})$, where δ is some arbitrary metric

      (similar to the median, but restricted to existing instances)
    * higher computational complexity: $O(N_k^2)$ vs mean: $O(N_k)$
  - Hierarchical clustering
    * start with considering each instance a cluster of its own
    * repeat until only a single cluster remains:
      · merge the pair of closest clusters ($\delta(C_k, C_{k'})$ is minimal
      · update the cluster distances between the new and all remaining clusters
  - result: "dendrogram"
    * define a threshold on cluster distance or desired number of clusters C where merging stops
    * good threshold: big gap between merge distances
    * the result critically depends on the cluster distance definition

      · single linkage: nearest neighbors between clusters
      · complete linkage: farthest neighbors
      · average linkage: mean distance
      · representative linkage: $\delta(\overline{X}_k, \overline{X}_{k'})$
    * computation complexity differs
      · single linkage $\Leftrightarrow$ minimum spanning tree

        algorithm of Kruskal $\mathcal{O}(E \log E)$, $E = \mathcal{O}(N^2)$
      · the others require $\mathcal{O}(N_k N_{k'})$ to update the cluster distance
    * effect of different choices:
      · when the data cluster well, all criteria generate similar clusters
      · single linkage may suffer from "chaining"
      · complete linkage may violate the "closeness property" (distances within a cluster may be bigger than distances between clusters)
      · average linkage & representative linkage provide a compromise.

23 Lecture 21 / 01

23.1 Clustering: k-means and EM algorithm

- given: X, distance $\delta(X_i, X_{i'})$
- total distance. $D = \frac{1}{2}\sum_{i,i'} \delta(X_i, X_{i'})$
- any clustering partitions this into the within-cluster distance D_{in} and between-cluster distance D_{betw}

  $$D = D_{\text{in}} + D_{\text{betw}}$$

  $$D_{in} = \frac{1}{2}\sum_{k=1}^{C}\sum_{i,i' \in C_k} \delta(X_i, X_{i'})$$

  and D_{betw} analogously
- a good clustering should minimize D_{in} and maximize D_{betw} $\Leftrightarrow$ similar instances (small distance) are clustered together
- ideally, we would optimize over all possible clusterings, but this is intractable or trivial if $C = N$

- try a number of fixed C and choose the "best" $\Rightarrow$ later
- consider C fixed (hyperparameter), but this is still intractable
- k-means is a greedy approximation algorithm ($k = C$) for the special case $\delta(X_i, X_{i'}) = ||X_i - X_{i'}||_2^2 \Rightarrow$ easy to show $D_{\text{in}} = \sum_{k=1}^{C} N_k \sum_{i \in C_k} ||X_i - \overline{X}_k||^2$
  - when cluster membership $i \in C_k$ is fixed, D_{in} is a minimum when $\overline{X}_k = \frac{1}{N_k} \sum_{i \in C_k} X_i$
  - when the representatives, D_{in} is minimized by assigning each instance to the nearest representative
- k-means algorithm: alternating optimization
  - fix C
  - select initial guess $\overline{X}_k^{(0)}$ for $k = 1, ..., C$
  - repeat until convergence ($t = 1, ..., T$)
    - optimize cluster membership
    - update representatives
- this converges to a local optimum, sometimes a bad one

  $\Rightarrow$ need a good initial guess
  - standard guess: choose $\overline{X}_k^{(0)}$ at random

    $\Rightarrow$ repeat optimization with several initial guesses and take the best optimum (minimal D_{in})
  - k-means++:
    - choose $\overline{X}_1^{(0)}$ at random
    - repeat for $k = 2, ..., C$
      - compute for each instance i the distance to the nearest representative already selected
      - define a pseudo-probability $\delta = \sum_i \delta_i$, $p_i = \frac{\delta_i}{\delta}$
      - select $\overline{X}_k^{(0)}$ at random such that X_i is selected with probability p_i.
- new data X_{new} are assigned to the nearest cluster
- variant: k-medoid: always choose $\overline{X}_k^{(t)}$ from X_i
  - works for arbitrary distances $\delta(\cdot, \cdot)$, but is $\mathcal{O}(N_k^2)$ instead of $\mathcal{O}(N_k)$
- how to choose the number of clusters C?
  - in general an unsolved problem, but many heuristics exist
  - simplest $C = \sqrt{N/2}$ (when the exact number is not important)

  - alternative: exhaustive search: set $C = C_{\min}, ..., C_{\max}$ and choose the "best", for example
    - cross-validation: compute average $\frac{D_{\text{in}}}{D}$ on test data
    - elbow[64] method: draw $\frac{D_{\text{in}}}{D}$ vs. C
- application: fit lines (alternative to RANSAC)

  [The lecture contains some plots here]

23.2 Expectation-Maximization Algorithm

- k-means produces hard cluster memberships (each X_i belongs to exactly one cluster)
- how about soft membership? $p(Y_i = k | X_i, \Theta) \in [0, 1]$
- mixture model with C components $p(X_i|\theta) = \sum_{k=1}^{C} \pi_k P_k(X_i|\beta_k)$

  π_k mixture coefficient, i.e. probability that X_i comes from component k $\sum_k \pi_k = 1$

  $P_k(X_i|\beta_k)$ distribution of the component (cluster) k
- usually, all components have the same distribution type, mostly Gaussian
- theorem: with sufficiently big C, any distribution can be represented as a Gaussian mixture to any desired precision.
- optimize this according to maximum likelihood method

  likelihood $p(x) = \prod_i p(X_i|\theta)$, log-likelihood $L(x) = \sum_i \log p(X_i|\theta)$

$$\frac{\partial L}{\partial \beta_k} = ... = \sum_i \underbrace{p(Y_i = k | X_i, \theta)}_{\text{soft cluster membership}} \frac{\partial}{\partial \beta_k} \log p(X_i|\beta_k) =^! 0$$

- if class membership were hard and known

$$p(Y_i = k' | X_i, \theta) = \delta_{kk'} \qquad \text{for some } k$$

$$\Rightarrow \sum_{i:Y_i=k} \frac{\partial}{\partial \beta_k} p(X_i|\beta_k) =^! 0$$

  standard log-likelihood for each cluster separately

[64]
- point nearest to the origin
- point of maximum curvature

an obvious elbow does not always exist

- if a soft clustering were known

$$p(Y_i = k|X_i, \theta) = \gamma_{ik} \qquad \sum_k \gamma_{ik} = 1 \quad \forall i$$

$$\sum_i \gamma_{ik} \frac{\partial}{\partial \beta_k} \log p(X_i|\beta_k) =^! 0$$

$\Rightarrow$ weighted log-likelihood

if $p(X_i|\beta) = \mathcal{N}(X_i|\mu_k, s_k) \Rightarrow$ weighted least squares problem

24 Lecture 26 / 01

24.1 Clustering with EM algorithm

- soft cluster assignment $\gamma_{ik} = p(Y_i = k|X_i, \theta)$ prob. that X_i belongs to cluster $k \in 1, ..., C$
- model[65]: $p(X_i|\theta) = p(X_i|Y_i, \theta)p(Y_i|\theta)$

  [whereas the LHS is difficult to learn, the hope is that the RHS is easier to learn, but: cluster membership unknown; but we can approximately determine them]

  $\Rightarrow$ alternating optimization: learn Y_i keeping θ fixed and learn θ keeping Y_i fixed.[66]
- mixture model:

$$p(X_i|\theta) = \sum_k \pi_k p(X_i|\beta_k)$$

  with $\sum_k \pi_k = 1, \pi_k > 0 \ \ \forall k$ for the mixture coefficients.
- gaussian mixture model: $p_k(X_i|\beta_k) = \mathcal{N}(X_i|\mu_k, S_k)$
- Data log likelihood: $L(\theta) = \sum_{i=1}^N \log p(X_i|\theta) = \sum_i \log(\sum_k \pi_k p_k(X_i|\beta_k)) \rightarrow \max$
-

$$\frac{\partial L}{\partial \beta_k} = \sum_i \gamma_{ik} \frac{\partial}{\partial \beta_k} \log p_k(X_i|\beta_k) =^! 0$$

  $\Rightarrow$ with γ_{ik} fixed, this is a weighted ML problem for the kth mixture component

  $\Rightarrow$ alternating optimization: Solve $\sum_i \gamma_{ik}(\theta^{(t-1)}) \frac{\partial}{\partial \beta_k} \log p_k(X_i|\beta_k)$ to get $\beta_k^{(t)}$

[65] Y_i is a "latent" or "hidden" variable, which denote cluster membership

[66] this is just an approximation, the exact solution requires a joint optimization of Y_i, θ

-

$$\frac{\partial(L + \lambda(\sum_k \pi_k - 1))}{\partial \pi_k} =^! 0 \qquad \Rightarrow \pi_k^{(t)} = \frac{1}{N} \sum_i \gamma_{ik}(\theta^{(t-1)})$$

- algorithm (for Gaussian mixture models)
  - choose number of mixture components C (difficult! see k-means)
  - define initial guess $\theta^{(0)}$ (random, k-means++ initialization)
  - repeat until convergence $t = 1, ..., T$
    - compute responsibilities γ_{ik}

$$\gamma_{ik} = p(Y_i = k|X_i, \theta^{(t-1)}) = \frac{\pi_k^{(t-1)} \mathcal{N}\left(X_i|\mu_k^{(t-1)}, s_k^{(t-1)}\right)}{\sum_{k'} \pi^{(t-1)} \mathcal{N}\left(X_i|\mu_{k'}^{(t-1)}, s_k^{(t-1)}\right)}$$

    - E-step (expectation step)

$$\pi_k^{(t)} = \frac{1}{N} \sum_i \gamma_{ik}$$

    - M-step (maximization step): determine the weighted ML solution for each mixture component individually

$$\mu_k^{(t)} = \frac{\sum_i \gamma_{ik} X_i}{\sum_i \gamma_{ik}}, \quad s_k^{(t)} = \frac{\sum_i \gamma_{ik}(X_i - \mu_k^{(t)})^\top (X_i - \mu_k^{(t)})}{\sum_i \gamma_{ik}}$$

24.2 State-of-the-art Classification Methods

- Support Vector Machines (later: ensemble methods (random forest, boosting))
  - consider linearly separable data (linear classifier has zero training error); two classes $Y_i \in \{-1, 1\}$

    $\Rightarrow$ generate infinitely many decision planes with zero training error, which one is the best? $\Rightarrow$ "margin maximization"
  - intuitive explanation:
    - assume that test data are similar to training data; test points are within distance r from training points with high probability
    - to minimize test error, we should also correctly classify the r neighborhood.

      $\Rightarrow$ a lot fewer decision planes
    - margin maximization means choose r as big as possible, s.t. only one plane remains

* decision rule = $\text{sign}(\underbrace{X\beta + \beta_0}_{\text{plane eq.}})$

  margin maximisation $Y_i(X_i\beta + \beta_0) = M_i \rightarrow$ maximize

  define $M = \min_i M_i$(most critical instances) ≥ 0 (because data is lin separable)

* value of M is arbitrary because $Y_i(X(\tau\beta) + \tau\beta_0) = \tau M_i$ is the same condition $\Rightarrow M = 1$.
* distance of a point from a plane: $Y_i\left(X_i\frac{\beta}{||\beta||} + \frac{\beta_0}{||\beta||}\right) = \text{margin} \geq \frac{1}{||\beta||}$

  $\Rightarrow$ max margin $\Leftrightarrow$ min $||\beta||$ $\Leftrightarrow$ min $\frac{1}{2}||\beta||^2$

- SVM optimization problem — separable case

$$\min_{\beta,\beta_0} \tfrac{1}{2}||\beta||^2 \quad \text{s.t. } \forall i : Y_i(X_i\beta + \beta_0) \geq 1$$

  – margin maximization theoretical explanation with "structured risk minimization") (Vapnik-Chervonenkis-theory VC-theory)

    * we want to minimize optimism = test error - training error)
    * we know: optimism decreases when we have more training data; increases when the model is more powerful (classifier complexity is high)
    * VC dimension: N_{VC} is a clever way to measure model complexity (closely related to Rademacher sampling)
    * a classifier has VC dimension N_{VC} if
      · exists a training set of size N_{VC} where the classifier achieves zero training error for all possible labelings. (even non-sensical ones)
      · no such training set exists with size $N_{\text{VC}} + 1$
    * canonical example: linear classifier in d-dimensional feature space: $N_{\text{VC}} = d + 1$
    * VC theorem: optimism $\leq O\left(\sqrt{\frac{N_{\text{VC}}}{N}}\right)$
    * for regularized linear classifier, it holds: $N_{\text{VC}} \propto ||\beta||^2 \rightarrow$ minimizing $||\beta||^2$ minimizes overfitting for sufficiently small $||\beta||^2$

- SVM optimization problem – non-separable case: two objectives:
  1. maximize margin of correctly classified instances
  2. minimize loss for wrongly classified instances

  – trick: introduce a slack variable $\xi_i \geq 0$ for each instance and change the constraint into $Y_i(X_i\beta + \beta_0) \geq 1 - \xi_i$

  – new minimization problem:

$$\min_{\beta,\beta_0,\xi_i} \tfrac{1}{2}||\beta||^2 + C\sum_i \xi_i \qquad \text{s.t. } \xi_i \geq 0, Y_i(X_i\beta + \beta_0) \geq 1 - \xi_i$$

  $C \geq 0$: hyper parameter to balance the two goals;

  ξ_i can be interpreted as a loss function:

$$\xi(X_i, Y_i|\beta, \beta_0) = \max\left(0, 1 - Y_i(X_i\beta + \beta_0)\right)$$

  (this is called the hinge loss)

  SVM-like classifiers can be defined for other loss functions as well:

  * squared hinge loss: $\max(0, 1 - Y_i(X_i\beta + \beta_0))^2$
  * logistic loss[67]: $\log(1 + e^{-Y_i(X_i\beta+\beta_0)})$

  – these losses are convex surrogats of the non-convex and non-smooth $0-1$ loss.

  $\Rightarrow$ easier optimization (unique solution, gradients,...)

25 Lecture 28 / 01

25.1 Support Vector Machines (SVM)

- training data $\{(X_i, Y_i)\}_{i=1}^N$, $Y_i \in \{-1, 1\}$
- prediction $\hat{Y}_{\text{new}} = \text{sign}(X_{\text{new}}\beta + \beta_0)$
- training

$$\min_{\beta,\beta_0,\xi_i} \tfrac{1}{2}\beta^\top\beta + C\sum \xi_i$$

- dual problem:

$$Lagrangian(\beta, \beta_0, \xi_i, \alpha_i, \lambda_i) = \tfrac{1}{2}\beta^\top\beta + C\sum_i \xi_i - \sum_i \alpha_i\left[Y_i(X_i\beta + \beta_0) - (1 - \xi_i)\right] - \sum_i \lambda_i\xi_i$$

$$\alpha_i, \lambda_i \geq 0$$

$$\frac{\partial Lagrangian}{\partial\beta} = \beta - \sum_i \alpha_i Y_i X_i^\top =^! 0$$

$$\Rightarrow \beta = \sum_i \alpha_i Y_i X_i^\top$$

[67] this is smooth in contrast to the other two

compute primal solution from the dual

$$\frac{\partial Lagrangian}{\partial \beta_0} = -\sum_i \alpha_i Y_i =^! 0 \Rightarrow \text{constraint on the } \alpha_i$$

$$\frac{\partial Lagrangian}{\partial \xi_i} = C - \alpha_i - \lambda_i =^! 0 \Rightarrow \alpha_i = C - \lambda_i \Rightarrow \alpha_i \leq C$$

$\Rightarrow$ constraint on α_i : $\quad 0 \leq \alpha_i \leq C$

- eliminate β, β_0, ξ_i from the Lagrangian: dual SVM problem

$$\max_{\alpha_i} \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,i'} \alpha_i \alpha_{i'} Y_i Y_{i'} X_i X_{i'}^\top$$
$$s.t. \sum \alpha_i Y_i = 0, 0 \leq \alpha_i \leq C$$

- Karush-Kuhn-Tucker (KKT) conditions describe necessary conditions for any solution of an optimization problem

$$\frac{\partial L_0}{\partial \alpha_i} = 0 \quad \forall i : \alpha_i [Y_i (X_i \beta + \beta_0) - (1 - \xi_i)] = 0$$

[For the rest of the computations see the literature..., the script will only contain a very abbreviated version of the remaining lectures]

- case 1: X_i is on the correct side of the margin $\Rightarrow$ no slack needed $\Rightarrow \alpha_i = 0$
- case 2: X_i is on the margin $\Rightarrow 0 < \alpha_i < C$
- case 3:X_i is on the wrong side of the margin plane $\Rightarrow \alpha_i = C$
- all those X_i with $\alpha_i = 0$ have no influence on β, the others are called the support vectors
- SVM solution depends on the critical points near (or beyond) the decision plane. In contrast, the solution of QDA and LDA depend on the mean, a typical point

25.2 SVM algorithms

- LIBLINEAR contains a collection of fast algorithms for linear SVM and its variants (alternative loss functions & regularization terms)
- Model $\hat{Y} = \text{sign}(X\beta + \beta_0)$
- standard algorithm for classical linear SVM (hinge loss, quadratic regularize) "dual coordinate descent" (Hsieh et al. 2008)
- [merge the intercept β_0 into β by adding an auxiliary constant feature]

  - initial guess $\alpha_i^{(0)}$ (zero or random, $0 \leq \alpha_i \leq C$)
  - compute $\beta^{(0)} = \sum \alpha_i^{(0)} Y_i X_i$
  - ...
- popular kernels:
  - polynomial $k(X_i, X_j) = (X_i X_j^\top + 1)^\tau$
  - Gaussian $K(X_i, X_j) = \exp\left(\frac{-(X_i - X_j)^2}{2\tau^2}\right)$
- prediction: $\hat{Y}_{\text{new}} = \text{sign}\left(\sum_i \alpha_i Y_i K(X_i, X_{\text{new}})\right)$

if K is Gaussian: "soft nearest neighbor rate" because $k \approx 0$ if X_{new} is far from X_i

- LIBSVM contains fast algorithms for kernel-SVM (also works for linear SVM, but are slower than LIBLINEAR)
- Sequential Minimal Optimization (SMO) Observation: SVM optimization can be done analytically if there are only two instances.
- SMO algorithm:

  [...]
- how to recognize convergence? KKT conditions must be fulfilled

$$Y_i \sum_j \alpha_j Y_j K(X_i, X_j) \begin{cases} > 1, & if\ \alpha_i = 0 \\ = 1, & if\ 0 < \alpha_i < C \\ < 1, & if\ \alpha_1 = C \end{cases}$$

- Multiclass problems: we train #classes 1-against-rest SVMs
- advantages of SVM:small error; reasonably fast (especially linear SVM)
- disadvantages: slow when many support vectors & features and need to adjust the hyperparameters C using cross-validation $\rightarrow$ some expert knowledge needed in training

26 Lecture 02/02

noisy XOR problem

- the best linear decision has 25% error [68]

[68] assuming equal class sizes

- kernel - SVM with order-2 polynomial kernel uses

$$K(X_i, X_j) = \tilde{X}_i \tilde{X}_j^\top = (X_i, X_j + 1)^2$$
$$\tilde{X}_i = \phi(X_i) = (X_{i1}^2, X_{i2}^2, \sqrt{2}X_{i1}, \sqrt{2}X_{i2}, \sqrt{2}X_{i1}X_{i2}, 1) = (\tilde{X}_{i1}, ..., \tilde{X}_{i6})$$

- feature $\tilde{X}_{i5}$ is the interesting one:

$$\tilde{X}_{i5} \begin{cases} > 0 & \text{if } Y_i = 0 \\ < 0 & \text{if } Y_i = 1 \end{cases}$$

  $\Rightarrow$ simple threshold ($\hat{=}$ linear classifier on $\tilde{X}$) achieves perfect classification $\Rightarrow$ kernel-SVM works

  (but: feature learning $\hat{=}$ explicitly computing the relevant subset of $\tilde{X}$ would be even better $\Rightarrow$ ML 2 next semester)

26.1 Ensemble Methods for Classification

- idea: create a strong classifier by cleverly combining many weak ones
- two main possibilities:
  - independent weak classifiers $\rightarrow$ take the average, example Random Forest
  - train classifier t conditional on class $1, ..., (t-1)$

    $\Rightarrow$ classifier t tries to correct the mistakes of $1, ..., (t-1)$ $\Rightarrow$ boosting

26.2 Random Forests

- why does the average of weak independent classifiers work?
- suppose each weak classifier has error rate $q < 0.5$ (if 2 classes)

  $\Rightarrow$ success rate $p = 1 - q$
- probability that t classifiers among T are correct is a binomial distribution

$$p(\#\text{ correct} = t) = \binom{T}{t} p^\top (1-p)^{T-t}$$

  $\mathbb{E}(\#\text{ correct}) = Tp$, $\text{Var}(\#correct) = Tp(1-p)$, $\mathbb{E}\left(\frac{\#\text{ correct}}{T}\right) = p > 0.5$ and $\text{Var}\left(\frac{\#\text{ correct}}{T}\right) = \frac{p(1-p)}{T} \rightarrow 0$
- example: $q = 1/3 \Rightarrow$ std(average of T = 100) ≈ 0047
- $\mathbb{E}$(average $T = 100$) $-$ 3std(average $T = 100$) ≈ 0.52. This is called the 3σ bound (aka 99.7% of the results are within 3σ) $\rightarrow$ ensemble error rate is 0.15%

- but: in practice the individual classifiers are *not* independent, which means that: $\mathbb{E}(\text{avg}) = p$ but $\text{var}(\sum_t \kappa_t) = \sum_{s,t} cov(\kappa_s, \kappa_t)$

  $\Rightarrow$ var(avg) $= var\left(\frac{\sum_t \kappa_t}{T}\right) = \frac{1-\bar{\rho}}{T}\sigma^2 + \frac{T-1}{T}\bar{\rho}\sigma^2$ where $var(\kappa_t) = \sigma^2$ and average correlation $\bar{\rho} = \frac{2}{T(T-1)} \sum_s \sum_{t>s} \frac{cov(\kappa_s, \kappa_t)}{\sigma^2}$
  - uncorrelated $\kappa_t \Rightarrow \bar{\rho} = 0 \Rightarrow var(avg) = \frac{\sigma^2}{T} \rightarrow 0$
  - when $\bar{\rho} > 0$, the second term $\frac{T-1}{T}\bar{\rho}\sigma^2 \rightarrow \bar{\sigma}^2 \neq 0$ as $T \rightarrow \infty$

    $\Rightarrow$ it makes no sense to increase T farther when the first term is already less than 1/10 of the second[69]

    $\Rightarrow$ use $T \approx 10\frac{1-\bar{\rho}}{\bar{\rho}}$
  - interesting trade off:
    - make individual classifiers more independent $\Rightarrow$ individual error rate goes up
    - make individuals better $\Rightarrow$ correlation increases, averaging has less effect
- Random Forest empirically achieves a very good compromise
- $\bar{\rho} \approx 0.05 \Rightarrow T \approx 200$
- individual classifiers are decision trees
- training of a single decision tree: essentially equal to density trees
  - put all instances in a single root node
  - place rot node on a stack
  - while stack is not empty
    - pop top node from stack
    - if termination condition fulfilled
      - turn current node into leaf node
    - else
      - find the best split & turn node into split node
      - put instances into child nodes & child nodes on stack
- termination condition:
  1. node is pure (all instances have same class) $\Rightarrow$ zero training error
  2. maximum predefined tree depth reached
  3. splitting would result in a child with too few instances
- Leaf node output (classes $k = 1, ..., C$)

[69] arbitrary term

- hard decisions (vote for majority class)
- soft decision (leaf returns class probabilities)

- split selection: goal: minimize training error $\hat{=}$ leafs should be as pure as possible, but: to avoid overfitting, this should be achieved with as few splits as possible

  $\Rightarrow$ optimize purity after every split

- two popular purity measures:
  - entropy $= -\sum_k \hat{p}_k \log \hat{p}_k \ \left(\hat{p}_k = \frac{N_k}{N}\right)$
  - Gini impurity $= 1 - \sum_k \hat{p}_k^2$
  - perfect purity $\hat{p}_k = 1$ for some k, $\hat{p}_j = 0$ for $j \neq k \Rightarrow H, G = 0$
  - perfect mixture $\hat{p}_k = \hat{p}_j$ for all $j, k \Rightarrow \hat{p}_k = \frac{1}{C}$ and $H = \log C$, $G = \frac{C-1}{C}$ [70]

- try all possible splits on all features (sort instance be each feature in turn and use all gaps as candidate split points)
  - for each split we get N_{Lk} instances of class k in the left child

  ...
  - C4.5 uses the entropy, while CART uses G

- single decision trees overfit strongly
  - traditional solution: pruning $\hat{=}$ replace subtrees that likely represent only overfitting by a single leaf
  - Breimann 1998: taking the average over many independent trees is much better (no pruning equired, although sometimes beneficial)
  - how to make the trees nearly independent?
    1. train each tree on a random subset of the training data: draw a bootstrap sample of size N (instance is out-of-bag with probability $\frac{1}{e} \approx 36.8\%$)
    2. select the split only over a subset of the features in each node, draw d_{try} features at random (usually $d_{\text{try}} = \sqrt{d}$). d_{try} must be higher when there are many uninformative features

27 Lecture 04 / 02

27.1 Ensemble Classifiers

- train uncorrelated classifiers and take the average of their predictions (Random Forest)
- conditional training: train classifier t so that it corrects the errors of classifiers $1, ..., (t-1)$

[70] these are both the maximum values respectively

  - boosting: increase the weights of difficult instances and minimize weighted loss
  - gradient boosting: classifier t learns a gradient step from the current loss (of ensemble $(t-1)$) to the optimal solution
  - cascaded classification: make easy decisions quickly and cheaply, use more complex methods for difficult cases $\Rightarrow$ early stopping in the ensemble

27.2 AdaBoost (Freund & Shapire, 1997)

- train $t = 1, ..., T$ classifiers sequentially, $g(X|\theta_t)$[71] model after step t:

$$f_t(X) = f_{t-1}(X) + \beta_t g(X|\theta_t)$$

- prediction: $\hat{y} = \text{sign}(f_T(X))$
- theory (Friedman et al. 2000): in each step, we minimize an exponential loss over the parameters θ_t and weight β_t, keeping $\theta_1, ..., \theta_{t-1}$ and $\beta_1, ..., \beta_{t-1}$ fixed

$$Loss_{t-1}(X_i, Y_i) = \exp(Y_i f_{t-1}(X_i))$$

  $\Rightarrow$ optimization problem:

$$\begin{aligned}(\beta_t, \theta_t) &= \arg\min_{\beta,\theta} \sum_i \exp(-Y_i(f(_{t-1}(X_i) + \beta g(X_i|\theta)))) \\ &= \arg\min_{\beta,\theta} \sum_i w^{(t-1)} \exp(-Y_i \beta g(X_i|\theta))\end{aligned}$$

  $w_i^{(t-1)} = \exp(-Y_i f_{t-1}(X_i))$ weight of instane i in training round $t \Rightarrow$ high weight for wrongly classified instances

- the optimum with respect to θ is independent of β

$$\theta_t = \arg\min_{\theta} \sum_i w_i^{(t-1)} \mathbb{1}(Y_i \neq g(X_i|\theta))$$

  which is nothing else than weighted $0-1$ loss

- optimize with respect to β:

  see ESL p342ff for details on the derivation etc

  $\Rightarrow \beta_t = \frac{1}{2} \log \frac{1-\text{err}_t}{\text{err}_t}$
  $\Rightarrow g(X_i|\theta_t)$ should be a weak classifier (non-zero training error) to avoid $\beta_t \to \infty$.

[71] θ_t are the parameters of classifier t

27.3 Adaboost algorithm

- init $f_0(x) = 0, w_t^{(0)} = 1$
- for $t = 1, ..., T$
  - optimize $\theta_t = \arg\min_\theta \sum_i w_i^{(t-1)} \mathbb{1}(Y_i \neq g(X_i|\theta))$
  - compute weighted training error
  - compute classifier weight
  - update instance weights. It goes up for wrongly classified and down for correctly classified examples
  - return $f_T(x)$
  - predict $\hat{y} = \text{sign}(f_T(x))$
- popular choices for $g(X|\theta)$
  - linear classifiers
  - decision stump $\hat{=}$ linear classifier with one feature $\hat{=}$ decision tree with one split node (and two leaves) $\Rightarrow$ very cheap, can use big T ($T = 10^4$)

27.4 Cascaded classifiers (Viola & Jones 2001)

- application: most decisions are simple (example: face detection: most boxes are clearly no faces)
- idea:
  - use cheap classifier for simple decisions and complex ones for difficult decisions
  - control training so that only one type of error (e.g. false positives) occurs in stages $1, ..., (T-1)$
- training: in each stage fix the maximum allowed false negative rate ε ($\hat{=}$ missed faces) and the allowed classifier complexity (e.g. number of features) and train to maximize true negative rate
- prediction: most instances are classified by the first stages $\Rightarrow$ average time is small
  - true positive rate $(1-\varepsilon)^\top$

Overlook Here is a summary of the topics covered in the lecture

Lecture	Lesson	Topic
Lecture 1	First Lesson	Machine Learning, Introduction and Notation, Kinds of variables, Kinds of training data
	Second Lesson	Modeling and Sources of Uncertainity
Lecture 2	First Lesson	Classification, How well does a decision perform?, fundamental methods, theoretical limits
	Second Lesson	How well can we perform?, Bayes decision regions, example
Lecture 3	First Lesson	Bayes Classifier, Candidate decision rules, Errors
	Second Lesson	Nearest Neighbor Classifier: intuitive definition, How well does this work?, empirical estimation (two-fold cross validation) , asymptotic theory
Lecture 4	First Lesson	clustering algorithms, Quadratic and Linear Discriminant Analysis (Introduction), QDA: generative method
	Second Lesson	fundamental meta-algorithm for training in ML, QDA: model, objective function, optimization, result, prediction
Lecture 5	First Lesson	QDA: Summary, LDA: prediction rule, case C = 2, intuitive meaning
	Second Lesson	LDA: general prediction rule
Lecture 6	First Lesson	Logistic Regression, maximum likelihood principle,
	Second Lesson	determine weights w by minimizing the negative log-posterior of the training data, stochastic gradient descent
Lecture 7	First Lesson	Histograms and Density Trees: (lecture so far, summary),
	Second Lesson	Histograms, curse of dimensionality
Lecture 8	First Lesson	How to generalize histograms to high dimensional value spaces, best split criterion
	Second Lesson	split criterion, prediction, improve performance, ensemble

Lecture 9	First Lesson	Regression: Zoo of methods
	Second Lesson	Ordinary Least Squares: model, task, example (fitting in 2D), general case, formal solution
Lecture 10	First Lesson	When to use what: Cholesky, application - tomography
	Second Lesson	weighted least squares
Lecture 11	First Lesson	Total Least Squares: introductory example, correction matrix, task, Constrained / Regularized Least Squares, Ridge Regression
	Second Lesson	Solve non-linear regression problems via linear least-squares
Lecture 12	First Lesson	Kernel Ridge Regression: prediction via the dual,
	Second Lesson	examples, parameter selection, gaussian Kernel, advantages
Lecture 13	First Lesson	Regularized / Constrained Regression purpose of regularization, ridge regression, sparse regression, example, method 1 L_0-regression
	Second Lesson	method 2 L_1-regression, non-linear regression (method 1, 2, 3)
Lecture 14	First Lesson	Robust Regression
	Second Lesson	RANSAC - Algorithm
Lecture 15	First Lesson	Risk, Loss and Optimism or How to judge the quality of predictions?
	Second Lesson	
Lecture 16	First Lesson	Evaluate the model: loss function, training error, sample a test set
	Second Lesson	
Lecture 17	First Lesson	How to determine optimism and / or predict test error: empirical estimates, analytical estimates
	Second Lesson	
Lecture 18	First Lesson	Variants of PCA: non-negative matrix factorization, algorithm, initial guess
	Second Lesson	example: face representation, independent component analysis (ICA), fast ICA, robust PCA,
Lecture 19	First Lesson	Non-linear dimension reduction
	Second Lesson	example: face representation, independent component analysis (ICA), fast ICA, robust PCA,