Contents

1 TA Instructor Info

- Frederik Hvilshøj
- Email: fhvilshoj@gmail.com

2 Important

2.1 Dictionary

- **RHS:** Right Hand Side

2.2 Jacobian

- If we have a function $f(z) : \mathbb{R}^a \to \mathbb{R}^b$ such that $f(z) = [f_1(z), \ldots, f_b(z)]$ then the Jacobian is the matrix

$$J_{i,j} = \frac{\partial f_i}{\partial z_j} \tag{1}$$

of size $b \times a$.

3 The Learning Problem (1)

3.1 Components of Learning

- The main components for the learning problem
  - An input x

- The unknown target function $f : \mathcal{X} \to \mathcal{Y}$
  - $\mathcal{X}$ is the input space
  - $\mathcal{Y}$ is the output space
- There is a set of data $\mathcal{D}$ of input output examples $(x_1, y_1), \cdots, (x_N, y_N)$
  - where $y_n = f(x_n)$ for $n = 1, \ldots, N$
  - often referred to as data points
- The learning algorithm that uses dataset $\mathcal{D}$ to pick a formula $g : \mathcal{X} \to \mathcal{Y}$ that approximates f
  - choose g from a set of candidate formulas under consideration which is called the hypothesis set $\mathcal{H}$
  - $\mathcal{H}$ could be the set of all linear formulas

§

3.2 A Simple Learning Model

- The hypothesis set and learning model is referred informally to as the *learning model*
- A simple learning model (the *perceptron*)
  - Let $\mathcal{X} = \mathbb{R}^d$ where $\mathcal{X} = \mathbb{R}^d$ is the d-dimensional Euclidean space be the input space
  - Let $\mathcal{Y} = \{+1, -1\}$ be the output space
  - The hypothesis set $\mathcal{H}$ is specified through a functional form that all $h \in \mathcal{H}$ share
    - The functional form $h(x)$ chosen is to give weights to the different coordinates of x which reflects their importance
  - The weighted score is compared to a threshold value which decides whether the output is $+1$ or -1

$$\begin{aligned} \text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold} \\ \text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold} \end{aligned} \tag{2}$$

This can be written mode compactly as

$$h(x) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right) \tag{3}$$

where $x_1, \ldots, x_n$ are the components of the vector $\boldsymbol{x}$ and b is the threshold

- The bias can be added as the first weight $w_0 = b$ and adding a fixed input $x_0 = 1$ gives us the same result
  - With this convention the previous equation can be written as

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}) \tag{4}$$

where $\boldsymbol{w}$ is the weight vector and $\boldsymbol{x}$ is the input vector

- The *perceptron learning algorithm* will determine $\boldsymbol{w}$ based on the data
  - To use this the data should be linearly separable
    - Means that there is a $\boldsymbol{w}$ which achieve the correct decision $h(\boldsymbol{x}_n) = y_n$ on all data examples
  - The algorithm finds $\boldsymbol{w}$ using the following simple iterative method
    - At iteration t where $t = 0, 1, 2, \ldots$, there is a current value of the weight vector $\boldsymbol{w}(t)$
    - The algorithm picks an example from $(x_1, y_1), \cdots, (x_N, y_N)$ $(x(t), y(t))$ and uses it to update $\boldsymbol{w}(t)$
    - The update rule is $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + y(t)\boldsymbol{x}(t)$
- The learning algorithm is guaranteed to arrive at the right solution at the end

3.3 Types of Learning

3.3.1 Supervised Learning

- The training data contains explicit examples of the correct output for given inputs
- There are two variations of this protocol
  - **Active learning:** where the data set is acquired though queries that we make

    - We get to choose a point $\boldsymbol{x}$ in the input space and the supervisor reports to us the target value for $\boldsymbol{x}$
    - This opens up a strategic chosen $\boldsymbol{x}$
  - **Online learning:** the data set is given to the algorithm one example at a time
    - This happens when we have streaming data that the algorithm has to process on the run
    - Useful for limitations of computing at storage
    - Can also be used in other paradigms of learning
    - e.g. a movie recommendation system

3.3.2 Reinforcement Learning

- When training data does not explicitly contain the correct output for each input we are no longer in the supervised setting
- The training example does not contain the target output
  - It contains so possible outputs of how good that output is
- The training examples in reinforcement learning are of the form

  (input, some output, grade for this output)

- The examples does not say how good the inputs would have been for other settings
- Can e.g. be useful for learning to play a game

3.3.3 Unsupervised Learning

- In unsupervised learning the data does not contain any output information at all
  - We are just given input examples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- The decision regions in unsupervised learning are the same as the one in supervised learning without label
- It can be viewed as a task of spontaneously finding patterns and structure in input data
- It can be a precursor to supervised learning

3.4 Linear Regression and Orthogonal Projections

- **Lemma 1.** Given a vector y the closest point in V to y, i.e. $\arg\min_v \in V : ||v - y||_2^2$ is the orthogonal projection of y onto V.
- The optimal weight vector w is found using the following formula

$$w = (X^T X)^{-1} X^T y \tag{5}$$

where X is a $n \times d$ data matrix where each row is an input point and a $n \times 1$ vector y of targets

3.5 Is Learning Feasible

3.5.1 General

- To see the relationship between the data $\mathcal{D}$ and the data outside the **Hoeffding Inequality** is used
  - It states for a random variable ν in terms of the parameter μ and the sample size N that

$$\mathbb{P}[|\nu - \mu > \epsilon] \geq 2e^{-2\epsilon^2 N} \text{ for any } \epsilon > 0 \tag{6}$$

- This shows that as one increase the sample size ν gets closer to μ for some small number ϵ
- The error rate within the sample is called the **in-sample error**

$$\begin{aligned} E_{\text{in}}(h) &= (\text{fraction of } D \text{ where } f \text{ and } h \text{ disagree}) \\ &= \frac{1}{N} \sum_{n=1}^{N} [h(\boldsymbol{x}_n \neq f(\boldsymbol{x}_n)] \end{aligned} \tag{7}$$

- where statement = 1 if the statement is true and 0 otherwise
- The **out-of-sample error** is defined as

$$E_{\text{out}}(h) = \mathbb{P}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})] \tag{8}$$

- Using in-sample and out-of-sample error the **Hoeffding Inequality** can be written as

$$[|E_{\text{in}}(h) - E_{\text{out}}(h) > \epsilon] \leq 2e^{-2\epsilon^2 N} \text{ for any } \epsilon > 0 \tag{9}$$

- where N is the number of training examples

3.5.2 Feasibility of Learning

- $\mathcal{D}$ does not deterministic tell us something about f outside of $\mathcal{D}$ but it gives us a probabilistic answer
- Since the **Hoeffding Inequality** tells us that $E_{\text{in}}(g) \approx E_{\text{out}}(g)$ for a large enough N $E_{\text{in}}(g)$ seems like a good proxy for $E_{\text{out}}(g)$
- The feasibility of learning is split into two questions
  1. Can we make sure that $E_{\text{out}}(g)$ is close enough to $E_{\text{in}}(g)$
  2. Can we make $E_{\text{in}}(g)$ small enough?
- **The complexity of $\mathcal{H}$**: If the number of hypothesis M goes up we run more risk that $E_{\text{in}}(g)$ will be a poor estimator of $E_{\text{out}}(g)$
  - M can be though of as a measure of the complexity of the hypothesis set $\mathcal{H}$ that we use
  - The bigger the M the higher the change of finding a small enough $E_{\text{in}}(g)$ becomes
- **The complexity of f**: A more complex f is harder to learn
  - A more complex hypothesis makes the likelihood that the $E_{\text{in}}(g)$ and $E_{\text{out}}(g)$ are approximately the same smaller
  - If the target function f is two hard one may not be able to learn it at all
  - Most target functions in real life are not too complex

3.6 Error and Noise

3.6.1 Error Measures

- An **error measure** quantifies how well each hypothesis h in the target function f

$$\text{Error} = E(h, f) \tag{10}$$

- While $E(h, f)$ is based on the entirety of h and f is almost universally defined based on the error of individual points $\boldsymbol{x}$
  - If we define a pointwise error measure $e(h(\boldsymbol{x}), f(\boldsymbol{x}))$ the overall error will be the average of the pointwise error
  - The defined error should be defined on the use of the application

3.6.2 Noisy target

- In a real world application the data that one learns from is not generated from a deterministic target function but in a noisy way
  - Formally we have a **target distribution** $P(y \mid \boldsymbol{x})$ instead of a taget function
  - One can think of a **noisy target** as a deterministic target, plus added noise

4 Training versus Testing (2)

4.1 Theory of Generalization

4.1.1 General

- The **generalization error** is the discrepancy between E_{in} and E_{out}
  - The Hoeffding Inequality provides a way to characterize it with a probabilistic bound
  - The Hoeffding Inequality can be rephrased as follows: pick a tolerance level δ e.g. 0.05 and assert with probability at least $1 - \delta$ that

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} \tag{11}$$

- It is referred to as the **generalization bound**

4.1.2 Effective Number of Hypotheses

- The **growth function** is the quantity that will formalize the effective number of hypotheses
  - It will replace M when $M = \infty$ in the generalization bound
- **Definition 2.1.** Let $x_1, \ldots, x_N \in \mathcal{X}$. The **dichotomies** generated by $\mathcal{H}$ on these points are defined by

$$\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N = \{(h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_N) \mid h \in \mathcal{H}) \tag{12}$$

- **Definition 2.2.** The **growth function** is defined for a hypothesis set $\mathcal{H}$ by

$$m_{\mathcal{H}}(N) = \max_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N \in \mathcal{X}} |\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)| \tag{13}$$

- where $|\cdot|$ denotes the number of elements of the set
- That $\mathcal{H}$ can **shatter** $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ signifies that $\mathcal{H}$ is as diverse as can be on this particular sample
- **Definition 2.3.** If no data set of size k can be shattered by $\mathcal{H}$, then k is said to be a break point for $\mathcal{H}$
- If k is a break point, then $m_{\mathcal{H}}(k) < 2^k$

4.1.3 Bounding the growth function

- **Definition 2.4.** $B(N, k)$ is the maximum number of dichotomies on N points such that no subset of size k of the N points can be shattered by these dichotomies
- **Lemma 2.3.** (Sauer's Lemma)

$$B(N, k) \leq \sum_{i=0}^{k-1} \binom{N}{i} \tag{14}$$

- **Theorem 2.4.** If $m_{\mathcal{H}}(k) < 2^k$ for some value of k, then

$$m_{\mathcal{H}}(n) \leq \sum_{i=0}^{k-1} \binom{N}{i} \tag{15}$$

- for all N. The RHS is polynomial in N of degree $k-1$

4.1.4 The VC Dimension

- **Definition 2.5.** The **Vapnik-Chervonekis dimension** of a hypothesis set $\mathcal{H}$, denoted by $d_{vc}(\mathcal{H})$ or simply d_{vc} is the largest value of N for which $m_{\mathcal{H}}(N) = 2^N$. If $m_{\mathcal{H}}(N) = 2^N$ for all N, then $d_{vc}(\mathcal{H}) = \infty$.
- No smaller breakpoint than $k = d_{vc} + 1$ exists

$$d_{vc} \geq N \iff \text{there } \textbf{exists } \mathcal{D} \text{ of size } N \text{ such that } \mathcal{H} \text{ shatters } \mathcal{D} \tag{16}$$

- The VC dimension of a d dimensional perceptron is $d + 1$.

4.1.5 The VC Generalization Bound

- **Theorem 2.5.** (VC generalization bound). For any tolerance $\delta > 0$,

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \frac{4m_{\mathcal{H}}(2N)}{\delta}} \tag{17}$$

- with probability $\geq 1 - \delta$

4.2 Interpreting the Generalization Bound

4.2.1 General

- The VC generalization bound is a universal result
  - It applies to all hypotheses set, learning algorithms, input spaces, probability distributions and binary target functions
  - The bound is quite loose
  - It can be used as a guideline for generalization
  - Learning models with lower d_{vc} tend to better than those with higher d_{vc}

4.2.2 Sample Complexity

- The **sample complexity** denotes how many training examples N are needed to achieve a certain generalization performance
  - The performance is specified using two parameters ϵ and δ
    - The error tolerance ϵ determines the allowed generalization error
    - The confidence parameter δ determines how often the error tolerance ϵ is violated
  - How fast N grows as ϵ and δ become smaller indicates the amount of data needed for a good generalization
- From the VC generalization bound it follows that

$$N \geq \frac{8}{\epsilon^2} \ln(\frac{4m_{\mathcal{H}}(2N)}{\delta}) \tag{18}$$

- If $m_{\mathcal{H}}(2N)$ is replaced by its generalization polynomial upper bound we get that

$$N \geq \frac{8}{\epsilon^2} \ln(\frac{4((2N)^{d_{\text{vc}}} + 1)}{\delta}) \tag{19}$$

- The numerical value for N can be obtained using simple iterative methods

4.2.3 Penalty for Model Complexity

- Often we have a fixed dataset, we can the use the Generalization bound to find out what performance we can expect to get

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \frac{4m_{\mathcal{H}}(2N)}{\delta}} \tag{20}$$

- We can again use the polynomial bound based on d_{vc} instead of $m_{\mathcal{H}}(2n)$

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \frac{4((2N)^{d_{\text{vc}}} + 1)}{\delta}} \tag{21}$$

- We often denote the second as $\Omega(N, \mathcal{H}, \delta)$ and call it the penalty

$$\sqrt{\frac{8}{N} \ln \frac{4((2N)^{d_{\text{vc}}} + 1)}{\delta}} \tag{22}$$

- More complex models help E_{in} and hurt $\Omega(N, \mathcal{H}, \delta)$
  - The optimal model is one that minimizes a combination of the two terms

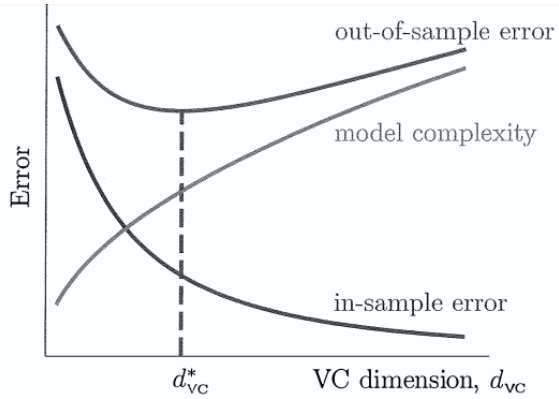


Figure 2.3: When we use a more complex learning model, one that has higher VC dimension d_{vc}, we are likely to fit the training data better resulting in a lower in sample error, but we pay a higher penalty for model complexity. A combination of the two, which estimates the out of sample error, thus attains a minimum at some intermediate d^*_{vc}.

4.2.4 The Test Set

- One often estimates E_{out} by using a **test set** that the learning algorithm has not seen before
  - Called E_{test}

4.3 Approximation-Generalization Tradeoff

4.3.1 Bias and Variance

- The bias variance decomposition out-of-sample error is

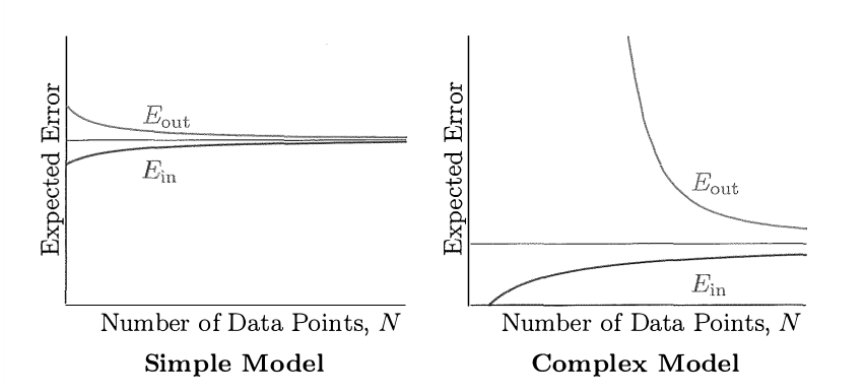
$$E_{\text{out}}(g^{(\mathcal{D})}) = \mathbb{E}_{\boldsymbol{x}}\left[(g^{(\mathcal{D})}(\boldsymbol{x}) - f(\boldsymbol{x}))^2\right] \tag{23}$$

- The function $\bar{g}(\boldsymbol{x})$ can be interpreted in the following operational way
  1. Generate many data sets $\mathcal{D}_1, \ldots, \mathcal{D}_K$
  2. Apply the learning algortihm to each data set obtaining final hypotheses $g_1, \ldots, g_K$.
  3. The average function for any $\boldsymbol{x}$ is then estimated by $\bar{g}(\boldsymbol{x}) \approx \frac{1}{K}\sum_{k=1}^{K} g_k(\boldsymbol{x})$
- The **bias** is $\text{bias}(\boldsymbol{x}) = (\bar{g}(\boldsymbol{x}) - f(\boldsymbol{x}))^2$
  - It measures how much the average function deviates from the target function
- The **variance** is

$$\text{var}(\boldsymbol{x}) = E_{\mathcal{D}}[(g^{\mathcal{D}}(x) - \bar{g}(\boldsymbol{x}))^2] \tag{24}$$

- Says how much the different hypotheses varies
- Since bias and variance cannot be computed in a real model
  - They are purly a conceptual tool used when developing a model

4.3.2 The Learning Curve



- For a simpler model the learning curves converge more quickly but to worse ultimate performance
  - The in-sample error learning curve is increasing in N
  - The out-of-sample error learning curve is decreasing in N

5 The Linear Model (3)

5.1 Linear Regression

5.1.1 The algorithm

- The linear regression algorithm is based on minimizing the squared error between $h(x)$ and y

$$E_{\text{out}}(h) = \mathbb{E}[(h(\boldsymbol{x}) - y)^2] \tag{25}$$

where the expected value is taken with respect to the joint probability distribution $P(x, y)$

- The goal is to find an hypothesis that achieves a small $E_{\text{out}}(h)$
  - Since the distribution $P(\boldsymbol{x}, y)$ is unknown $E_{\text{out}}(h)$ cannot be computed the in-sample version is therefore used instead

$$E_{\text{in}}(h) = \frac{1}{n}\sum_{n=1}^{N}(h(\boldsymbol{x}_n) - y_n)^2 \tag{26}$$

- In linear regression h takes the form of a linear combination of the components of x that is

$$h(\boldsymbol{x}) = \sum_{i=0}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x} \tag{27}$$

where $x_0 = 1$ an $\boldsymbol{x} \in \{1\} \times \mathbb{R}^d$ as usual and $\boldsymbol{w} \in \mathbb{R}^{d+1}$

- For the special case of linear h, it is very useful to have a matrix representation of $E_{\text{in}}(h)$
  - First we define the data matrix $X \in \mathbb{R}^{N \times (d+1)}$ to be the $N \times (d+1)$ matrix whose rows are the inputs x_n as row vector
  - Define the target vector $\boldsymbol{y} \in \mathbb{R}^N$ to be the column vector whose components are the target values y_n

Linear regression algorithm:

1: Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, where each $\mathbf{x}$ includes the $x_0 = 1$ bias coordinate, as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1^{\mathsf{T}}— \\ —\mathbf{x}_2^{\mathsf{T}}— \\ \vdots \\ —\mathbf{x}_N^{\mathsf{T}}— \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2: Compute the pseudo-inverse $\mathrm{X}^\dagger$ of the matrix X. If $\mathrm{X}^{\mathsf{T}}\mathrm{X}$ is invertible,

$$\mathrm{X}^\dagger = (\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}.$$

3: Return $\mathbf{w}_{\text{lin}} = \mathrm{X}^\dagger \mathbf{y}$.

5.2 Logistic Regression

5.2.1 Predicting a Probability

- To predict a probability we want something which restricts the output to the probability range $[0, 1]$, one choice that accomplishes this goal is the logistic regression model

$$h(\boldsymbol{x}) = \theta(\boldsymbol{w}^T \boldsymbol{x}) \tag{28}$$

- Where θ is the *logistic* function $\theta(s) = \frac{e^s}{1+e^s}$ whose output is between 0 and 1
  - The output can be interpret as a probability for a binary event
  - The logistic function θ is referred to as a **soft threshold** in contrast to the **hard threshold** in classification
  - It is also called a **sigmoid**
- When using Logistic Regression we are formally trying to learn the target function

$$f(\boldsymbol{x}) = \mathbb{P}[y = +1 \mid \boldsymbol{x}] \tag{29}$$

- The data given is generated by a noisy target $P(y \mid \boldsymbol{x})$

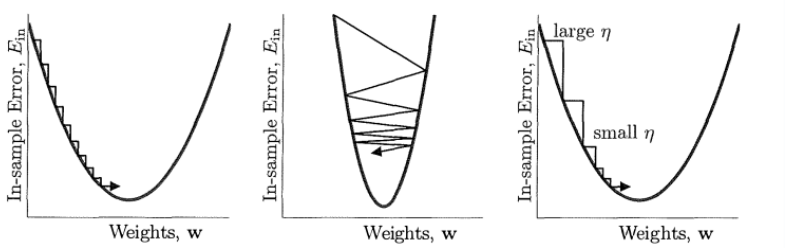
$$P(y \mid \boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{for } y = +1 \\ 1 - f(\boldsymbol{x}) & \text{for } y = -1 \end{cases}$$

- The standard **error measure** $e(h(\boldsymbol{x}), y)$ used in logistic regression is based how likely it is that we would get this output y from the input $\boldsymbol{x}$. if the target distribution $P(y|\boldsymbol{x})$ was indeed captured by our hypothesis $h(x)$.
- The **error measure** is $E_{\text{in}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \boldsymbol{w}^T x_n})$

5.2.2 Gradient Descent

1. Batch Gradient Descent
   - Gradient descent is a general technique for minimize a twice differentiable function
     - e.g. $E_{\text{in}}(\boldsymbol{w})$ in logistic regression
     - You start somewhere and go the steepest way down the surface
     - You may end up in a local minima
     - When using a convex function such as E_{in} there is only one minima the global unique minimum

- When steeping in a direction you need that the step η is not too small or too large



η too small η too large variable η just right

- You typically want to choose $\eta_t = \eta || \bigtriangledown E_{\text{IN}} ||$ to obtain a good variable step size

Fixed learning rate gradient descent:

1: Initialize the weights at time step $t = 0$ to $\mathbf{w}(0)$.
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Compute the gradient $\mathbf{g}_t = \nabla E_{\text{in}}(\mathbf{w}(t))$.
4: Set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$.
5: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$.
6: Iterate to the next step until it is time to stop.
7: Return the final weights.

- A typical good choice for η is a fixed learning rate is around 0.1 the

Logistic regression algorithm:

1: Initialize the weights at time step $t = 0$ to $\mathbf{w}(0)$.
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: Compute the gradient

$$\mathbf{g}_t = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathrm{T}}(t) \mathbf{x}_n}}.$$

4: Set the direction to move, $\mathbf{v}_t = -\mathbf{g}_t$.
5: Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$.
6: Iterate to the next step until it is time to stop.
7: Return the final weights $\mathbf{w}$.

- **Initialization**
  - Most of the time the initializing the initial weights as zeros works well
  - It is in general safer to initialize the weights randomly
  - Choosing each weight independently from a Normal distribution with zero mean and small variance usually works well
- **Termination**
  - A simple approach would be to set and upper limit on the number of iterations
    - Does not guarantee anything on the quality of the final weights
  - A natural terminal criterion would be to stop ones $||\boldsymbol{g}_t||$ drops below a certain threshold
    - Eventually this must happen but we do not know but we will now know when
  - For logistic regression a combination of the two termination conditions is used

2. Stochastic Gradient Descent (SGD)

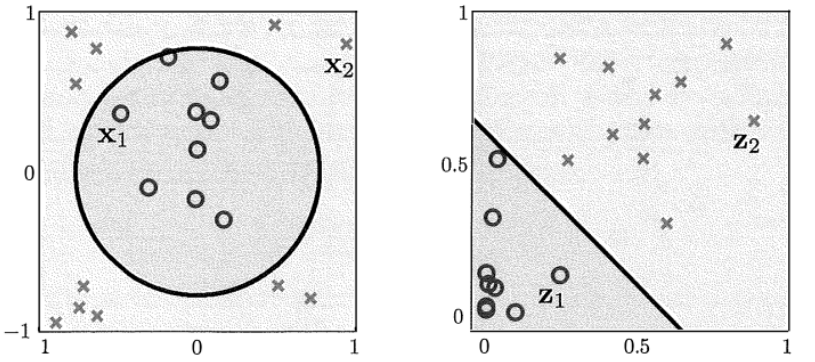
- A sequential version of Batch Gradient Decent
  - Often beats the batch version in practise
- Instead of considering the full batch gradient on all N training points, we consider a stochastic version of the gradient
  (a) Pick a training data point $(\boldsymbol{x}_n, y_n)$ at uniformly random
  (b) Consider only the error on that point (in case of logistic regression)

$$e_n(w) = \ln(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n}) \tag{31}$$

- The gradient needed is

$$\nabla e_n(\boldsymbol{w}) = \frac{-y_n \boldsymbol{x}_n}{1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n}} \tag{32}$$

- The weight update is $\boldsymbol{w} \leftarrow \boldsymbol{w} \cdot \eta \nabla e_n(\boldsymbol{w})$



(a) Data in $\mathcal{X}$-space (b) Transformed data in $\mathcal{Z}$ space

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{z} = \Phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Figure 1: Example of nonlinear transform

5.3 Nonlinear Transformation

5.3.1 The $\mathcal{Z}$ space

- Using a nonlinear transformation we can convert data which is not linear separable into data that is
  - The space $\mathcal{Z}$ generated is called the **feature space**
  - The transformation from the original space $\mathcal{X}$ to $\mathcal{Z}$ is called a **feature transform**
- Any linear hypothesis $\tilde{h}$ in $\boldsymbol{z}$ corresponds to a (possible nonlinear) hypothesis of $\boldsymbol{x}$ given by $h(\boldsymbol{x}) = h(\theta(\boldsymbol{x}))$ where θ is a non linear transform
  - The set of these hypothesis is denoted by $\mathcal{H}_\theta$
- The feature transform θ_Q is defined for degree-Q curves in $\mathcal{X}$
  - It is called the Qth order polynomial transform

p- The power of the feature transform should be used with care, it may not be worth it to insist on linear separability and employ a highly complex surface

- It is sometime better to tolerate a small E_in than using a feature transform

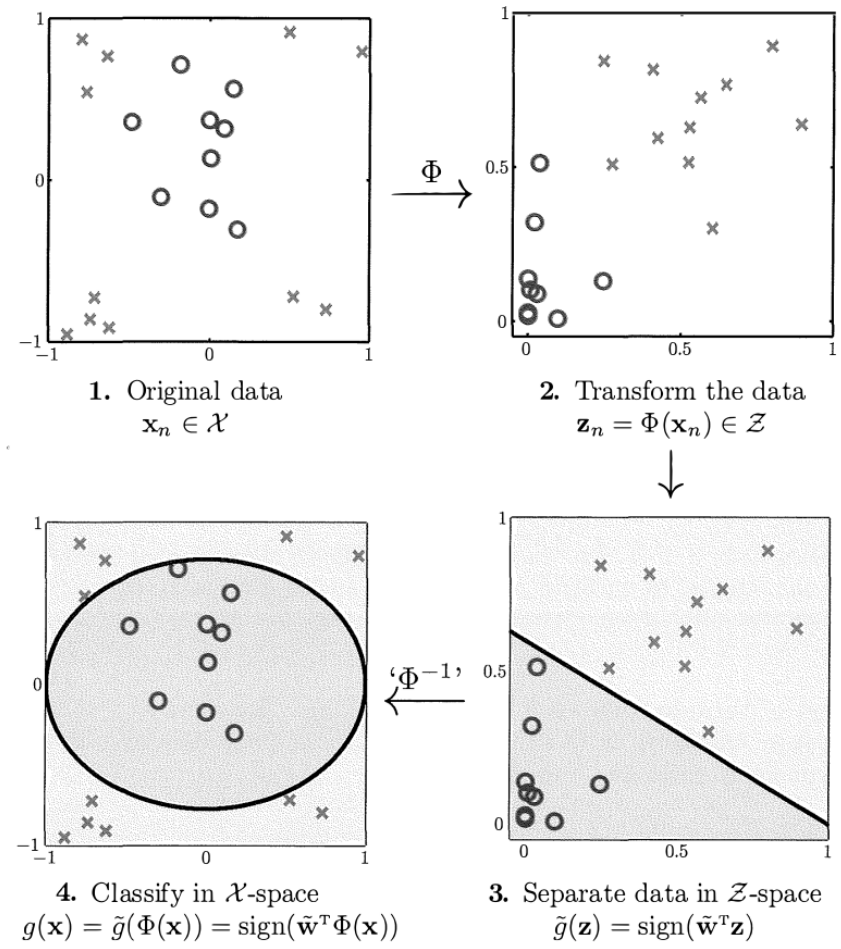


Figure 2: The nonlinear transform for separating non separable data.

5.3.2 Computation and generalization

- Computation is an issue because θ_Q maps a two dimensional vector $\boldsymbol{x}$ to $\tilde{d} = \frac{Q(Q+3)}{2}$ dimensions, which increases the memory and computational cost
  - Things could get worse if $\boldsymbol{x}$ is in a higher dimension to begin with
- The problem of generalization is another important issue
  - We will have a weaker guarantee that E_{OUT} is small
  - It is sometime balanced by the advantage we get in approximating the target better

higher $\tilde{d}$	better chance of being linearly separable ($E_{\text{in}} \downarrow$)	$d_{\text{vc}} \uparrow$
lower $\tilde{d}$	possibly not linearly separable ($E_{\text{in}} \uparrow$)	$d_{\text{vc}} \downarrow$

6 Multinomial/Softmax Regression

6.1 Setup

- Multinomial/Softmax Regression generalizes logistic regression to handle K classes instead of 2
  - A target value y is represented as a vector of length K with all zeroes except one which is called a one-in-K encoding
  - To store all the data points a matrix Y of size $n \times K$ and the data matrix X is unchanged

$$X = \begin{pmatrix} 1 & - & x_1^T & - \\ \vdots & \vdots & \vdots & \\ 1 & - & x_n^T & - \end{pmatrix} \in \mathbb{R}^{n \times d} \qquad y = \begin{pmatrix} - & y_1^T & - \\ - & \vdots & - \\ - & y_n^T & - \end{pmatrix} \in \{0,1\}^{n \times K} \tag{33}$$

- To generalize to K classes we will use K weight vectors $w_1, \ldots, w_k$ each of length d, one for each class.
  - To classify data we can use the following algorith: Given data x, compute $w_i^\intercal x$ for $i = 1, \ldots, K$ and return the index of the largest value.
  - The list of weight vectors is packed into a matrix W of size $d \times K$ by putting w_1 in column one and so on.

$$W = \begin{pmatrix} (w_1)_1 & \dots & (w_K)_1 \\ \vdots & \vdots & \vdots \\ (w_1)_d & \dots & (w_K)_d \end{pmatrix} \in \mathbb{R}^{d \times n}$$

- This way we can compute the weighed sum for each class by the vector matrix product $x^\intercal W$ and then pick argmax of that to do the classification. Pretty Neat!.
- Numpy example

```
import numpy as np
# example with 3 classes and d = 10
W = np.random.rand(10, 3)
print('Shape w:', W.shape)
x = np.array([1., 2., 3., 4., 5., 6., 7., 8., 9., 10.0]).reshape(10, 1)
print('Shape x:', x.shape)
model_predictions = x.T @ W
print('model (unnormalzed log) predictions: - picke the larger one\n', model_predictio
```

6.2 Probabilistic Outputs

- Given a set of model parameters W and a data point x we want $P(y = i \mid x, W)$ for $i = 1, \dots K$.
- **Softmax** is used in our probabilistic model
  - It takes as input a vector of length K and outputs another vector of the same length K, that is a mapping from the K input numbers into K **probabilities**

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}} \quad \text{for} \quad j = 1, \dots, K. \tag{34}$$

- where $\text{softmax}(x)_j$ denote the j'th entry in the vector.
  - The denominator acts as a normalization term that ensures that the probabilities sum to one
  - The exponentiation ensures all numbers are positive.
  - We get the following derivatives:

$$\frac{\partial \, \text{softmax}(x)_i}{\partial x_j} = (\delta_{i,j} - \text{softmax}(x)_j)\text{softmax}(x)_i \qquad \text{where} \qquad \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases} \tag{35}$$

- The following is our probabilistic model

$$p(y \mid x, W) = \text{softmax}(W^\intercal x) = \begin{cases} \text{softmax}(W^\intercal x)_1 & \text{if } y = e_1, \\ \vdots & \\ \text{softmax}(W^\intercal x)_K & \text{if } y = e_K. \end{cases} \tag{36}$$

- We compute the likelihood of the data given a fixed matrix of parameters.
  - The notation $[z]$ for the indicator function

$$P(D \mid W) = \prod_{(x,y)\in D} \prod_{j=1}^{K} \text{softmax}(W^\intercal x)_j^{[y_j=1]} = \prod_{(x,y)\in D} y^\intercal \text{softmax}(W^\intercal x).$$

- This way of expressing is the same as we did for logistic regression.

6.3 The Negative Log Likelihood

- The negative log likelihood of the data is minimized instead of maximizing the likelihood of the data and get a pointwise sum.

$$\text{NLL}(D \mid W) = -\sum_{(x,y)\in D} \sum_{j=1}^{K} [y_j = 1] \ln(\text{softmax}(W^\intercal x)_j) \tag{37}$$

$$= -\sum_{(x,y)\in D} y^\intercal \ln(\text{softmax}(W^\intercal x)) \tag{38}$$

- In the last summation only one value will be nonzero:

$$-\ln \text{softmax}(z)_j = \ln\left(\frac{e^{z_j}}{\sum_{i=1}^{d} e^{z_i}}\right) = -(z_j - \ln \sum_{i=1}^{d} e^{z_i}) \tag{39}$$

- The insample error is defined to be $E_{\text{in}} = \frac{1}{|D|}\text{NLL}$
  - Cannot be solved for a 0 analytically
  - To apply stochastic mini-batch gradient descent as for Logistic Regression all you really need is the gradient of the negative log likelihood function.
    - The gradient is a **simple** generalization of the one used in logistic regression.

* There is a set of parameters for each of K classes, W_j for $j = 1, \ldots, K$
* The gradient is

$$\nabla \text{NLL}(W) = -X^\intercal (Y - \text{softmax}(XW)),$$

6.4 Implementation Issues

6.4.1 Numerical Issues with Softmax

- There are some numerical issues with the softmax function

$$\text{softmax}(x)_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}} \text{ for } j = 1, \ldots, K.$$

- This is because this is a sum of exponentials and exponentiation of numbers tend to make them very large giving numerical problems.
- The problematic part is the logarithm of the sum of exponentials.
- We can move e^c for any constant c outside the sum easily, that is:

$$\ln\left(\sum_i e^{x_i}\right) = \ln\left(e^c \sum_i e^{x_i - c}\right) = c + \ln\left(\sum_i e^{x_i - c}\right).$$

- We need to find a good c, and we choose $c = \max_i x_i$
- Since e^{x_i} is the dominant term in the sum. We are less concerned with values being inadvertently rounded to zero since that does not

6.4.2 One in k encoding

- Representing a number k in $[1, \ldots, k]$ as a vector of length may K be quite cumbersome.
  - In general the input labels will just be a list/vector of numbers between 1 and k.
  - It is your job to transform it into a matrix if needed.
  - But this will be a very sparse matrix.
  - It may be worthwhile to consider whether it is possible to implement the operations with the matrix Y without actually creating the matrix.

6.4.3 Always check your shapes

- If the shapes dont fit then
  - If trying to implement softmax it is very useful to ensure you have full control over the shapes of all matrices and vectors you use.
  - If there is a shape mismatch then clearly there is a larger issue.
  - Checking shapes is a very efficient heuristic for catching bugs.

6.4.4 Bias Variable

- If you need a Bias variable b (remember $w^\intercal x + b$) for each class you need to add a columns of ones to X and make W a $d + 1 \times K$ matrix.

7 Overfitting (4)

7.1 When Does Overfitting Occur?

7.1.1 General

- The main case of overfitting is when you pick the hypothesis with lower E_{in} and it results in higher E_{out}
  - Means that E_{in} alone is no longer a good guide for learning
  - A typical overfitting scenario is when a complex model uses its addition degress of freedom to "learn" the noise

7.1.2 Catalysts for Overfitting

Number of data points	$\uparrow$	Overfitting	$\downarrow$
Noise	$\uparrow$	Overfitting	$\uparrow$
Target complexity	$\uparrow$	Overfitting	$\uparrow$

- On a finite data set the algorithm inadvertently uses some of the degrees of freedom to fit the noise
  - Can result in overfitting and a spurious final hypothesis
- There are two types of noise which that algorithm cannot differentiate

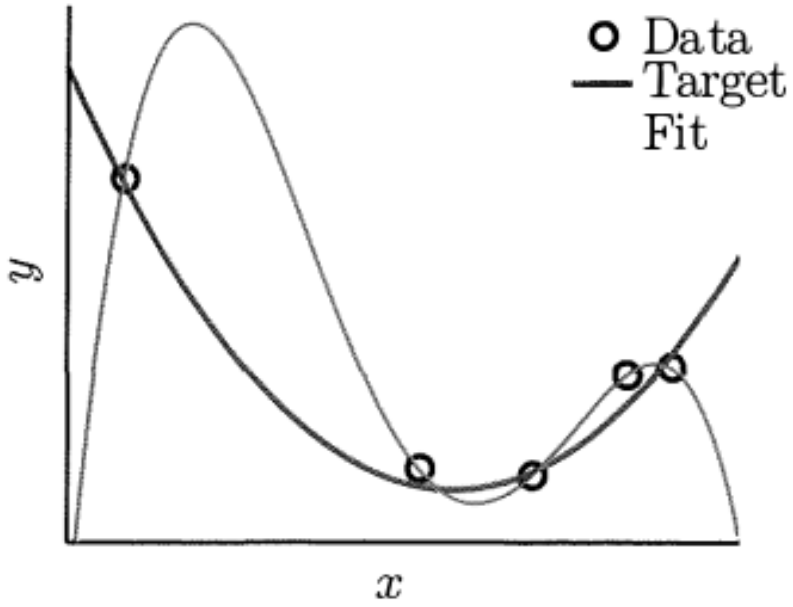


Figure 3: Example of overfitting

- **Deterministic noise** will not change if the dataset was generated again
  - Is different depending on which model we use
  - Related to the bias
- **Stochastic noise** will change if the dataset was generated again
  - Related to the variance

7.2 Regularization

7.2.1 General

- **Regularization** is a way to combat overfitting
  - Constraints the learning algorithm to improve out-of-sample error
  - Especially when noise is present
- A view of regularization is thought the VC bound, which bounds E_{out} using a model complexity penalty $\Omega(\mathcal{H})$:

$$E_{\text{out}}(h) \leq E_{\text{in}}(h) + \Omega(\mathcal{H}) \text{ for all } h \in \mathcal{H} \tag{40}$$

- We are better off fitting the data using a simple $\mathcal{H}$
- Instead on minimizing $E_{\text{in}}(h)$ alone one minimizes the combination of $E_{\text{in}}(h)$ and $\Omega(h)$
  - Avoids overfitting by constraining the learning algorithm to fit data well using a simple hypotheses

7.2.2 A Soft Order Constraint

- A **Soft Order Constraint** can be defined as the hypotheses set

$$\mathcal{C} = \{h \mid h(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}, \boldsymbol{w}^T\boldsymbol{w} \leq C\} \tag{41}$$

- Solving for $\boldsymbol{w}_{\text{reg}}$:
  - If $\boldsymbol{w}_{\text{lin}}^T\boldsymbol{w}_{\text{lin}} \leq C$ then $\boldsymbol{w}_{\text{reg}} = \boldsymbol{w}_{\text{lin}}$ since $\boldsymbol{w}_{\text{lin}} \in \mathcal{H}(C)$
  - If $\boldsymbol{w}_{\text{lin}} \notin \mathcal{H}(C)$ then not only is $\boldsymbol{w}_{\text{lin}}^T\boldsymbol{w}_{\text{lin}} \leq C$ but $\boldsymbol{w}_{\text{lin}}^T\boldsymbol{w}_{\text{lin}} = C$
    - The weights $\boldsymbol{w}$ must lie on the surface of thee \$sphere $\boldsymbol{w}^T\boldsymbol{w} = C$

- If $\boldsymbol{w}_{\text{reg}}$ is to be optimal then for some positive parameter λ_C

$$\nabla E_{\text{in}}(\boldsymbol{w}_{\text{reg}}) = -2\lambda_C \boldsymbol{w}_{\text{reg}} \tag{42}$$

- ∇E_{in} must be parallel to $\boldsymbol{w}_{\text{reg}}$

For some $\lambda_C > 0$ $\boldsymbol{w}_{\text{reg}}$ locally minimizes

$$E_{\text{in}}(\boldsymbol{w}) + \lambda_C W^T W \tag{43}$$

7.2.3 Weight Decay and Augmented Error

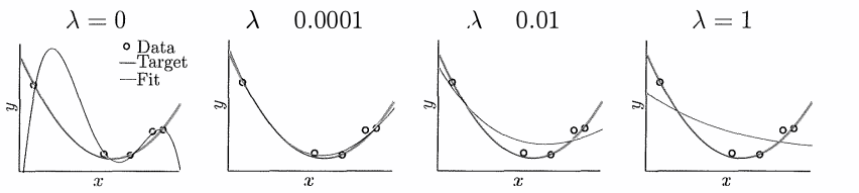


Figure 4.5: Weight decay applied to Example 4.2 with different values for the regularization parameter λ. The red fit gets flatter as we increase λ.

- The **augmented error** is defined as

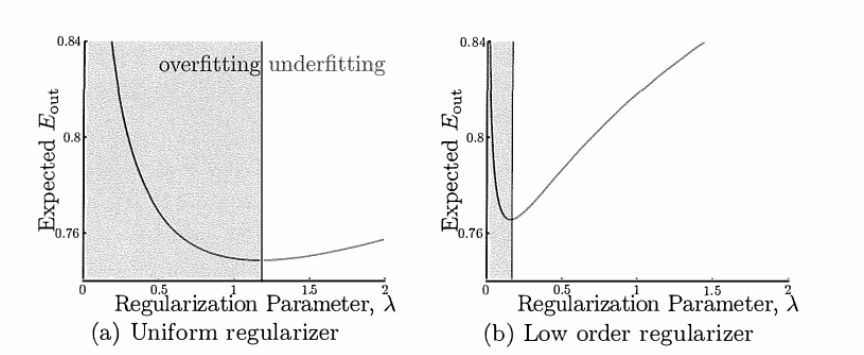
$$E_{\text{aug}}(\boldsymbol{w}) = E_{\text{in}}(\boldsymbol{w}) + \lambda \boldsymbol{w}^T \boldsymbol{w} \tag{44}$$

- where $\lambda \geq 0$ is now a free parameter
- The penalty term enforces a trade-off between making the in-sample error small and making the weights small
  - Is also known as the **weight decay**
  - Minimizing the error together with the decay is known as **ridge regression**
- If we can find the optimal λ^* we can minimize the out-of-sample error
- In general the **augmented error** for a hypothesis set $h \in \mathcal{H}$ is

$$E_{\text{aug}}(h, \lambda, \Omega) = E_{\text{in}}(h) + \frac{\lambda}{N}\Omega(h) \tag{45}$$

- For weight decay $\Omega(h) = \boldsymbol{w}^T \boldsymbol{w}$
  - The need for regularization goes down as the number of data points goes up

7.2.4 Choosing a Regularizer



(a) Uniform regularizer (b) Low order regularizer

- A uniform regularizer, is a penalizes the weights equally
  - encourages all weights to be small uniformly
  - Example $\Omega_{\text{unif}}(\boldsymbol{w}) = \sum_{q=0}^{15} w_q^2$
- A lower-order regularizer
  - Pairs more attention to the higher order weights
  - Example $\Omega_{\text{low}}(\boldsymbol{w}) = \sum_{q=0}^{15} q w_q^2$
- The price paid for overfitting is generally more severe than underfitting
- The optimal value for the regularization parameter increases with noise
- No regularizer will be ideal for all settings
  - Not even specific settings
  - The entire burden rest on picking the right λ
- Some for of regularization is necessary as learning is quite sensitive to stochastic and deterministic noise

7.3 Validation

7.3.1 The Validation Set

- **Validation** tries to estimate the out-of-sample error directly
- The idea of a **validation set** is almost identical to that of a test set
  - A subset of the data is removed and not used in training

  - Will be used to make certain choice in the learning process
    * Therefore not a test set

- The **validation set** is created and used in the following way

  1. Partition the data set $\mathcal{D}$ using into a training set $\mathcal{D}_{\text{train}}$ of size $(N-K)$ and a validation set $\mathcal{D}_{\text{val}}$ of size K
     - Any partitioning method which does not depend on the data will do
  2. Run the learning algorithm using the training set $\mathcal{D}_{\text{train}}$ to obtain a final hypothesis $g^- \in \mathcal{H}$
  3. The validation error is then computed for g using the validation set $\mathcal{D}_{\text{val}}$

$$E_{\text{val}}(g^-) = \frac{1}{K} \sum_{\boldsymbol{x}_n \in \mathcal{D}_{\text{val}}} e(g^-(\boldsymbol{x}_n), y_n) \tag{46}$$

- where $e(g(\boldsymbol{x}), y)$ is the pointwise error measure

- The validation error is an *unbiased* estimate of E_{out} because the final hypothesis g^- was created independently of the validation set
  - The expected error of E_{val} is E_{out}
  - If K is neither too small nor too large E_{val} is a good estimate of E_{out}
  - A rule of thumb in practise is to set $K = \frac{N}{5}$

- We should not output g^- we should output g which is trained on the entire hypothesis set D
  - To estimate E_{out} we use that $E_{\text{out}}(g) \leq E_{\text{out}}(g^-)$ because of the learning curve
    * Is not rigorously proved
    * It is just very likely

7.3.2 Model Selection

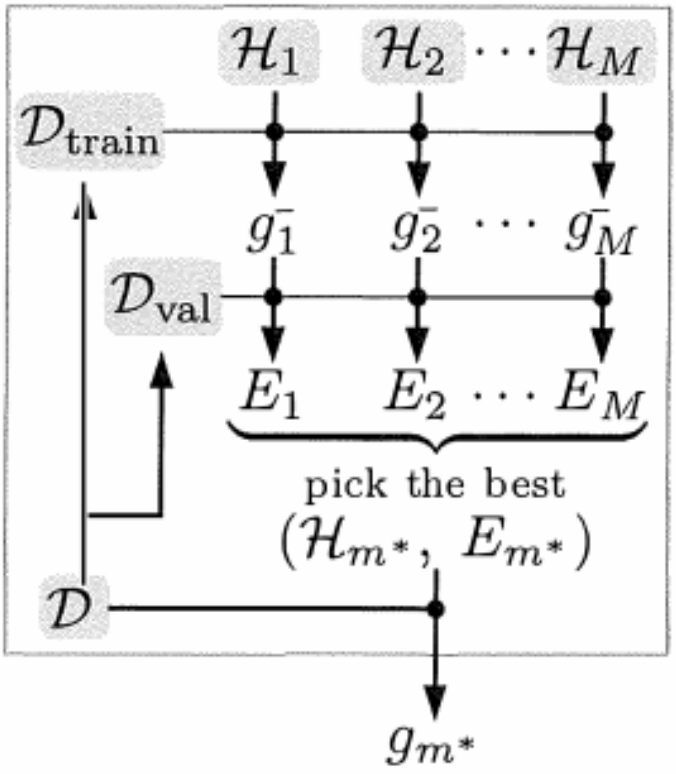


Figure 4.11: Using a validation set for model selection

- The most important use of validation is for **model selection**
  - Choosing linear or nonlinear, polynomial or not...
  - It could any choice that affects the learning process
- It can be used to estimate the out-of-sample error for more than one model, suppose we have M models $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  - Validation can be used to select one of these models
  - Use the training set $\mathcal{D}_{\text{train}}$ to learn the final hypothesis g_m^- for each model

- Evaluate each model on the validation set to obtain the validation errors $E_1, \ldots, E_M$ where

$$E_m = E_{\text{val}}(g_m^-); \text{ for } m = 1, \ldots, M \tag{47}$$

- Then just select the model with the lowest validation error.
- For suitable K even g_{m*}^- is better than in-sample selection of the model
- The validation error can also be used to select a lambda by using $(\mathcal{H}, \lambda_1), (\mathcal{H}, \lambda_2), \ldots, (\mathcal{H}\lambda_M)$ as our M different models
- The more one uses the validation set to fine tune the model the more the it becomes like the training set

7.3.3 Cross Validation

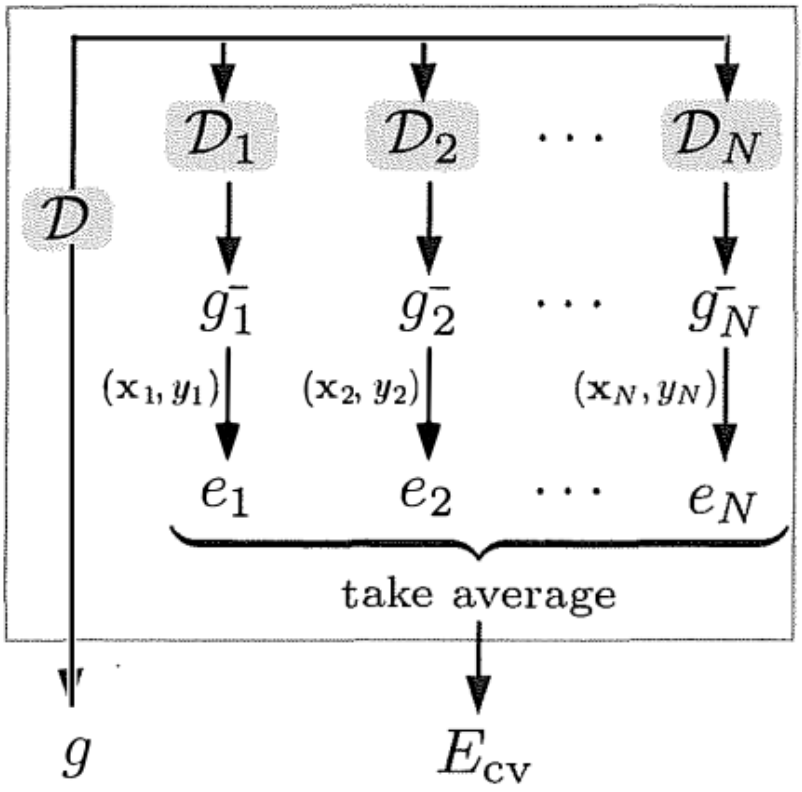


Figure 4.14: Using cross validation to estimate E_{out}

- There are N ways to partition a set of size $N-1$ and a validation set of size 1. Let

$$\mathcal{D}_n = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{n-1}, y_{n-1}), (\boldsymbol{x}_{n+1}, y_{n+1}), \ldots, (\boldsymbol{x}_N, y_N) \tag{48}$$

- The final hypothesis learned from $\mathcal{D}_n$ is denoted g_n^-
- Let e_n be the error made by g_n^- on its validation set which is just a single point $\{(\boldsymbol{x}_n, y_n)\}$

- The cross validation estimate is the average value of the e_n's

$$E_{\text{cv}} = \frac{1}{N}\sum_{n=1}^{N} e_n \tag{49}$$

- **Theorem 4.4.** E_{cv} is an unbiased estimate of $\bar{E}_{\text{out}}(N-1)$
  - The expectation of the model performance, $\mathbb{E}[E_{\text{out}}]$), over data set of size $N-1$
- The cross validation estimate will on average be an upper estimate for the out-of-sample error: $E_{\text{out}}(g) \leq E_{\text{cv}}$
- Cross validation can be for model selection for a given set of models $\mathcal{H}_1, \ldots, \mathcal{H}_M$ in the same way as validation set

Cross validation for selecting λ:

1: Define M models by choosing different values for λ in the augmented error: $(\mathcal{H}, \lambda_1), (\mathcal{H}, \lambda_2), \ldots, (\mathcal{H}, \lambda_M)$
2: **for** each model $m \quad 1, \ldots, M$ **do**
3: Use the cross validation module in Figure 4.14 to estimate $E_{\text{cv}}(m)$, the cross validation error for model m.
4: Select the model m^* with minimum $E_{\text{cv}}(m^*)$.
5: Use model $(\mathcal{H}, \lambda_{m^*})$ and all the data $\mathcal{D}$ to obtain the final hypothesis g_{m^*}. Effectively, you have estimated the optimal λ.

- To get cross validation for M models and a data set D of size N is requires MN rounds of learning
- If one could analytically obtain E_{cv} it would be a big bonus
  - Analytical results are hard to come
  - An analytical method exists for linear models
- The cross validation estimate can be analytically computed as

$$E_{\text{cv}} = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{\hat{y}_n - y_n}{1 - H_n n(\lambda)}\right)^2 \tag{50}$$

- where $H(\lambda) = Z(Z^T Z + \lambda I)^{-1} Z^T$

8 Support Vector Machines

8.1 Notation

- The classifier considered will be a linear classifier for a binary classification problem with labels y and features x
  - $x \in \{-1, 1\}$
  - The classifier with parameters w and b is written as

$$h_{w,b}(x) = g(w^T x + b) \tag{51}$$

- where g = 1 if $z \geq 0$ and $g(z) = -1$ otherwise

8.2 Functional and geometric margins

- Given a training example $(x^{(i)}, y^{(i)})$, the **functional margin** of (w, b) is defined with respect to the training example

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b) \tag{52}$$

- A large functional margin represents a confident and a correct prediction

- Given a training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ the functional margin of (w, b) with respect to S is

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)} \tag{53}$$

- The **geometric margin** of (w, b) with respect to a training example $(x^{(i)}, y^{(i)})$ to be

$$\gamma^{(i)} = y(i)\left(\left(\frac{w}{||w||}\right)^T x^{(i)} + \frac{b}{||w||}\right) \tag{54}$$

- If $||w|| == 1$ then the functional margin equals the geometric margin

- Given a training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ the **geometric margin** of (w, b) with respect to S is

$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)} \tag{55}$$

8.3 The optimal margin classifier

- The problem of finding a decision boundary which has the largest geometric margins is the following optimisation problem

$$\begin{aligned} &\max_{\gamma,w,b} \gamma \\ &\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq \gamma,\ i = 1, \ldots, m \\ &\quad ||w|| = 1 \end{aligned} \tag{56}$$

- this can be be turned into the following problem using functional margins and rescaling it

$$\begin{aligned} &\max_{\gamma,w,b} \frac{1}{2}||w||^2 \\ &\text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1,\ i = 1, \ldots, m \end{aligned} \tag{57}$$

- It is called the **optimal margin classifier**

8.4 Lagrange duality

- Consider a problem of the following form:

$$\begin{aligned} &\min_w f(w) \\ &\text{s.t. } h_i(w) = 0,\ i = 1, \ldots, l \end{aligned} \tag{58}$$

- The **Lagrangian** is defined to be

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w) \tag{59}$$

- The β_i's are called the **Lagrange multipliers**
  - We would the find and set $\mathcal{L}$'s partial derivatives to zero

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0; \frac{\partial \mathcal{L}}{\partial \beta_i} = 0 \tag{60}$$

and solve for w and β

- The **primal** optimization problem is the following

$$\begin{aligned} &\min_w f(w) \\ &\text{s.t. } g_i(w) \leq 0,\ i = 1, \ldots, k \\ h_i(w) = 0,\ i = 1, \ldots, l \end{aligned} \tag{61}$$

- The primal optimization problem is solved by defining the **generalized Lagrangian**

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{k} \beta_i h_i(w) \tag{62}$$

- The α_i's and β_i's are the Lagrange multipliers.
- Consider the quantity

$$\theta_{\mathcal{P}}(w) = \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) \tag{63}$$

- If w violates any of the primal constraints then one should be able to verify that

$$\theta_{\mathcal{P}}(w) = \infty \tag{64}$$

- If w does not violate the constraints then $\theta_{\mathcal{P}}(w) = f(w)$ and therefore the minimization problem

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) \tag{65}$$

- is the same as the original problem
  - the objective is called the **value** of the primal problem and is denoted $p* = \min_w \theta_{\mathcal{P}}(w)$
- If one define

$$\theta_D(\alpha, \beta) = \min_w \ \mathcal{L}(w, \alpha, \beta) \tag{66}$$

- The "$\mathcal{D}$" subscript stands for dual
  - This can be used to pose the **dual** optimization problem

$$\max_{\alpha,\beta:\alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha,\beta:\alpha_i \geq 0} \min_w \ \mathcal{L}(w, \alpha, \beta) \tag{67}$$

- Which is exactly the same as the primal problem, except the order of the min and max has been exchanged
  - The solution to the dual problem is defined as d^*
  - It holds that $d^* \leq p^*$
- The KKT conditions on w^*, α^* and β^*

$$
\begin{aligned}
\frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, n \\
\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, l \\
\alpha_i^* g_i(w^*) &= 0, \quad i = 1, \ldots, k \\
g_i(w^*) &\leq 0, \quad i = 1, \ldots, k \\
\alpha^* &\geq 0, \quad i = 1, \ldots, k
\end{aligned}
$$

- If some w^*, α^*, β^* satisfy the KKT conditions they are also a solution to the primal and dual problems

8.5 Optimal margin classifiers

- By using the KKT conditions obtain the following optimization problem, which gives us a decision boundary with the largest margins:

$$
\begin{aligned}
\max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\
\text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,
\end{aligned}
$$

- Where $w = \sum_{i=1}^{m} a_i y^{(i)} x^{(i)}$
  - The prediction $w^T x + b$ can also be written as

$$
\begin{aligned}
w^T x + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\
&= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b.
\end{aligned}
$$

- Where post of the inner products will be zero except for the support vectors

8.6 Kernels

- Given a feature mapping ϕ we define the corresponding **Kernel** to be

$$K(x, z) = \phi(x)^T \phi(z) \tag{68}$$

- Everywhere we previously had $\langle x, z \rangle$ we could simple replace it with $K(x, z)$ and the algorithm would now be learning using the features ϕ
- The **Gaussian kernel** which corresponds to an infinite dimensional feature mapping ϕ

$$K(x, z) = \exp(-\frac{||x - z||^2}{2\sigma^2}) \tag{69}$$

- The matrix called the **Kernel matrix** is defined from some m data points $\{x^{(1)}, \ldots, x^{(m)}\}$ as the m-by-m matrix K where the (i, j) entry is given by $K_{ij} = K(x^{(i)}, y^{(i)})$
- **Theorem (Mercer).** Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, \ldots, x^{(m)}\}$, $m < \infty$, the corresponding kernel matrix is symmetric positive semi-definite

8.7 Regularization and the non-separable case

- To make the algorithm work for non-linearly separable datasets and being less sensitive to outliers, the optimization can be reformulated as follows using regularization:

$$
\begin{aligned}
\min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\
\text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\
& \xi_i \geq 0, \quad i = 1, \ldots, m.
\end{aligned}
$$

- Which means that examples are now permitted to have a functional margin less than 1
- By using some of the KKT conditions one can obtain the following dual form of problem

$$
\begin{aligned}
\max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle \\
\text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m}\alpha_i y^{(i)} = 0,
\end{aligned}
$$

8.8 The SMO algorithm

8.8.1 General

- The SMO algorithm gives an efficient way of solving the dual problem arising from the derivation of the SVM

8.8.2 Coordinate ascent

- If one is trying to solve the unconstrained optimization problem

$$
\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m) \tag{70}
$$

- One can use the algorithm called **coordinate ascent:**

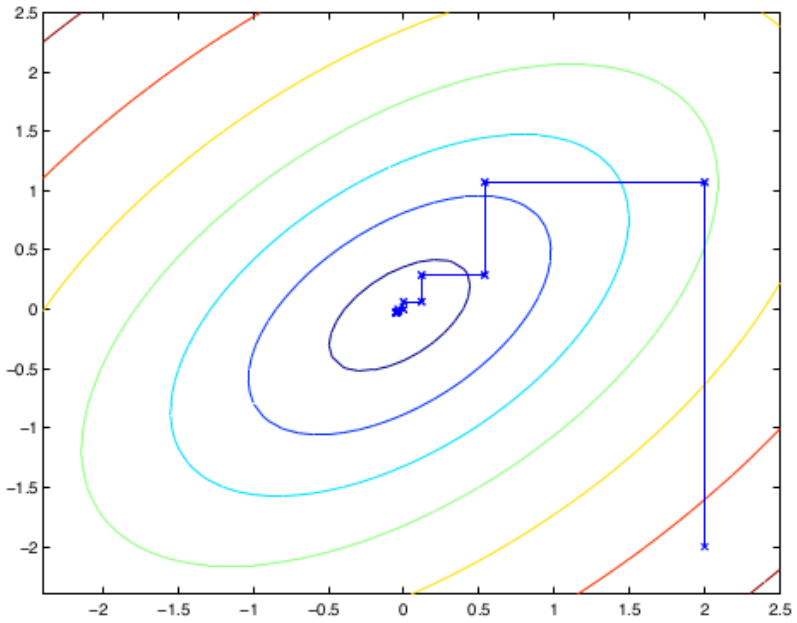


Figure 4: Coordinate ascent example

Loop until convergence: {

For $i = 1, \ldots, m$, {

$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$.

}

}

- Where one holds all the variables fixed except some a_i

8.8.3 SMO

- The SMO algorithm does the following

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

- To test for convergence, one can test whether the KKT conditions are satisfied within some *tol*
  - The tol is the convergence tolerance parameter normally set around 0.01 to 0.001

9 Deep Feedforward Networks

9.1 General

- **Deep feedforward networks** are quintessential deep learning models
  - The goal of a feedforward network is to approximate some function f^*
  - It defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation

- It is called **feedforward** since information flows through the function being evaluated from $\boldsymbol{x}$, through intermediate computations used to define f and finally to the output $\boldsymbol{y}$
  - When feedforward neural networks are extended to include feedback connections, they are called **recurrent neural networks**
- They are called **networks** because they typically are represented by composing together many different function
  - It is associated with a DAG describing ow the functions are composed together
  - The different functions are called **layers**
  - The overall length of the function chain gives the **depth** of the model
  - The final layer is called the **output layer**
  - During the training we drive $f(\boldsymbol{x})$ to match $f^*(\boldsymbol{x})$
  - The training data provides us with noisy, approximate examples of $f^*(\boldsymbol{x})$ evaluated at different training points
  - The training data does not say what each individual layers should do
    - That is the training algorithms job
    - They are called **hidden layers**
  - The dimensionality of these hidden layers determines the **width** of the model
- The strategy of deep learning is the feature transform ϕ
  - We have a model $= f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}, \boldsymbol{\theta})^\top \boldsymbol{w}$
  - We have the parameters $\boldsymbol{\theta}$ that we use to learn ϕ from a broad class of functions
  - We have the parameters $\boldsymbol{w}$ that map from $\phi(\boldsymbol{x})$ to the desired output
- Training a feedforward network requires making many of the same design decisions as are necessary for a linear model:
  - Choosing the optimizer
  - The cost function
  - The form of the output units.

9.2 Gradient-Based Learning

9.2.1 General

- The non-linearity of a neural network causes most interesting loss functions to become non-convex
  - It means that NNs are usually are trained by using iterative, gradient-based optimizes that merely drive the cost function to a very low value
  - It is important to initialize all weights to random values becomes of the error function being non-convex, when using stochastic gradient descent
    - The biases may be initialized to zero or a small positive values

9.2.2 Cost Functions

1. General
   - Cost functions for neural network are more or less the same as those for other models, such as linear models
     - The total cost function used to train a neural network will often combine one of the primary cost functions with a regularization term
2. Learning Conditional Distributions with Maximum Likelihood
   - Most NNs are trained using maximum likelihood
     - The cost function is simply the NLL which is equivalently described as the cross-entropy between the training data and the model distribution
     - This cost function is given by:

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}).$$

   - The advantage of deriving the cost function from maximum likelihood is that it removes the burden of designing cost functions for each model
     - Specifying a model $p(\boldsymbol{y} \mid \boldsymbol{x})$ automatically determines a cost function $\log p(\boldsymbol{y} \mid \boldsymbol{x})$

- Instead of learning a full probability distribution $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$, we often want to learn just one conditional statistic of $\boldsymbol{y}$ given $\boldsymbol{x}$
  - Such as predicting the mean of $\boldsymbol{y}$ given the predictor $f(\boldsymbol{x}; \boldsymbol{\theta})$

3. Learning Conditional Statistics

   - Instead of learning a full probability distribution $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ one often want to learn just one condition statistic of $\boldsymbol{y}$ given $\boldsymbol{x}$
     - Such as predicting the mean of $\boldsymbol{y}$
     - The cost function can be viewed as being a **functional** rather than just a function
       - A mapping from functions to real numbers
     - The cost functional can be designed to have its minimum occur at some specific function we desire
   - Solving the optimization problem

$$f^* = \arg\min_f \mathbb{E}_{\boldsymbol{x},\boldsymbol{y} \sim p_{data}} ||\boldsymbol{y} - f(\boldsymbol{x})||^2 \tag{71}$$

   yields

$$f^*(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{y} \sim p_{data}(\boldsymbol{y}|x)}[\boldsymbol{y}] \tag{72}$$

   - The following function yields a function that predicts the *median* value of $\boldsymbol{y}$ for each $\boldsymbol{x}$

$$f^* = \arg\min_f \mathbb{E}_{\boldsymbol{x},\boldsymbol{y} \sim p_{data}} ||\boldsymbol{y} - f(\boldsymbol{x})||_1 \tag{73}$$

   - This cost function is commonly call **mean absolute error**

9.2.3 Output Units

1. General

   - The choice of cost function is tightly coupled with the choice of output unit
     - Most of the time one simply uses the cross-entropy between the data distribution and the model distribution
     - The choice of how to represent the output determines the cross-entropy function
     - Any kind of neural network that may be used as output can also be used as a hidden unit

- The hidden features is defined by $\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta})$
  - The role of the output layer is to provide some additional transformation from the features to complete the task that the network must perform

2. Linear Units for Gaussian Output Distributions

   - Linear units is an output unit based on an affine transformation with no non-linearity
     - Given features $\boldsymbol{h}$, a layer of linear output units produces a vector $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + b$
     - Linear output layers are often used to produce the mean of a conditional Gaussian distribution

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I}) \tag{74}$$

   - Maximizing the log-likelihood is the equivalent to minimizing the mean squared error

9.3 Hidden Units

9.3.1 General

- Some valid hidden units are not differentiable at all input points
  - Such as $g(z) = \max\{0, z\}$
  - Solved by using the right or left differential

9.3.2 Rectied Linear Units and Their Generalizations

- Rectified linear units use the activation function $g(z) = \max\{0, z\}$
  - Easy to optimize because they are similar to linear units
    - Only difference is that a rectified unit outputs zero across half its domain

- Rectified linear units are typically used on top of an affine transformation

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}) \tag{75}$$

- When initializing the parameters of the affine transformation it can be a good pratise to set all elements of $\boldsymbol{b}$ to a small positive value

- A drawback to rectified linear units is that they cannot learn via gradient based methods on examples for which their activation is zero
  - Some generalizations of rectified linear units guarantee that they receive gradient everywhere
- **Maxout units** generalize rectified linear units further
  - Instead of applying an element-wise function $g(z)$ maxout units divide $\boldsymbol{z}$ into groups of k values
  - Each maxout unit the outputs the maximum element of one of these groups

$$g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(u)}} z_j \tag{76}$$

- where $\mathbb{G}^{(I)}$ is the set of indices into the inputs for group i
- A maxout unit can be seen as learning the activation function itself rather than just the relationship between units

9.3.3 Logistic Sigmoid and Hyperbolic Tangent

- The widespread saturation of sigmoid units can make gradient-based learning very difficult
  - Their use as hidden units in feedforward networks are discouraged
  - Training using the *tanh* function for hidden layers are easier

9.4 Architecture Design

9.4.1 General

- **Architecture** refers to the overall structure of the network
  - How many units it should have
  - How these units should be connected to each other
- Most NNs are organized into groups of units called layers
  - They are often arranged in a chain structure where each layer is a function of the layer that preceded it
  - The ith layer is given by

$$h^{(i)} = g^{(i)}(\boldsymbol{W}^{(i)T}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i)}) \tag{77}$$

- where the first layer uses x instead of $h^{(i)}$
- In chain based architectures, the main architectural considerations are to choose the the depth of the network and the width of each layer
  - The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error.

9.4.2 Universal Approximation Properties and Depth

- The universal approximation theorem means that regardless of what function we are trying to learn, we know that a large multi layered perceptron will be able to represent this function.
  - However, we are not guaranteed that the training algorithm will be able to learn that function
  - Learning it can fail for two different reasons
    1. The optimization algorithm used for training may not be able to find the value of the parameters corresponds to the desired function
    2. The training algorithm might choose the wrong function due to overfitting
- Feedforward networks provide a universal system for representing functions, in the sense that, given a function, there exists a feedforward network that approximates the function.
  - There is no universal procedure for examining a training set of specific examplesd and choosing a function that will generalize to point not in the training set
- The universal approximation theorem says that there exists a network large enough to achieve any degree of accuracy we desire, but the theorem does not say how large this network will be.
- A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
  - In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error

- The number of linear regions carved out by a deep rectifier network with d inputs, depth l, and n units per hidden layer is

$$O\left(\binom{n}{d}^{d(l-1)} n^d\right),$$

- Choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions.

9.4.3 Other Architectural Considerations

- The layers need not be connected in a chain, but it is the most common practice.
  - Many architectures build a main chain but then add extra architectural features to it
    - Such as skip connections going from layer i to layer i + 2 or higher.
  - These skip connections make it easier for the gradient to ow from output layers to layers nearer the input.
- A key consideration of architecture design is how to connect a pair of layers to each other
  - The default way is having every input unit connected to every output unit
  - Strategies for reducing the number of connections reduce the number of parameters and the amount of computation required to evaluate the network,
    - They are often highly problem-dependent.

9.5 Back-Propagation and Other Differentiation Algorithms

9.5.1 General

- **Forward propagation** is when the inputs $\boldsymbol{x}$ provide initial information that then propaget up to the hidden units at each layer and finally produces an output $\hat{\boldsymbol{y}}$

- During training it can continue onward until it produces a scalar cost $J(\boldsymbol{\theta})$

- The **back-propagation** algorithm (**backprop**) allows the information from the cost to flow backwards through the network in order to compute the gradient
  - It is used to compute the gradient, another algorithm is used to do the learning e.g. stochastic gradient descent

9.5.2 Computational Graphs

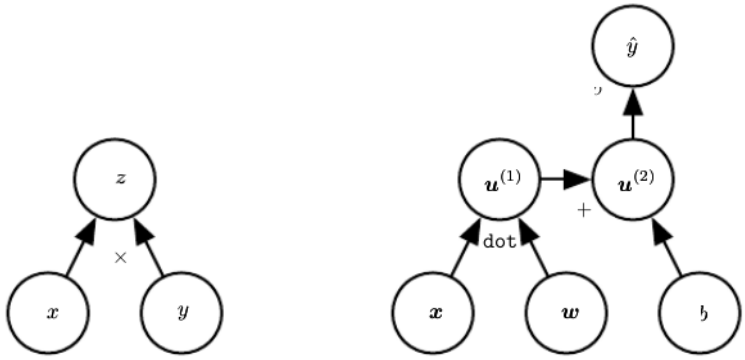


Figure 5: Computational Graph Examples

- An **operation** is a simple function of one or more variables
  - Defined to return only a single output variable

- The graph language is accompanied by a set of allowable operations
  - If a variables y is computed by applying an operation to a variable x, then there is drawn a directed edge from x to y
  - The output node is sometimes annotated with the name of the operation applied

9.5.3 Chain Rule of Calculus

- Back-propagation computes the chain rule, with a specific order of operations that is highly efficient

- The chain rule can generalize beyond the scalar case suppose that $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$, g maps from $\mathbb{R}^m$ to $\mathbb{R}^n$, and f maps from $\mathbb{R}^n$ to R. If $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.$$

- In vector notation, this may equivalently be written as

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right)^\top \nabla_{\boldsymbol{y}} z,$$

- where $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ Jacobian matrix of g
- The back-propagation algorithm consists of performing Jacobian-gradient product given by the chain rule for each operation in the graph
- The back-propagation algorithm is typically applied to tensors of arbitrary dimensionality
  - Is exactly the same as back-propagation with vector conceptually
  - Denoting a gradient of a value z with respoect to a tensor \$s.
- To denote the gradient of a value z with respect to a tensor $\boldsymbol{X}$, we write $\nabla_{\boldsymbol{X}} z$
  - The indices into $\boldsymbol{X}$ have multiple coordinates
- The chain rule for tensors:

can write the chain rule as it applies to tensors. If $\mathbf{Y} = g(\mathbf{X})$ and $z = f(\mathbf{Y})$, then

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}. \tag{6.47}$$

9.5.4 Recursively Applying the Chain Rule to Obtain Backprop

- Given a scalar $u^{(n)}$ which is the quantity whose gradient we want to obtain with respect to all the n_i input nodes $u^{(1)}$ to $u^{(n_i)}$
  - We wish to compute $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ for $i \in \{1, 2, \ldots, n_i\}$
  - In backprop $u^{(n)}$ will be the cost associated with an example or a minibatch
  - $u^{(1)}$ to $u^{(n_i)}$ correspond to the parameters of the model
  - The nodes of the graph is assumed to be order in such a way that we can compute their output one after the other
  - Each node $u^{(i)}$ is associated with an operation $f^{(i)}$ and is computed by evaluating the function

$$u^{(i)} = f(\mathbb{A}^{(i)}) \tag{78}$$

- where $\mathbb{A}^{(i)}$ is the set of all nodes that are parents of $u^{(i)}$

Algorithm 6.1 A procedure that performs the computations mapping n_i inputs $u^{(1)}$ to $u^{(n_i)}$ to an output $u^{(n)}$. This defines a computational graph where each node computes numerical value $u^{(i)}$ by applying a function $f^{(i)}$ to the set of arguments $\mathbb{A}^{(i)}$ that comprises the values of previous nodes $u^{(j)}$, $j < i$, with $j \in Pa(u^{(i)})$. The input to the computational graph is the vector $\boldsymbol{x}$, and is set into the first n_i nodes $u^{(1)}$ to $u^{(n_i)}$. The output of the computational graph is read off the last (output) node $u^{(n)}$.

for $i = 1, \ldots, n_i$ **do**
  $u^{(i)} \leftarrow x_i$
end for
for $i = n_i + 1, \ldots, n$ **do**
  $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$
  $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$
end for
return $u^{(n)}$

- The forward propagation computation is put in a graph $\mathcal{G}$
  - In order to perform back-propagation, one can constructs a computational graph that depends on $\mathcal{G}$ and add to it an extra set of nodes
    - These form a subgraph $\mathcal{B}$ with one node per node of $\mathcal{G}$
    - Computation in $\mathcal{B}$ proceeds in the reverse of the order of computation in $\mathcal{G}$

    - Each node of $\mathcal{B}$ computes the derivative $\frac{\partial u^{(n)}}{\partial u^{i}}$ associated with the forward graph node $u^{(i)}$ using the chain rule
  - The subgraph $\mathcal{B}$ contains one edge for each edge for each edge of $\mathcal{G}$
    - The edge from $u^{(j)}$ to $u^{(i)}$ is associated with the computation of $\frac{\partial u^{(i)}}{\partial u^{(j)}}$
    - The dot product is performed for each node between the gradient already computed with respect to nodes $u^{(i)}$ that are children of $u^{(j)}$ and the vector containing the partial derivatives $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ for the same children nodes

- The amount of computation required for performing back-propagration scales linearly with the number of edges in $\mathcal{G}$

Algorithm 6.2 Simplified version of the back-propagation algorithm for computing the derivatives of $u^{(n)}$ with respect to the variables in the graph. This example is intended to further understanding by showing a simplified case where all variables are scalars, and we wish to compute the derivatives with respect to $u^{(1)}, \ldots, u^{(n_i)}$. This simplified version computes the derivatives of all nodes in the graph. The computational cost of this algorithm is proportional to the number of edges in the graph, assuming that the partial derivative associated with each edge requires a constant time. This is of the same order as the number of computations for the forward propagation. Each $\frac{\partial u^{(i)}}{\partial u^{(j)}}$ is a function of the parents $u^{(j)}$ of $u^{(i)}$, thus linking the nodes of the forward graph to those added for the back-propagation graph.

Run forward propagation (algorithm 6.1 for this example) to obtain the activations of the network
Initialize `grad_table`, a data structure that will store the derivatives that have been computed. The entry `grad_table`$[u^{(i)}]$ will store the computed value of $\frac{\partial u^{(n)}}{\partial u^{(i)}}$.
`grad_table`$[u^{(n)}] \leftarrow 1$
for $j = n - 1$ down to 1 **do**
  The next line computes $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$ using stored values:
  `grad_table`$[u^{(j)}] \leftarrow \sum_{i:j \in Pa(u^{(i)})}$ `grad_table`$[u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$
end for
return $\{$`grad_table`$[u^{(i)}] \mid i = 1, \ldots, n_i\}$

9.5.5 Back-Propagation Computation in Fully-Connected MLP

Algorithm 6.3 Forward propagation through a typical deep neural network and the computation of the cost function. The loss $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ depends on the output $\hat{\boldsymbol{y}}$ and on the target $\boldsymbol{y}$ (see section 6.2.1.1 for examples of loss functions). To obtain the total cost J, the loss may be added to a regularizer $\Omega(\theta)$, where θ contains all the parameters (weights and biases). Algorithm 6.4 shows how to compute gradients of J with respect to parameters $\boldsymbol{W}$ and $\boldsymbol{b}$. For simplicity, this demonstration uses only a single input example $\boldsymbol{x}$. Practical applications should use a minibatch. See section 6.5.7 for a more realistic demonstration.

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \ldots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \ldots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output
  $\boldsymbol{h}^{(0)} = \boldsymbol{x}$
  for $k = 1, \ldots, l$ **do**
    $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
    $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
  end for
  $\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
  $J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

Algorithm 6.4 Backward computation for the deep neural network of algorithm 6.3, which uses in addition to the input $\boldsymbol{x}$ a target $\boldsymbol{y}$. This computation yields the gradients on the activations $\boldsymbol{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a stochastic gradient update (performing the update right after the gradients have been computed) or used with other gradient-based optimization methods.

  After the forward computation, compute the gradient on the output layer:
  $\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
  for $k = l, l-1, \ldots, 1$ **do**
    Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
    Compute gradients on weights and biases (including the regularization term, where needed):
    $\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
    $\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
    Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$
  end for

9.5.6 Symbol-to-Symbol Derivatives

- **Symbolic Representations** is algebraic expressions and computational graphs that both operate on symbols, or variables that do not have specic values.
- Some approaches to back-propagation take a computational graph and a set of numerical values for the inputs to the graph, then return a set of numerical values describing the gradient at those input values.
  - This is called "symbol-to-number" dierentiation
- Another approach for backprop is to take a computational graph and add additional nodes to the graph that provide a symbolic description of the desired derivativess

9.5.7 General Back-Propagation

- Each node in the graph $\mathcal{G}$ corresponds to a variable
  - This is described as being a tensor V
  - Tensor can in general have any number of dimensions
  - They subsume scalars, vectors, and matrices
- It is assumed that each variable V is associated with the following subroutines:
  - `get_operation`(V): This returns the operation that computes V, represented by the edges coming into V in the computational graph. For example, there may be a Python or C++ class representing the matrix multiplication operation, and the `get_operation` function. Suppose we have a variable that is created by matrix multiplication, $\boldsymbol{C} = \boldsymbol{AB}$. Then `get_operation`($\mathsf{V}$) returns a pointer to an instance of the corresponding C++ class.
  - `get_consumers`($\mathsf{V}, \mathcal{G}$): This returns the list of variables that are children of V in the computational graph $\mathcal{G}$.
  - `get_inputs`($\mathsf{V}, \mathcal{G}$): This returns the list of variables that are parents of V in the computational graph $\mathcal{G}$.
- Each operation `op` is also associated with a `bprop` operation
  - This `bprop` operation can compute a Jacobian vector product
  - Formally, `op.bprop(inputs,X,G)` must return:

$$\sum_i \left(\nabla_{\mathsf{X}} \mathtt{op.f}(\mathtt{inputs})_i\right) \mathsf{G}_i,$$

- Inputs is a list of inputs that are supplied to the operation
- `op.f` is the mathematical function that the operation implements
- X is the input whose gradient we which to compute
- G is the gradient on the output of the operation

Algorithm 6.5 The outermost skeleton of the back-propagation algorithm. This portion does simple setup and cleanup work. Most of the important work happens in the `build_grad` subroutine of algorithm 6.6

Require: $\mathbb{T}$, the target set of variables whose gradients must be computed.
Require: $\mathcal{G}$, the computational graph
Require: z, the variable to be differentiated
Let $\mathcal{G}'$ be $\mathcal{G}$ pruned to contain only nodes that are ancestors of z and descendents of nodes in $\mathbb{T}$.
Initialize `grad_table`, a data structure associating tensors to their gradients
`grad_table`$[z] \leftarrow 1$
for V in $\mathbb{T}$ **do**
  `build_grad`$(\mathbf{V}, \mathcal{G}, \mathcal{G}', \texttt{grad\_table})$
end for
Return `grad_table` restricted to $\mathbb{T}$

Algorithm 6.6 The inner loop subroutine `build_grad`$(\mathbf{V}, \mathcal{G}, \mathcal{G}', \texttt{grad\_table})$ of the back-propagation algorithm, called by the back-propagation algorithm defined in algorithm 6.5.

Require: $\mathbf{V}$, the variable whose gradient should be added to $\mathcal{G}$ and `grad_table`.
Require: $\mathcal{G}$, the graph to modify.
Require: $\mathcal{G}'$, the restriction of $\mathcal{G}$ to nodes that participate in the gradient.
Require: `grad_table`, a data structure mapping nodes to their gradients
  if $\mathbf{V}$ is in `grad_table` **then**
    Return `grad_table`$[\mathbf{V}]$
  end if
  $i \leftarrow 1$
  for $\mathbf{C}$ in `get_consumers`$(\mathbf{V}, \mathcal{G}')$ **do**
    `op` $\leftarrow$ `get_operation`$(\mathbf{C})$
    $\mathbf{D} \leftarrow$ `build_grad`$(\mathbf{C}, \mathcal{G}, \mathcal{G}', \texttt{grad\_table})$
    $\mathbf{G}^{(i)} \leftarrow$ `op.bprop`(`get_inputs`$(\mathbf{C}, \mathcal{G}'), \mathbf{V}, \mathbf{D}$)
    $i \leftarrow i + 1$
  end for
  $\mathbf{G} \leftarrow \sum_i \mathbf{G}^{(i)}$
  `grad_table`$[\mathbf{V}] = \mathbf{G}$
  Insert $\mathbf{G}$ and the operations creating it into $\mathcal{G}$
  Return $\mathbf{G}$

- The backprop algorithm uses dynamic programming to get a better running time

9.6 Backpropagation equations

- w^l_{ji} is the weight from neuron i in layer $l-1$ to neuron j in layer l
- $a^1_j = x_j$
- $s^l_j = \sum_i a^{l-1}_i w^{l-1}_{j,i} + b^{l-1}_j$
- $a^l_k = \Phi(s^l_k)$
- $\delta^l_j = \frac{\partial e}{\partial s^l_j}$

$$\delta_j^L = \frac{\partial e}{\partial \mathrm{nn}_{W,B}} \frac{\partial \Phi(s_j^L)}{\partial s_j^L}$$

$$\delta_j^l = \sum_{i=1}^{d_l+1} \delta_i^{l+1} w_{j,i}^l \frac{\partial \Phi(s_j^l)}{\partial s_j^l}$$

$$\frac{\partial e}{\partial w_{j,i}^{l-1}} = a_i^{l-1}\, \delta_j^l$$

$$\frac{\partial e}{\partial b_j^{l-1}} = \delta_j^l$$

All values computable by one forward And one backward pass

Classic Backpropagation Algorithm

Initialize weights $w^l_{i,j}, b^l_j$ randomly (important)

$$a_j^1 = x_j$$

$$l = 2 \to L$$

$$s_j^l = \sum_i a_i^{l-1} w_{j,i}^{l-1} + b_j^{l-1}$$

$$a_j^l = \Phi(s_j^l)$$

Forward Pass:

$$\delta_j^L = \frac{\partial e}{\partial s_j^L} = \frac{\partial e}{\partial \mathrm{nn}_{W,B}} \frac{\partial \mathrm{nn}_{W,B}}{\partial s_j^L}$$

$$l = L - 1 \to 1$$

$$\delta_j^l = \sum_{i=1}^{d_l+1} \delta_i^{l+1} w_{j,i}^l \frac{\partial \Phi(s_j^l)}{\partial s_j^l}$$

Backward Pass

$$\frac{\partial e}{\partial w_{j,i}^{l-1}} = a_i^{l-1}\, \delta_j^l \qquad \frac{\partial e}{\partial b_j^{l-1}} = \delta_j^l$$

Output

10 Convolutional Networks

10.1 General

- **Convolutional Networks** (CNNs) are a specialized kind of neural network for processing data that has a grid-like topology
  - **Convolution** is a specialized kind of linear operation
  - Convolutional networks are neural networks that use convolution in place of general matrix multiplication in at least one of their layers

10.2 The Convolution Operation

- A convolution is in its most general form an operation on two functions of a real valued argument
  - Example of a convolution: $s(t) \int x(a)w(t-a)da$
  - The convolution operation is typically denoted with an asterisk: $s(t) = (x * w)(t)$
  - The first argument to the convolution is often referred to as the **input**
  - The second argument is referred to as the **kernel**
  - The output is referred to as the **feature map**
  - In machine learning applications
    - The input is usually a multidimensional array of data
    - The kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm.
    - The multidimensional dimensional arrays are referred to as tensors
    - Because each element of the input and kernel must be explicitly stored separately, it is often assumed that these functions are zero everywhere but in the finite set of points for which we store the value
  - Convolutions are often used over more than one axis at a time
    - e.g. on a two-dimensional image I as input one would probably use a two dimensional kernel K
  - Convolution is commutative which means that for a two dimensional kernel K and a input I: $S(i,j) = (I*K)(i,j) = (K*I)(i,j)$

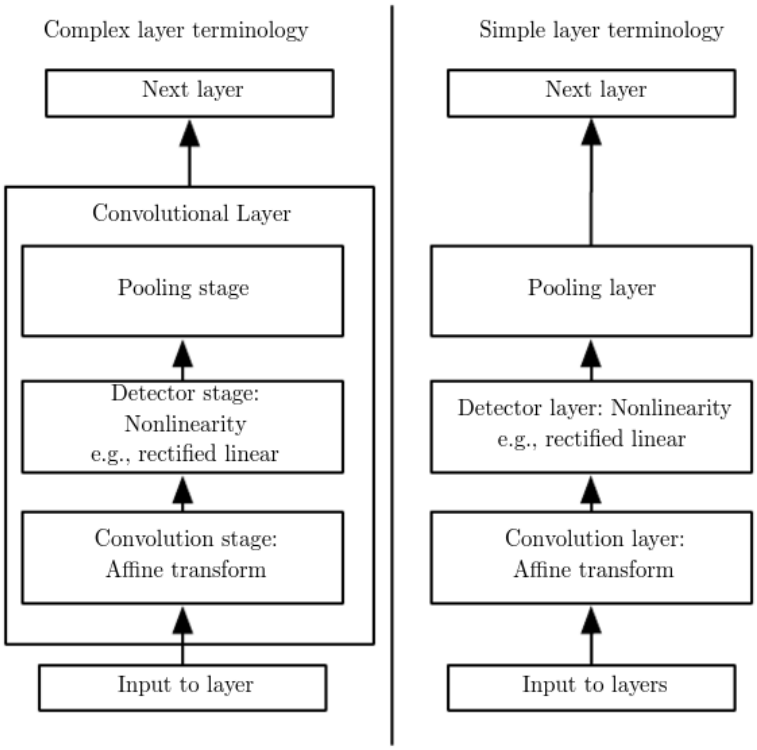
    * The last one is usually more straightforward to implement in a machine learning library, since there is less variation in the range of valid values for m and n
  - Many neural network libraries implement are related function called **cross-correlation**, which is the same as convolution but without flipping the kernel
    * e.g. $S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$
    * Some also call this convolution

10.3 Motivation

- Convolution leverages three important ideas that can help improve a machine learning system:
  - **Sparse interactions** is accomplished by making the kernel smaller than the input
    * It is also referred to as **parse connectivity or *sparse weights**
    * e.g. one can detect small meaningful features in images such as edges with kernels
    * One needs to store fewer parameters which reduces the memory requirements of the model and improves its statistical efficiency
      · Computing the output requires fewer operations
      · The improvements in efficiency are usually quite large
  - **Parameter sharing:** refers to using the same parameter for more than one function in the model
    * One can say that a network has **tied weights**, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere
    * Each member of the kernel is used at every position of the input
  - **Equivariant representations**: If g is any function that translates the input then that is shifts is, then the convolution function is equivalent to g
    * A function f is equivalent to a function g if $f(g(x)) = g(f(x))$

- Some kinds of data cannot be processed by neural networks dened bymatrix multiplication with a xed-shape matrix. Convolution enables processing of some of these kinds of data.

10.4 Pooling



- A typical layer of a convolutional network consists of three stages
  1. In the first stage the layer performs several convolutions in parallel to produce a set of linear activations
  2. In the second stage each linear activation is run through a non-linear activation, such as rectified linear activation function
     - Is sometimes called the **detector stage**
  3. The third stage we use a **pooling function** to modify the output of the layer further
     - It replaces the output of the net at a certain location with a summary statistic of the nearby output

    - e.g. the **max pooling operation** which reports the maximum output within a rectangular neighborhood
  - Pooling helps to make the representation approximately **invariant** to small translation of the input
    - Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs
    - Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is.
  - It is possible to use fewer pooling units than detector units by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixels apart
    - Improves computational efficiency of the network because the next layer has approximately k times fewer inputs to process
    - Can be used to handle images of variable size by changing how much it is space depending on the input size

11 Tree-Based Methods

11.1 Background

- Three based methods partition the feature space into a set of rectables and then fit a simple model in each one
  - They are simple yet powerful
  - One first split the space into two regions and models the response by the mean of Y in each region, then one or both of the regions are split into two more regions, this process is continued until some stopping rule is applied

11.2 Regression Trees

- The data consists of p inputs and a response, for each of N observations, that is (x_i, y_i) for $i = 1, 2, \ldots, N$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$
- The algorithm needs to automatically decide on the splitting variables and split points and what topology (shape) the three should have

- If one have a partition into M regions $R_1, R_2, \ldots, R_M$, and model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{79}$$

- If the criterion is minimization of the sum of squares $\sum(y_i - f(x_i))^2$, he best $\hat{c}_m$ is just the average of y_i in the region R_m:

$$\hat{c}_m = \text{ave}(y_i \mid x_i \in R_m). \tag{80}$$

- Since the binary partition in terms of minimum sum of squares is generally computationally infeasible one needs a greedy algorithm:

algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}. \tag{9.12}$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,\,s}\Big[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2\Big]. \tag{9.13}$$

For any choice j and s, the inner minimization is solved by

$$\hat{c}_1 = \text{ave}(y_i|x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = \text{ave}(y_i|x_i \in R_2(j,s)). \tag{9.14}$$

- For each splitting variable, the determination of the split point s can be done very quickly by scanning through all the inputs
  - Having found the best split, one partition the data into the two resulting regions and repeat the splitting process on each of the two regions, which is the repeat on all the resulting regions
  - The optimal tree size should be adaptively chosen from the data
    - A preferred strategy is to grow a large T_0 stopping the splitting process only when some minimum node size is reached
    - The large tree is then pruned using cost-complexity pruning
- A **subtree** $T \subset T_0$ is defined to be any tree that can be obtained by **pruning** T_0
  - Pruning is collapsing any number of its internal (non-terminal nodes).

- Terminal nodes is indexed by m, with node m representing region R_m
- $|T|$ denotes the number of terminal nodes in T

- Letting

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{9.15}$$

we define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|. \tag{9.16}$$

- The idea is to find for each α the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$
  - The tuning parameter $\alpha \geq 0$ governs the tradeof between tree size and its goodness of fit to the data
  - Larger values of α result in smaller trees T_α and the converse for smaller values of α
  - With $\alpha = 0$ the solution is the full tree T_0
  - For each α there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$

- If one want to find T_α one uses **weakest link pruning**:
  - One successively collapse the internal node that produces the smallest per-node increase in $\sum_m N_m Q_m(T)$ until we produce P and continue until we produce the single-node (root) tree.
  - This gives a finite sequence of subtrees and T_α must be one of these subtrees
  - Estimation of α is achieved by five-fold or ten-fold cross-validation:
    - The value $\hat{\alpha}$ is chosen to minimize the cross-validated sum of squares
    - The final tree is $T_{\hat{\alpha}}$

11.3 Classification Trees

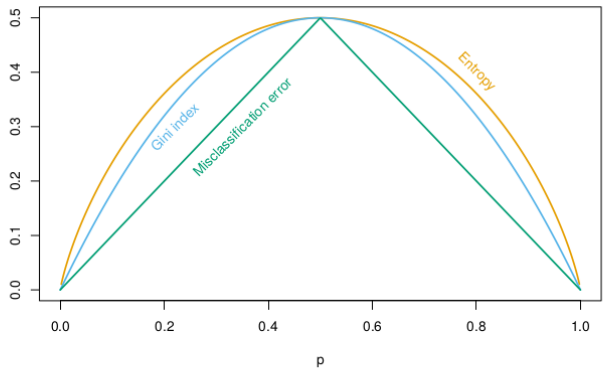


FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

- If the target is classification outcome taking values $1, 2, \ldots, K$ the only changes needed is the tree algorithm is the criteria for splitting nodes and pruning the tree
  - In a nodes m, representing a region R_m with N_m observations, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

the proportion of class k observations in node m. We classify the observations in node m to class $k(m) = \arg\max_k \hat{p}_{mk}$, the majority class in node m. Different measures $Q_m(T)$ of node impurity include the following:

Misclassification error:	$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$.
Gini index:	$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$.
Cross-entropy or deviance:	$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.

- The Gini index and cross-entropy are differentiable and are therefore more amenable to numerical optimization
  - They are often used for growing the three
  - To guide the cost-complexity pruning any of the three measures can be used but typically it is the misclassification rate

11.4 Other Issues

- **Categorial Predictors**
  - When splitting a predictor having q possible unordered values, there are $2^{q-1} - 1$ possible partitions of the q values into two groups
    - The computations become prohibitive for large q
  - With a $0-1$ outcome the computation simplifies
    - One orders the predictor classes according to the proportion falling in outcome class 1
    - This gives the optimal split in terms of cross-entropy or Gini index- among all possible $2^{q-1} - 1$ splits
- **The Loss Matrix**: A KxK loss matrix $\boldsymbol{L}$, is defined with $L_{kk'}$ being the loss incurred for classifying a class k observation as class k'
  - Typically no loss is incurred for correct classifications, that is $L_{kk} = 0 \; \forall k$
  - To incorporate P the losses into the modeling process, one could modify the Gini index to $\sum_{k \neq k'} L_{kk'} \hat{p}_{mk} \hat{p}_{mk'}$
  - This does not help in the two-class case and a better approch is to weight the observations in class k by $L_{kk'}$
- **Missing Predictor Values:** If some of the data has some missing predictor values in some of the values
  - There are two better approaches than throwing the data away
    1. The first is applicable to variable predictors, where one simply makes a new category for *"missing"*
       - This might make one discover that the observation with missing values for some measurement behave differently that those with nonmissing values
    2. The second is a more general approach which is the construction of surrogate variables
       - When considering a predictor for a split, we use only the observations for which that predictor is not missing
       - Having chosen the best (primary) predictor and split point, we form a list of surrogate predictors and split points.

· The first surrogate is the predictor and corresponding split point that best mimics the split of the training data achieved by the primary split.
· The second surrogate is the predictor and corresponding split point that does second best, and so on.

- **Why Binary Splits?**
  - The problem with using multiway splits is that it fragment the data too quickly, leaving insufficient data at the next level down.
  - Since multiway splits can be achieved by a series of binary splits, the binary splits are preferred.

12 Random forests

12.1 Introduction

- **Bagging** or **bootstrap aggregation** is a technique for reducing the variance of an estimated prediction function
  - Bagging works seems to work especially well for high-variance, low-ias procedures such as trees
  - For regression we simply fit the same regression tree many time to bootstrapsampled versions of the training data and average the result
  - For classification, a **committee** of trees each cast a vote for the predicted class
- **Random forests** is a substantial modification of bagging that builds a large collection of de-correlated trees and averages them
  - On many problems its performance is very similar to boosting
  - They are simpler to train and tune than the boosting example

12.2 Bootstrap aggregating technique

- Given a standard training set D of size n
  - Bagging generates m new training set D_i each of size n' by sampling from D uniformly and with replacement
    - This is know as a **bootstrap** sample

- The m models are fitted using the bootstrap sample and combine by
  - Averaging the output (for regresion)
  - Voting (for classification)

12.3 Definition of Random Forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

- The essential idea in bagging is to average many noisy but approximately unbiased models
  - This reduces the variance
  - Trees are ideal candidates for bagging since:
    - They can capture complex interaction structures in the data
    - If grown sufficiently deep, they have relative low bias
  - Since trees are very noisy they benefit greatly from averaging
  - The bias of bagged trees is the same as that of the individual (bootstrap) trees
    - The only hope of improvement is through variance reduction
- An average of B independent identically distributed random variables, each with variance σ^2 has variance $\frac{1}{B}\sigma^2$

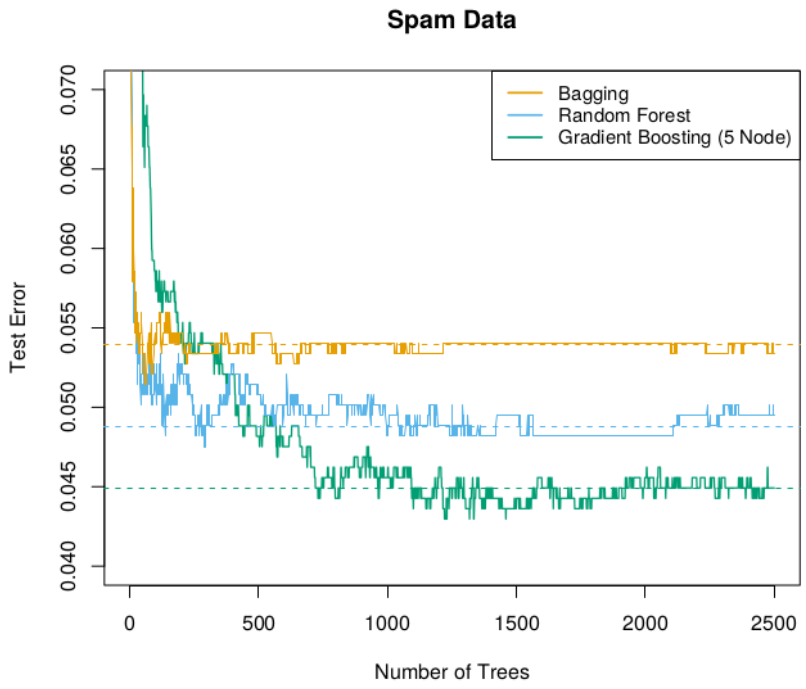
  - If the variables are simply identically distributed with positive correlation ρ the variance of the average is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{81}$$

- As B increases, the second terms disappears but the first remains
  - The size of the correlation of pairs of bagged trees limits the benefits of averaging
  - The idea in random forest is to improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance to much
    - This is achieved in the tree-growing process through random selection of the input variables
    - Typical values for m are $\sqrt{p}$ or even as low as 1
    - Reducing m will reduce the correlation between any pair of trees in the ensemble and therefore reduce the variance of the average
  - After B such trees $\{T(x;\Theta_b)\}_1^B$ are grown the random forest (regression) predictor is

$$\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b) \tag{82}$$

- Where Θ_b characterizes the bth random forest tree in terms of split variables, cutpoints at each nodes, and terminal-node values



13 Boosting and Additive Trees

13.1 Boosting Methods

- The motivation for **boosting** is to get a procedure that combines the outputs of many "eak" classifiers to produce a powerful "committee"
- One of the most popular algorithm called AdaBoost.M1 is as follows
  - Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$
  - Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$
    - The error rate on the training sample is

$$\bar{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)) \tag{83}$$

- and the expected error on future predictions is $E_{XY}(I \neq G(X))$

- A **weak classifier** is one whose error rate is only slightly better than random guessing

    1. The purpose of boosting is to sequentially apply the weak classification algorithm to modified versions of the data and produces a sequence of weak classifiers $G_m(x), m = 1, 2, \ldots, M$
    2. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \tag{84}$$

- $\alpha_1, \alpha_2, \ldots, \alpha_M$ are computed by the boosting algorithm and weight the contribution of each respective $G_m(x)$

    – This gives higher influence to the more accurate classifiers in the sequence

---

Algorithm 10.1 *AdaBoost.M1.*

---

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
    (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
    (b) Compute
$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

13.2 Boosting Fits an Additive Model

- The key in boosting $G(x)$ expression

    – It is a way of fitting an additive expansion in a set of elementary "basiss functions

    – The basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$

  - Basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{85}$$

- where β_m, $m = 1, 2, \ldots, M$ are the expansion coefficients and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions over the multivariate argument x characterized by a set of parameters γ
  - Typically these models are fit by minimizing a loss function averaged over the training data
  - For many loss functions $L(y, f(x)$ and/or basis functions $b(x; \gamma$ this requires computationally intensive numerical optimization techniques
  - A simple alternative can often be found when it is feasible to rapidly solve the subproblem of fitting just a single basis function

$$\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, (x_i; \gamma) \tag{86}$$

13.3 Forward Stagewise Additive Modeling

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

- **Forward stagewise modeling** approximates a solution by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added
  - At each iteration m one solves the optimal basis function $b(x; \gamma_m)$ and the corresponding coefficient β_m to add to the current expansion $f_{m-1}(x)$

- It produces $f_m(x)$ and the process is repeated

- For squared loss

$$L(y, f(x) = (y - f(x))^2 \tag{87}$$

- one has

$$\begin{aligned} L(y_i, f_{m-1}(x_1) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2 \end{aligned} \tag{88}$$

- where $r_i m = y_i - f_{m-1}(x_i)$ is simply the residual (difference) of the current modeland the ith observation

  - Therefore for squared-error loss, the term $\beta_m b(x; \gamma_m)$ that best fits the current residuals is added to the expansion at each step

13.4 Exponential Lost and AdaBoost

- The **exponential loss function** is the following

$$L(y, f(X)) = \exp(-yf(x)) \tag{89}$$

- The exponential loss is more sensitive to changes in the estimated class probabilities

- AdaBoost.M1 minimizes the exponential loss criterion via a forward-stagewise additive modeling approach

13.5 Why Exponential Loss?

- The principal attraction of exponential loss in the context of additive modeling is computational

  - It leads to the simple modular reweighting AdaBoost algorithm
  - It has the same population minimizer has binomial log likelihood which is true probabilities

13.6 Loss Functions and Robustness

13.6.1 Robust Loss Functions for Classification

- Since the squared-error is not a decreasing function of the margin it is not good for classification

  - Due to it penalizing large positive correct margins

- For K classification problems the logistic function generalizes nice

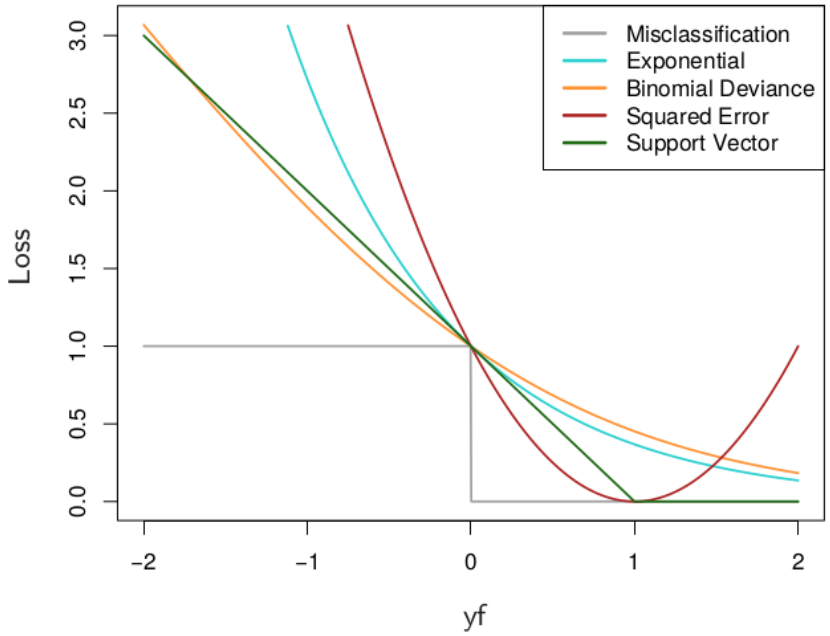
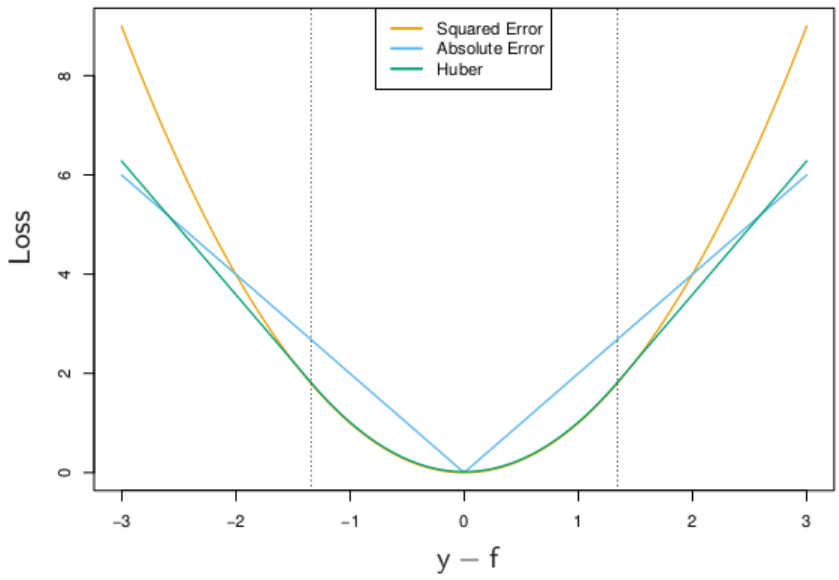


Figure 6: Loss functions for two-class classification

13.6.2 Robust Loss Functions for Regression



- In the regression setting, analogous to the relationship between exponential loss and binomial log-likelihood is the relationship between **squared-error loss** $L(y, f(x)) = (y - f(x))^2$ and **absolute loss** $L(y, f(x)) = |y - f(X)|$
  - The population solutions are
    - For squared-error loss: $f(x) = E(Y \mid x)$
    - For absolute loss: median$(Y \mid x)$
  - For symmetric error distributions the two population solutions are the same
  - On finite samples squared-error loss places much more emphasis on observations with large absolutie residual $|y_i - f(x_i)|$ during the fitting process
    - It is far less robust and it performance severely degrades for the fitting process for long-tailed error distributions and especially for grossly mismeasured y-values ("outliers").
    - Other more robust criteria such as absolute loss perform much better in these situations

- A criterion which is much more robust against outliers while being nearly as efficient as least squares is **Huber loss criterion** used for M-regression

$$L(y, f(x)) = \begin{cases} [y - f(x)^2] & \text{for } |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2 & \text{otherwise} \end{cases} \tag{90}$$

13.7 "Off-the-Shelf" Procedures for Data Mining

TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲*= good,* ◆*=fair, and* ▼*=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

- Of all the well-known learning methods, decision trees come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining
  - They are relatively fast to construct and they produce interpretable models (if the trees are small)
  - They have one aspect that prevents them from being the ideal tool forpredictive learning, namely inaccuracy.
    - They seldom provide predictive accuracy comparable to the best that can be achieved with the data at hand

13.8 Boosting Trees

- A tree can be formally expressed as

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_i I(x \in R_j) \tag{91}$$

- with parameters $\Theta = \{R_{j \cdot j}\}_1^J$
  - J is usually treated as a meta parameters
  - The parameters are found by minimizing the empirical risk

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \tag{92}$$

- The optimization problem is useful to divide into two parts
  - **Finding γ_j given R_j**: $\hat{\gamma} = \bar{y}_j$ is often used and for misclassification $\hat{\gamma}$ is the modal class of the observations falling in region R_j
  - **Finding R_j**: It is the difficult part, for which approximate solutions are found
    - Finding R_j typically involves estimating γ_j as well
    - The typically strategy is to use a greedy, top-down recursive partitioning algorithm to find it
    - It is sometimes necessary to approximate the empirical risk by a smoother and more convenient criterion for optimizing the R_j:

$$\tilde{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{N} \tilde{L}(y_i, T(x_i, \Theta)) \tag{93}$$

- Then given the $\bar{R}_j = \tilde{R}_j$, the γ can be estimated
- The classification trees described in 9.2 using the **Gini index** instead of misclassification loss in growing the tree is used in the **boosted tree model** as a sum

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m) \tag{94}$$

- induced in a forward stagewise manner
  - At each step in the forward stagewise procedure one must solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{j=1}^{N} L(y_i, f_{m-1}(x_i)T(x_i, \Theta_m)) \tag{95}$$

- for the region set and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ of the next tree given the current model $f_{m-1}(x)$
  - Given the regions R_{jm} finding the optimal constants γ_{jm} in each region is typically straightforward:

$$\hat{\gamma} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{96}$$

- Finding the regions is difficult, and even more difficult than for a single tree, though for a few special cases it simplifies
  - For squared-error loss, the solution is no harder than for a single tree
    - It is simply the regression tree that best predicts the current residuals $y_i - f_{m-1}(x_i)$ and $\hat{\gamma}_{jm}$ is the mean of these residuals in each corresponding region
  - For two-class classification and exponential loss, this stagewise approach gives rise to the AdaBoost method for boosting classification trees
    - If the trees $T(x; \Theta_m)$ are restricted to be scaled classification trees the the solution is one that minimizes the weighted error rate $\sum_i = 1^N w_i^{(i)} I(y_i \neq T(x_i; \Theta_m))$ with weights $w_i^{(m)} = e^{-y_i f_{m-1}(x_i)}$
    - A scaled classification tree is $\beta_m T(x; \Theta_m)$ with the restriction that $\gamma_{jm} \in \{-1, 1\}$
    - A non-scaled classification three the weighted exponential criterion for the new tree is

$$\hat{\theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} w_i^{(m)} \exp[-y_i T(x_i; \Theta_m)] \tag{97}$$

- It is straightforward to implement a greedy recursive partioning algorithm using this weighted exponential loss as a splitting criterion

– Given the R_{jm} one can show that solution is the weighted log-odds in each corresponding region

$$\hat{\gamma}_{jm} = \frac{1}{2} \log \frac{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = 1)}{\sum_{x_i \in R_{jm}} w_i^{(m)} I(y_i = -1)}.$$

- Using loss criteria such as the absolute error or the Huber loss in place of squared-error loss for regression, and the deviance in place of exponential loss for classification, will serve to robustify boosting trees
  - Unfortunately, unlike their nonrobust counterparts, these robust criteria do not give rise to simple fast boosting algorithms

13.9 Numerical Optimization via Gradient Boosting

13.9.1 General

- The loss in using $f(x)$ to predict y on the training training data is

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{98}$$

- The goal is to minimize $L(f)$ with respect to f
  - Where $f(x)$ is constrained to be a sum of trees.
  - Ignoring the constraint, minimizing it can be viewed as a numerical optimization

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} L(\boldsymbol{f}) \tag{99}$$

- where the "parameters $\boldsymbol{f} \in \mathbb{R}^n$ are the values of the approximating function $f(x_i)$ for each of the N data points x_i

$$\boldsymbol{f} = \{f(x_1), f(x_2), \ldots, f(x_N)\}^T \tag{100}$$

- Numerical optimization procedures solve it as a sum of component vectors

$$\boldsymbol{f}_M = \sum_{m=0}^{M} \boldsymbol{h}_m, \ \boldsymbol{h}_m \in \mathbb{R}^n \tag{101}$$

- where $\boldsymbol{f}_0 = \boldsymbol{h}_0$ is the initial guess, and each successive $\boldsymbol{f}_m$ is based on the current parameters vecor $\boldsymbol{f}_{m-1}$ which is the sum of the previously induced updates induced updates.
  - Numerical optimization methods differ in their prescriptions for computing each increment vector $\boldsymbol{h}_m$

13.9.2 Steepest Decent

- Steepest decent chooses $\boldsymbol{h}_m = -\rho_m \boldsymbol{g}_m$ where ρ_m is a scalar and $\boldsymbol{g}_m \in \mathbb{R}^N$ is the gradient of $L(\boldsymbol{f})$ evaluated at $\boldsymbol{f} = \boldsymbol{f}_{m-1}$
  - The component of the gradient $\boldsymbol{g}_m$ are

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} \tag{102}$$

- The step length ρ_m is the solution to

$$\rho_m = \arg\min_{\rho} L(\boldsymbol{f}_{m-1} - \rho \boldsymbol{g}_m) \tag{103}$$

- The current solution is the updated

$$\boldsymbol{f}_m = \boldsymbol{f}_{m-1} - \rho_m \boldsymbol{g}_m \tag{104}$$

- and the process repeated at the next iteration
  - It can be viewed as a very greedy strategy, since $\boldsymbol{g}_m$ is the local direction in $\mathbb{R}^N$ for which $L(\boldsymbol{f})$ is most rapidly decreasing
  -

13.9.3 Gradient Boosting

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

- Forward stagewise boosting is a very greedy strategy
  - The tree prediction $T(x_i; \Theta_m)$ are analogous to the components of the negative gradient
    * The difference between them are that the tree components $\boldsymbol{t}_m = \{T(x_1; \Theta_m), \ldots T(x_N; \Theta_m)\}$ are not independent
    * They are constrained to the predictions of a J_m terminal node whereas the negative gradient is the unconstrained maximal descent direction
    * A way to solve this is to induce a tree $T(x; \Theta_m)$ at the mth iteration whose predictions $\boldsymbol{t}_m$ are as close as possible to the negative gradient which leads to

$$\tilde{\theta}_m = \arg\min_{\theta} \sum_{i=1}^{N} (-g_{im} - T(x_i; \theta))^2 \tag{105}$$

- One fits the tree T to the negative gradient values by least squares
  - Eventough the solution regions $\tilde{R}_{jm}$ to this will not be the same as the solution regions to the general loss function, it will be similar enough to server the same purpose

13.9.4 Implementations of Gradient Boosting

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm},\ j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

- The algorithm for classification is similar
  - Lines 2(a)-(d) are repeated K times at each iteration m, once for each class using 10.38
  - The result at line 3 is K different (coupled) tree expansions \$f$_{kM}$(x), k=1,2,. . . ,K£
  - The result produce probabilities via the softmax function

13.10 Right-Sized Trees for Boosting

- Using pruning to decide the size of the tree results in too large trees to begin with
  - The simplest strategy for avoiding this is to restrict all trees to be the same size $J_m = J \forall m$
  - At each iteration a J terminal node regression tree is induced
  - J becomes a meta-peter of the entire boosting procedure to be adjusted to maximize the estimate performance

- One can get an idea of useful values for J by considering the properties of the target function

$$\eta = \arg\min_{f} {}_{XY} L(Y, f(X)) \tag{106}$$

- The target function $\eta(x)$ is the one with minimum prediction risk on future data
  - This is the function we are trying to approximate
  - On important property of $\eta(X)$ is the degree to which the coordinate variables $X^T = (X_1, X_2, \ldots, X_p)$ interact with each other
  - ANOVA expansions catches the about of interaction between coordinate variables

$$\eta(X) = \sum_j \eta_j(X_j) + \sum_{jk} \eta_j k(X_j, X_k) + \sum_{jkl} \eta_j kl(X_j, X_k, X_l) + \ldots \tag{107}$$

- The sum is over functions of only a single predictor variable X_j
  - The particular functions $n_j(X_j)$ are those that jointly best approximate $\eta(X)$ under the loss criterion being use
    - They are called the "main effect" of $\mathrm{X_j}$@
  - The second sum over two variable functions are called second-order interactions of each respectable variable pair
  - The third sum over three variable functions are called third-order interactions and so on
  - For many problems in practice are typically dominate by low-order interactions
    - When this is the case, models that produce strong higher-order interaction effect such as large decision trees suffer in accuracy
- The interaction level of tree based approximations is limited by the tree size J
  - No interactions effect of level greater than $J - 1$ are possible
  - Since boosted model are additive in the trees this limits them as well
  - Setting $J = 2$ produces models with on allow models with only main effects

- Setting $J = 2$ produces models where two variable interaction effects are also allowed and so on
- Experience so far indicates that $4 \leq J \leq 8$ works well in the context of boosting, with results being fairly insensitive to particular choices in this range

13.11 Regularization

13.11.1 General

- The other meta-parameter of gradient boosting is the number of boosting interactions M
  - Each iteration usually reduces training risk $L(f_M)$
  - For large enough M the risk can be made arbitrarily small
    - This can lead to overfitting
  - The optimal number M^* for minimizing future risk is application dependent
    - A way to estimate M is to use a validation sample

13.11.2 Shrinkage

- The simplest implementation of shrinkage in the context of boosting is to scale the contribution of each tree by a factor $0 < \nu < 1$ when it is added to the current approximation
  - Line 2(d) of algorithm 10.3 is replaced by

$$f_m(x) = f_{m-1}(x) + \nu \cdot \sum_{j=1}^{J} \gamma_{j_m} I(x \in R_{jm}) \tag{108}$$

- The parameter ν can thought of ass the learning rate of the boosting procedure
  - Smaller values of ν results in larger training risk for the same number of iterations M
  - Both ν and M control prediction risk on the training data
    - They are no independent
    - Smaller values of ν lead to larger values of M
  - It has been found that smaller values of ν favor better test error

    - * The best strategy appears to be to set ν to be very small ($\nu < 0.1$) and then choose M by early stopping
    - * Yields dramatic improvements for regression and probability estimatio
  - The price paid for using ν is computation
    - * Many iterations are generally computationally feasible
    - * Even on larger data sets
    - * Due to it operating on small trees with no pruning

13.11.3 Subsampling

- With *stochastic gradient boosting*, at each iteration we sample a fraction η of the training observations (without replacement) and grow the next tree using that subsample
  - The rest of the algorithm is identical
  - A typically value for η can be $\frac{1}{2}$
    - * For larger N is can be substantially smaller than $\frac{1}{2}$
  - It not only reduces computing time but in many cases it actually produces a more accurate model
- The downside is that we have four parameters to set: J, M, ν and η
  - Typically some early explorations determine suitable values for J, ν and η leaving M as the primary parameter

13.12 Interpretation

13.12.1 General

- Single decision trees are highly interpretable.
  - The model can be completely represented by a simple two-dimensional graphic (binary tree) that is easily visualized.
  - Linear combinations of trees (10.28) lose this important feature, and must therefore be interpreted in a different way.

13.12.2 Relative Importance of Predictor Variables

- In data mining applications the input predictor variables are seldom equally relevant.
  - Often only a few of them has substantial influence on the response
    - Vast majority are irrelevant
  - It is often useful to learn the relative importance or contribution of each variable in an given prediction
    - For a single decision tree T the following is a measurement of relevance for ech predictor variable X_ℓ

$$\mathcal{T}^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 I(v(t) = \ell) \tag{109}$$

- The sum is over the internal nodes of the tree.
  - At each node t one of the input variables $X_{v(t)}$ is used to partition the region associated with that node into two sub regions
    - Within each a separate constant is fit to the response values
    - The particular variable chosen is the once that gives maximal estimate improvement $\hat{i}_t^2$ in squared error risk over that for a constant fit over the entire region
    - The squared relative importance of variable $X_\mathcal{L}$ is the sum of sub squared improvements over all internal nodes
  - This importance is easily generalized to additive tree expansions

$$\mathcal{T}_\ell^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{T}_\ell^2(T_m) \tag{110}$$

- This measure is more reliable than for a single tree
  - Both measurements referees to the squared relevance

13.12.3 Partial Dependence Plots

- The step after identifying the most relevant variable is to attempt to understand the nature of the dependence of the approximation $f(X)$ on their join values

  - Graphical renderings of the $f(X)$ as a functions of its arguments gives a comprehensive summary of this de dependence
    * Limited to low-dimensional view
    * Can be viewed by fixing all but one or two producing a trellis of plots

- A general function $f(X)$ will in principle depend on all of the input variables: $f(x) = f(X_{\mathcal{S}}, X_{\mathcal{C}})$
  - One way to define the average or partial dependence of $f(X)$ on $X_{\mathcal{S}}$ is

$$f_{\mathcal{S}}(X_{\mathcal{S}}) = E_{X_c} f(X_{\mathcal{S}}, X_{\mathcal{C}}) \tag{111}$$

- This can server as a useful description of the effect of the chosen subset on $f(X)$
  - e.g. when the variables in $X_{\mathcal{S}}$ do not have strong interactions with those in $X_{\mathcal{C}}$
  -

- Partial dependence function can used to interpret the result of any "black box" learning method
  - One way to estimate them is

$$\hat{f}_{\mathcal{S}}(X_{\mathcal{S}}) = \frac{1}{N} \sum_{i=1}^{N} f(X_{\mathcal{S}}, X_{i'\mathcal{C}}) \tag{112}$$

- where $x_{1\mathcal{C}}, x_{2\mathcal{C}}, \ldots, x_{N\mathcal{C}}$ are the values of $X_{\mathcal{C}}$ occurring in the training data
  - It can be rapidly computed from the tree itself without reference to the data

14 Sequential Data

14.1 Markov Models

- Without the loss of generality one can use the product rule to express the joint distribution for a sequence of observations in the form

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \prod_{n=1}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) \tag{113}$$

- If it is assumed that each of the conditional distributions on the right-hand side is independent of all previous observations except the most recent, we obtain the first-order Markov chain
  - The joint distribution for a sequence of N observations under this models is given by

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = p(\boldsymbol{x}_1) \prod_{n=2}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}) \tag{114}$$

- The conditional distribution for observation $\boldsymbol{x}_n$, given all of the observations up to time n is this given by

$$p(\boldsymbol{x}_n \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) = p(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}) \tag{115}$$

- In most applications of these model, the conditional distributions $p(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1})$ that define the model will be constrained to be equal
  - Known as a **homogeneous Markov chain**
  - Still very constrained even though it is more general than the independent model

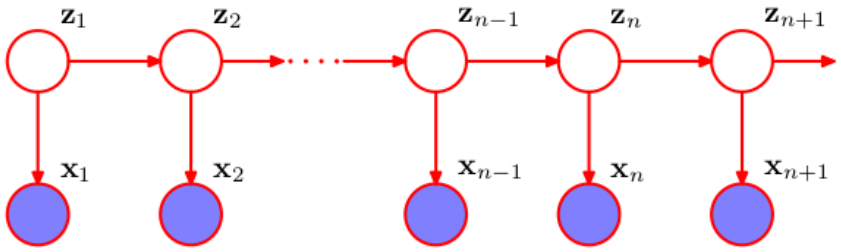
- If predictions also are allowed to depend on the previous-but-one value we obtained a second order Markov chain where the joint distribution know is given by

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = p(\boldsymbol{x}_1) p(\boldsymbol{x}_2 \mid \boldsymbol{x}_1) \prod_{n=3}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-2}) \tag{116}$$

- Each observation is now influenced by two previous observations

- One can consider extensions to an M^{th} order Markov chain in which the condition distribution for a particular variable depends on the previous M variables
  - The price paid for this increased flexibility is that the number of paramers in the models is now much larger
  - The number of parameters is $K^{M-1}(K-1)$ which grows exponential with M and is therefore impractical for higher numbers of M

- For continuous variables we can use linear-Gaussian conditional distributions in which each node has a Gaussian distribution whose mean is a linear function of its parents
  - Known as an autoregressive or AR model
- An alternative approach is to use a parameter model for $p(x_n \mid x_{n-M}, \ldots, x_{n-1})$ such as a neural network
  - Sometime called a tapped delay line
  - The number of parameters can be much smaller than in a completely general model
  - It is achieved at the expense of a restricted family of conditional distributions



- To build a model for sequences that is not limited by the Markov assumption in any order and that can be specified using a limited number of free parameters
  - It can be achieve by introducing additional latent variables to permit a rich class of models to be constructed out of simple components
  - For each observation $\boldsymbol{x}_n$ there is introduced a corresponding latent variable $\boldsymbol{z}_n$
    - May be a different type of dimensionality ti the observed variable
  - It is assumed that it is the latent variables that form a Markov chain
    - It gives rice to a graphical structure known as a state space model

* It satisfies the key conditional independence property that $\boldsymbol{z}_{n-1}$ and $\boldsymbol{z}_{n+1}$ are independent given $\boldsymbol{z}_n$ so that

$$\boldsymbol{z}_{n+1} \perp\!\!\!\perp \boldsymbol{z}_{n-1} \mid \boldsymbol{z}_n \tag{117}$$

- The joint distribution for this model is given by

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{z}_1, \ldots, \mathbf{z}_N) = p(\mathbf{z}_1) \left[\prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1}) \right] \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{z}_n).$$

- There is always a path connecting any two observed variables $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ via the latent variables
  - The observation for a given $\boldsymbol{x}_{n+1}$ depends on all previous observations

14.2 Hidden Markov Models

14.2.1 General

- The hidden Markov model can be viewed as a specific instance of the state space model where the latent variables are discrete
  - If one examines a single time slice in the model it corresponds to a mixture distribution with component densities given by $p(\boldsymbol{x} \mid \boldsymbol{z})$
  - In the case of a standard mixture model the latent variables are discrete multinomial variables $\boldsymbol{z}_n$ describing which component of the mixture is responsible or generating the corresponding observation $\boldsymbol{x}_n$
    * It is convenient to use a 1 of K coding scheme
  - The probability distribution of $\boldsymbol{z}_n$ is allowed to depend on the state of the previous latent variable $\boldsymbol{z}_{n-1}$ through a conditional distribution $p(\boldsymbol{z}_n | \boldsymbol{z}_{n-1})$
    * Since they are K dimensional binary variable, the conditional distribution corresponds to a table of numbers that we denote by $\boldsymbol{A}$
      · Its elements are known as **transition probabilities**
      · They are given by $A_{jk} = p(z_{nk} = 1 \mid z_{n-1}, j)$

· Since they are probabilities they satisfy $0 \leq A_{jk} \leq 1$ with $\sum_k A_{jk} = 1$
· It has $K(K-1)$ independent parameters
· The conditional distribution can be written explicitly in the form

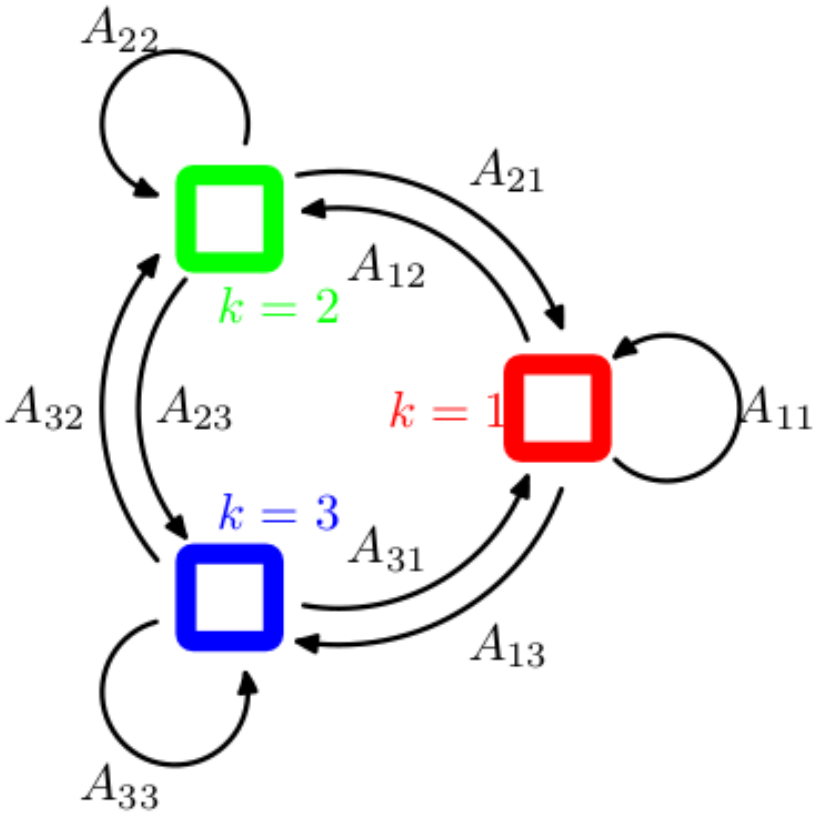
$$p(\boldsymbol{z}_n \mid \boldsymbol{z}_{n-1,\boldsymbol{A}}) = \prod_{k=1}^{K} \prod_{j=1}^{K} A_{jk}^{z_{n-1,j} z_{nk}} \tag{118}$$

- The initial latent node z_1 is special in that it does not have a parent node
  - It has a marginal distribution $p(\boldsymbol{z}_1)$ represented by a vector of probabilities $\boldsymbol{\pi}$ with elements $\pi \equiv p(z_{z1k} = 1)$ so that

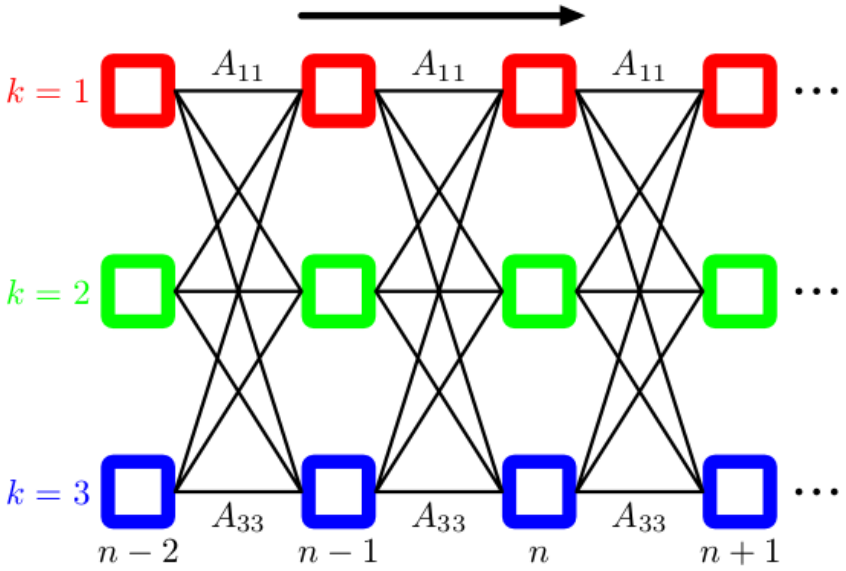
$$p(\boldsymbol{z}_1 \mid \boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_{1k}} \tag{119}$$

where $\sum_k \pi_k = 1$

- The transition matrix is sometimes illustrated diagrammatically by drawing the state as nodes in a state transition diagram e.g.



- It is sometimes to take a state diagram and unfold over time
  - This gives an alternative representation known as a *lattice* or *trellis* diagram
  - e.g.



- The specification of the probabilistic model is completed by defining the conditional distributions of the observed variables $p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \phi)$, where ϕ is a set of parameter governing the distribution
  - These are known as **emission probabilities**
  - They might e.g. be given by Gaussians if the elements of $\boldsymbol{x}$ are continuous variables or by conditional probability tables if $\boldsymbol{x}$ is discrete
  - Since $\boldsymbol{x}_n$ is observed the distribution $p(\boldsymbol{x}_n, \boldsymbol{z}_n, \phi)$ consists for a given value of ϕ of a vector of K numbers corresponding to the K possible states of the binary vector $\boldsymbol{z}_n$
  - The emission probabilities can be represented in the form

$$p(\boldsymbol{x}_n \mid \boldsymbol{z}_n, \phi) = \prod_{k=1}^{K} p(\boldsymbol{x}_n \mid \phi_k)^{z_{nk}} \tag{120}$$

- The joint probability distribution for a *homogeneous* model over both latent and observed variables is given by

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = p(\mathbf{z}_1|\boldsymbol{\pi}) \left[\prod_{n=2}^{N} p(\mathbf{z}_n|\mathbf{z}_{n-1}, \mathbf{A}) \right] \prod_{m=1}^{N} p(\mathbf{x}_m|\mathbf{z}_m, \boldsymbol{\phi})$$

- where $\boldsymbol{X} = \{x_1, \ldots, x_N\}, \boldsymbol{Z} = \{z_1, \ldots, \boldsymbol{z}_N\}$, and $\theta = \{\boldsymbol{\pi}, \boldsymbol{A}, \phi\}$ denoted the set of parameter governing the model
- It is generated the following way
  1. We first choose the initial latent variable $\boldsymbol{z}_1$ with probabilities govern by the parameters π_k
     - We then sample the corresponding observation $\boldsymbol{x}_1$
  2. We chose the state of the variable $\boldsymbol{z}_2$ according to the transition probabilities $p(\boldsymbol{z}_2 \mid \boldsymbol{z}_1)$ using the already instantiated value of $\boldsymbol{z}_1$
  3. If the sample for $\boldsymbol{z}_1$ corresponds to state j, then we chose state k of $\boldsymbol{z}_2$ with probabilities A_{jk} for $k = 1, \ldots K$
  4. Once we know $\boldsymbol{z}_2$ we can draw a sample for $\boldsymbol{x}_2$ and the next latent variable $\boldsymbol{z}_3$ and so on
- If the model has large transition elements A_{kk} that are much larger than the off-diagonal elements a typical data sequence will have long runs of points generated from a single component
- There are many variants of the standard HMM model
  - It can e.g. be obtained for instance by putting constraints on the form of the transition matrix $\boldsymbol{A}$
  - An example if the *right-to-left* HMM, which is obtained by setting the elements A_{jk} of $\boldsymbol{A}$ to zero if $k < j$
    * Such models has typically their initial states probabilities for $p(\boldsymbol{z}_1)$ modified such that \$p(z_{11} = 1) and $p(z_{1j}) = 0$ for $j \neq 1$
    * The state is typically constrained to start in state 1
    * It can also be further constrained to ensure that large changes in the state index do not occur, so that $A_{jk} = 0$ if $k > k + \Delta$

14.2.2 Decodings

- **Posterior decoding**: $\boldsymbol{z}_n^*$ is the most likely state to be in the n'th step

$$\boldsymbol{z}_n^* = \arg\max_{\boldsymbol{z}_n} p(\boldsymbol{z}_n \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \tag{121}$$

14.2.3 Problems

- **Evaluation problem**
  - What is the probability that a particular sequence of symbols is produced by a particular model
  - Two algorithms are used (DO NOT confuse them with the forward-backward algorithm).
    1. The forward algorithm
    2. The backwards algorithm
- **Decoding problem**
  - Given a sequence of symbols (your observations) and a model, what is the most likely sequence of states that produced the sequence.
  - For decoding we use the Viterbi algorithm.
- **Training problem**
  - Given a model structure and a set of sequences, find the model that best fits the data.
    - Can be solved by the following 3 algorithms:
      - MLE (maximum likelihood estimation)
      - Viterbi training(DO NOT confuse with Viterbi decoding)
      - Baum Welch = forward-backward algorithm

14.2.4 Maximum likelihood for the HMM

- If one have observed a data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ one can determine the parameters of an HMM using maximum likelihood
  - It can be obtained from the joint distribution by marginalizing over the latent variables

$$p(\boldsymbol{X} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}, \mid \boldsymbol{\theta}) \tag{122}$$

- Since it does not factorize over n one cannot simply treat each of the summations over $\boldsymbol{z}_n$ independently

  - The summations cannot be performed explicitly because there are N variables to be summed over each of which has K states resulting in K^N terms
  - A further difficulty with this expression is that, because it corresponds to a generalization of a mixture distribution, it represents a summation over the emission models for different settings of the latent variables.
    * Directly maximizing over this will therefore lead to complex expressions with no closed form solutions

- The **EM algorithm** is used to find an efficient framework for maximizing the likelihood function in hidden Markov models
  - It starts with some initial selection for the model parameters which is denoted $\boldsymbol{\theta}^{\text{old}}$
  - In the E step the parameter values are taken to find the posterior distribution of the latent variables $p(\boldsymbol{Z} \mid \boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$
  - It then uses this to evaluate the expectation of the logarithm of the complete-data likelihood function, as a function of the parameters θ, to give the function $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ defined by

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z} \mid \boldsymbol{X}, \theta^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\theta}) \tag{123}$$

- $\gamma(\boldsymbol{z}_n)$ is used to denote the marginal posterior distribution of a latent variable $\boldsymbol{z}_n$ and $\xi(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n)$ to denote the joint posterior distributions of two successive latent variables so that

$$\begin{aligned} \gamma(\boldsymbol{z}_n) &= p(\boldsymbol{z}_n \mid \boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \\ \xi(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n) &= p(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n \mid \boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \end{aligned} \tag{124}$$

- For each value of n
  - We can store $\gamma(\boldsymbol{z}_n)$ using a set of K non-negative number that sum to unity
  - We can store $\xi(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n)$ using a $K \times K$ matrix of non-negative numbers that sum to unit

- $\gamma(z_{nk})$ is used tot denote the condition probability of $z_{nk} = 1$
  - A similar notions is used for $\xi(z_{n-1,j}, z_{nk})$

- Since the expectation of binary random variables is just the probability that it takes the value 1 we have

$$\begin{aligned} \gamma(z_n) &= \mathbb{E}[z_{nk}] = \sum_{\boldsymbol{z}} \gamma(\boldsymbol{z}) z_{nk} \\ \xi(z_{n-1,j}, z_{nk}) &= \mathbb{E}[z_{n-1,j} z_{nk}] = \sum_{\boldsymbol{z}} \gamma(\boldsymbol{z}) z_{n-1,j} z_{nk} \end{aligned} \tag{125}$$

- If we make use of the definitions of γ and ξ we obtain

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) &= \sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi(z_{n-1,j}, z_{nk}) \ln A_{jk} \\ &\quad + \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \ln p(\mathbf{x}_n | \boldsymbol{\phi}_k). \end{aligned} \tag{13.17}$$

- The goal of the E step is to evaluate the quantities $\gamma(\boldsymbol{z}_n)$ and $\xi(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n)$ efficiently

- In the M step we maximize $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ with respect to the parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}\}$ the parameters $\gamma(\boldsymbol{z}_n)$ and $\xi(\boldsymbol{z}_{n-1}, \boldsymbol{z}_n)$ are treated as constants

  - Maximization with respect to π and $\boldsymbol{A}$ is easily achieved by using appropriate Lagrange multipliers with the results

$$\begin{aligned} \pi_k &= \frac{\gamma(z_{1k})}{\sum_{j=1}^{K} \gamma(z_{1j})} \\ A_{jk} &= \frac{\sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nk})}{\sum_{l=1}^{K} \sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nl})}. \end{aligned}$$

- The EM algorithm must be initialized by choosing starting values for $\boldsymbol{\pi}$ and $\boldsymbol{A}$

  - It should respect the summation constraints associated with their probabilistic interpretation

  - Any elements of $\boldsymbol{\pi}$ or $\boldsymbol{A}$ that are initially will remain zero in subsequent EM update
  - A typical initialization procedure would involve selecting random starting values of the parameters subject to the summation and non-negativity constraints
  - There are no particular modification to the EM results for the left-to-right models beyond choosing initial values for the elements A_{jk} in which the appropriate elements are set to zero

- To maximize $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ with respect to ϕ_k it is only the final term in the extended form that depends on ϕ_k
  - It has the same form as the data-dependent term in the corresponding function for a mixture distribution for i.i.d. data
  - We are simply maximizing the weighted log likelihood function for emission density $p(\boldsymbol{x} \mid \boldsymbol{\phi}_k)$ with weights $\gamma(z_{nk})$
  - In the case of Gaussian emission densities we have $p(\boldsymbol{x} \mid \phi_k) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and the maximization of the function $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$ gives

$$\begin{aligned}
\boldsymbol{\mu}_k &= \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})} \\
\boldsymbol{\Sigma}_k &= \frac{\sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_{n=1}^{N} \gamma(z_{nk})}.
\end{aligned}$$

- For the case of discrete multinomial variables the conditional distribution takes the form

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} \prod_{k=1}^{K} \mu_{ik}^{x_i z_k}$$

- and the corresponding M-step equations are given by

$$\mu_{ik} = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) x_{ni}}{\sum_{n=1}^{N} \gamma(z_{nk})}.$$

- The EM algorithm requires initial values for the parameters of the emission distribution.
  - A way to set these is first to treat the data initially as i.i.d. and fit the emission density by maximum likelihood, and then use the resulting values to initialize the parameters for EM.

14.2.5 The forward-backward algorithm

- The **alpha-beta algorithm** is a variant of the **forward-backward algorithm**
- Since the posterior distributions of the latent variables is independent of the form $p(\boldsymbol{x} \mid \boldsymbol{z})$ all we require is the values of the quantities $p(\boldsymbol{x}_n \mid \boldsymbol{z}_n)$ for each value of $\boldsymbol{z}_n$ for every n
  - Therefore the following conditional independence properties does also hold

$$\begin{aligned}
p(\mathbf{X}|\mathbf{z}_n) &= p(\mathbf{x}_1, \ldots, \mathbf{x}_n|\mathbf{z}_n) \\
&\quad p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_n) && (13.24)\\
p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{x}_n, \mathbf{z}_n) &= p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{z}_n) && (13.25)\\
p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{z}_{n-1}) && (13.26)\\
p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_n, \mathbf{z}_{n+1}) &= p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_{n+1}) && (13.27)\\
p(\mathbf{x}_{n+2}, \ldots, \mathbf{x}_N|\mathbf{z}_{n+1}, \mathbf{x}_{n+1}) &= p(\mathbf{x}_{n+2}, \ldots, \mathbf{x}_N|\mathbf{z}_{n+1}) && (13.28)\\
p(\mathbf{X}|\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1}|\mathbf{z}_{n-1}) \\
&\quad p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N|\mathbf{z}_n) && (13.29)\\
p(\mathbf{x}_{N+1}|\mathbf{X}, \mathbf{z}_{N+1}) &= p(\mathbf{x}_{N+1}|\mathbf{z}_{N+1}) && (13.30)\\
p(\mathbf{z}_{N+1}|\mathbf{z}_N, \mathbf{X}) &= p(\mathbf{z}_{N+1}|\mathbf{z}_N) && (13.31)
\end{aligned}$$

- where $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$

- To evaluate $\gamma(z_{nk})$ we can we can use the following formular which uses the Bayers' theorem

$$\gamma(\boldsymbol{z}_n \mid \boldsymbol{X}) = \frac{p(\boldsymbol{X} \mid \boldsymbol{z}_n)p(\boldsymbol{z}_n)}{p(\boldsymbol{X})} \tag{126}$$

- The denominator $p(\boldsymbol{X})$ is implicitly conditions on the parameter $\boldsymbol{\theta}^{\text{old}}$ of the HMM and there for represents the likelihood function
  - Based on property (13.24) together with the product rule of probability we obtain

$$\gamma(\boldsymbol{z}_n) = \frac{p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{z}_n)p(\boldsymbol{x}_{n+1}, \ldots, \boldsymbol{x}_N|\boldsymbol{z}_n)}{p(\boldsymbol{X})} = \frac{\alpha(\boldsymbol{z}_n)\beta(\boldsymbol{z}_n)}{p(\boldsymbol{X})} \tag{127}$$

- where the following is defined

$$\begin{aligned} \alpha(\boldsymbol{z}_n) &\equiv p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{z}_n) \\ \beta(\boldsymbol{z}_n) &\equiv p(\boldsymbol{x}_{n+1}, \ldots, \boldsymbol{x}_N \mid \boldsymbol{z}_n) \end{aligned} \tag{128}$$

- The quantity $\alpha(\boldsymbol{z}_n)$ represents the joint probability of observing all of the given data up to time n and the value of $\boldsymbol{z}_n$
  - $\alpha(z_{nk})$ is used to denote the value of $\alpha(\boldsymbol{z}_n)$ when $z_{nk} = 1$
- $\beta(\boldsymbol{z}_n)$ represents the conditional probability of all future data from time $n + 1$ up to N given the value of $\boldsymbol{z}_n$
- To more efficiently compute $\alpha(\boldsymbol{z}_n)$ the following equation can be used

$$\alpha(\boldsymbol{z}_n) = p(\boldsymbol{x}_n \mid \boldsymbol{z}_n) \sum_{\boldsymbol{z}_{n-1}} \alpha(\boldsymbol{z}_{n-1})p(\boldsymbol{z}_n \mid \boldsymbol{z}_{n-1}) \tag{129}$$

- In order to start computing this recursively an initial condition is need which is given by

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1) = \prod_{k=1}^{K} \{\pi_k p(\mathbf{x}_1|\boldsymbol{\phi}_k)\}^{z_{1k}}$$

- this tells us that $\alpha(z_{1k})$, for $k = 1, \ldots, K$ takes the value $\pi_k p(\boldsymbol{x}_1 \mid \boldsymbol{\phi}_k)$
  - To evaluate $\alpha(\boldsymbol{z}_n)$ along the chain for every latent node the cost is $O(K^2N)$

- Another way to evaluate $\beta(\boldsymbol{z}_n)$ is using the following formula

$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n).$$

- The starting point for recursion is now $\beta(\boldsymbol{z}_N)$ which is equal to

$$p(\boldsymbol{z}_n \mid \boldsymbol{X}) = \frac{p(\boldsymbol{X}, \boldsymbol{z}_n)\beta(\boldsymbol{z}_n)}{p(\boldsymbol{X})} \tag{130}$$

- If we sum both sides over $\boldsymbol{z}_n$ and just the fact that the left-hand side is a normalized distribution we obtain

$$p(\boldsymbol{X}) = \sum_{\boldsymbol{z}_n} \alpha(\boldsymbol{z}_n)\beta(\boldsymbol{z}_b) \tag{131}$$

$p(\boldsymbol{X})$ can be computed by the following formula

$$p(\boldsymbol{X}) = \sum_{\boldsymbol{z}_n} \alpha(\boldsymbol{z}_N) \tag{132}$$

this is much better computing it by summing over all possible values of $\boldsymbol{Z}$

- To implement the forward algorithm without numeric problems $\hat{\alpha}(\boldsymbol{z}_n)$ is used

$$\hat{\alpha}(\boldsymbol{z}_n) = \frac{\alpha(\boldsymbol{z}_n)}{\prod_{m=1}^{n} c_m} \tag{133}$$

- where $c_n = p(\boldsymbol{x}_n \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})$
- To compute the basis step the following formula is used

$$\hat{\alpha}(\boldsymbol{z}_1) = \frac{\alpha(\boldsymbol{z}_1)}{c_1} \tag{134}$$

- where $c_1 = \sum_{k=1}^{K} \pi_k p(\boldsymbol{x}_1 \mid \phi_k)$
- To compute the recursion step the following formula is used

$$\delta(\boldsymbol{z}_n) = c_n \hat{a}(\boldsymbol{z}_n) = p(\boldsymbol{x}_n \mid \boldsymbol{z}_n) \sum_{\boldsymbol{z}_{n-1}} \hat{\alpha}(\boldsymbol{z}_{n-1}) p(\boldsymbol{z}_n \mid \boldsymbol{z}_{n-1}) \tag{135}$$

- The c_n is computed an stored as

$$c_n = \sum_{k=1}^{K} \delta(z_{zk}) \tag{136}$$

- Then compute and store $\hat{\alpha}(z_{nk}) = \delta(z_{nk})/c_n$
- To implement the backward algorithm without numeric problems $\hat{\beta}(\boldsymbol{z}_n)$ is used

$$\hat{\beta} = \beta(\boldsymbol{z}_b) \prod_{m=n+1}^{N} c_m \tag{137}$$

- The basis step is $\hat{\beta}(\boldsymbol{z}_N) = 1$ as the unscaled version
- The recursion step n is to compute the following K values $\epsilon(z_{n1}), \ldots, \epsilon(z_{nK})$

$$\epsilon(\boldsymbol{z}_n) = c_{n+1}\hat{\beta}(\boldsymbol{z}_n) = \sum_{z_{n+1}} \hat{\beta}(\boldsymbol{z}_{n+1}) p(\boldsymbol{x}_{n+1} \mid \boldsymbol{z}_{n+1}) p(\boldsymbol{z}+1 \mid \boldsymbol{z}_n) \tag{138}$$

- Using the c_{n+1} computed during forward recursion compute and store $\hat{\beta}(z_{nk}) = \epsilon(z_{nk})/c_{n+1}$
- Using the scaled versions $\hat{\alpha}$ and $\hat{\beta}$ the following holds

$$p(\boldsymbol{z}_n \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \hat{\alpha}(\boldsymbol{z}_n)\hat{\beta}(\boldsymbol{z}_n) \tag{139}$$

- Then the posterior decoding $\boldsymbol{z}_n^*$ for a given n becomes

$$\boldsymbol{z}_n^* = \arg\max_{\boldsymbol{z}_n} \hat{\alpha}(\boldsymbol{z}_n)\hat{\beta}(\boldsymbol{z}_n) \tag{140}$$

14.2.6 The Viterbi decoding

- **Viterbi decoding**: Z^* is the overall most likely explanation of $\boldsymbol{X}$:

$$\boldsymbol{Z}^* = \arg\max_{\boldsymbol{z}} p(\boldsymbol{X}, \boldsymbol{Z} \mid \boldsymbol{\Theta}) \tag{141}$$

- The following is true for $p(\boldsymbol{X}, \boldsymbol{Z}^*)$

$$p(\boldsymbol{X}, \boldsymbol{Z}^*) = \max_{\boldsymbol{z}_N} \omega(\boldsymbol{z}_N) \tag{142}$$

- where $\omega(\boldsymbol{z}_n) \equiv \max_{z_1,\ldots,\boldsymbol{z}_{n-1}} p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{z}_1 \ldots, \boldsymbol{z}_n)$ is the probabilitry of the most likely sequence of state $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ ending in $\boldsymbol{z}_n$ generating the observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$

- ω can be computed recursivly using the following which takes time $O(K^2N)$ and space $O(KN)$ using memorization
  - A table where $\omega[k][n] = \omega(\boldsymbol{z}_n)$ if $\boldsymbol{z}_n$ is state k
  - Basis: $\omega(\boldsymbol{z}_1) = p(\boldsymbol{x}_1, \boldsymbol{z}_1) = p(\boldsymbol{z}_1)p(\boldsymbol{x}_1 \mid \boldsymbol{z}_1)$
  - Recursion step: $\omega(\boldsymbol{z}_n) = p(\boldsymbol{x}_n \mid \boldsymbol{z}_n) \max_{\boldsymbol{z}_{z-1}} \omega(\boldsymbol{z}_{n-1})p(\boldsymbol{z}_n \mid \boldsymbol{z}_{n-1})$
- To find $\boldsymbol{Z}^*$ from the ω table one can just do backtrack and find out which probability with the equation
  - It takes $O(KN)$ and space $O(KN)$ using ω

14.2.7 Extension of the hidden Markov model

- The basic hidden Markov model, along with the standard training algorithm based on maximum likelihood, has been extended in numerous ways to meet the requirements of particular applications
- If the goal is sequence classification, there can be significant benefit in determining the parameters of hidden Markov models using discriminative rather than maximum likelihood techniques.
  - Suppose we have a training set of R observation sequences X_r, where $r = 1, \ldots, R$ each of which is labelled according to its class m, where $m = 1, \ldots, M$
  - For each class, we have a separate hidden Markov model with its own parameters $\boldsymbol{\theta}_m$
  - The problem of determining the parameter values is done as a standard classification problem where the cross-entropy is optimized

$$\sum_{r=1}^{R} \ln p(m_r \mid \boldsymbol{X}_r) \tag{143}$$

- Using Bayes' theorem this can be expressed in terms of the sequence probabilities associated with the hidden Markov models

$$\sum_{r=1}^{R} \ln \left\{ \frac{p(\mathbf{X}_r|\boldsymbol{\theta}_r)p(m_r)}{\sum_{l=1}^{M} p(\mathbf{X}_r|\boldsymbol{\theta}_l)p(l_r)} \right\}$$

- where $p(m)$ is the prior probability of class m
  - Optimization of this cost function is more complex than for maximum likelihood
  - It requires that every training sequence be evaluated under each of the models in order to compute the denominator
- A significant weakness of the hidden Markov model is the way in which it represents the distribution of times for which the system remains in a given state
  - The problem is that the probability that the Markov model will spend precisely T steps in state k and then make a transition is a exponentially decaying function of T
  - For many applications, this will be a very unrealistic model of state duration
  - The problem can be resolved by
    * Modelling state duration directly in which the diagonal coefficients A_{kk} are all set to zero
    * Each state k is explicitly associated with a probability distribution $p(T \mid k)$ of possible duration times.
- A limitation of the standard HMM is that it is poor at capturing longrange correlations between the observed variables
  - i.e. variables that are separated by many time steps
  - One way to address this is to generalize the HMM to give the autoregressive hidden Markov model
    * For discrete observations it corresponds to expanded tables of conditional probabilities for the emission distributions

15 Conditional probabilities and graphical models

15.1 You have a joint probability — Now what?

- We can split a variables into three kinds
  - Those we have observed
    * Which are thus no longer random
  - Those we don't care about

    - ∗ They are called nuisance parameter
    - ∗ They are just there to build the model
  - The parameters we are interested in
    - ∗ It can be underlying parameters
    - ∗ Future events we want to predict

- **Condition on observed parameters**: If we have the joint probability $p(X, Y, Z)$ but we have already observed $Z = z$ we are no longer interested in the outcome of Z any longer
  - We will still be interested in X and Y
  - When observing Z we move our interest from $p(X, Y, Z)$ to $p(X, Y \mid Z)$
  - It can in general be done from marginalization e.g.

$$p(X, Y \mid Z) = \frac{p(X, Y, Z)}{p(Z)} \tag{144}$$

- and

$$p(Z) = \sum_X \sum_Y p(X, Y, Z) \tag{145}$$

- **Marginalise away nuisance parameters:** If one is only interested in the outcome of X and the Y variable that was needed to connect the variable of interest to the variable Z but in itself has no use
  - The interest does not lie in $p(X, Y \mid Z)$ but in $p(X \mid Z)$ which can be obtained from marginalization

$$p(X \mid Z) = \sum_Y$$

p(X,Y | Z) (146)

- The summing over all possible values used in the marginalisation is potentially computational intractable, so one cannot always simply do this
  - Some times clever algorithms are necessary, but they will typically always just do one of those two things in a clever way

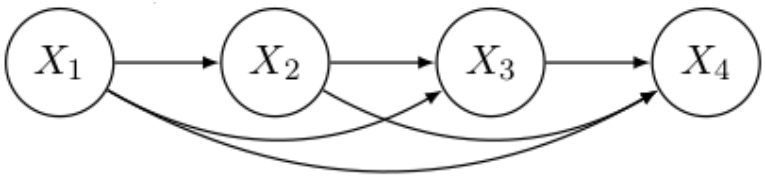


Figure 7: Dependency Graph Example for the distribution $p(X_1, X_2, X_3, X_4)$

15.2 Dependency graphs

- A graphical notation for describing dependency relationships when specifying a joint distribution
  - Each random variable is represented as a node
  - Whenever the composition of the joint probability has a term $p(Y \mid X, \ldots, X_k)$ we have directed edges from all X_i's to Y

16 Johnson-Lindenstrauss Dimensionality Reduction

16.1 Intro

- The goal of **dimensionality reduction** is to represent a high dimensional data set in a lower dimensional space while preserving much of the important structure
- **Principal Component Analysis** is a well known dimensionality reduction technique that finds (a few) directions of high variance in the data and projects the data onto these vectors of maximal variance
  - The idea is to maintain as much variance in the data as possible (the structure) while reducing the dimensionality
  - The mapping of data from a high dimensional space to a low dimensional space is called an **embedding**
    - i.e. we embed the higher dimensional data points in the lower dimensional space
  - The dimensionality reduction technique described approximately presevers all pairwise distances between the data points.

- A fundamental result of Johnson and Lindenstrauss says that any m point subset of Euclidean space can be linearly embedded in $k = O(\lg m/\epsilon^2)$ dimensions without distorting the distances between any pair of points by more than a factor of $1 \pm \epsilon$ for any $0 < \epsilon < \frac{1}{2}$
  - For Principal Component Analysis to be relevant the original data points $x_1, \ldots, x_m$ must be inherently low dimensional
  - The **Johnson-Lindenstraus theorem** requires no assumption on the original data and the target dimension is even independent of the input dimension
  - Another fundamental result og Larsen and Nelson, shows that that the Johnson-Lindenstrauss Lemma is tight even for non-linear embedding!
  - Besides the obvious application of using dimensionality for compression, the Johnson–Lindenstrauss theorem has found numerous applications in algorithms and machine learning for instance for Approximate

k-Means, Linear Regression, and Approximate Nearest Neighbors.

16.2 Simple JL Lemma

- **Theorem 1** For any $0 < \epsilon < \frac{1}{2}$ and any integer m, then for integer

$$k = O(\frac{1}{\epsilon^2} \lg m) \tag{147}$$

- large enough and any points $x_1, \ldots, x_m \subset \mathbb{R}^d$ there exists a linear map (matrix) $L : \mathbb{R}^d \to \mathbb{R}^k$ such that for any $1 \leq i, j \leq m$

$$(1-\epsilon)||x_i - x_j||_2^2 ||Lx_i - Lx_j||_2^2 \leq (1+\epsilon)||x_i - x_j||_2^2 \tag{148}$$

- The linear transformation L in Theorem 1 is simply multiplication by a matrix whose entries are sampled independently from a standard Gaussian
  - To be precise define a random variable A as a $k \times d$ matrix where each entry $A_{i,j} \sim \mathcal{N}(0,1)$
  - The final embedding matrix is sample of the random variable $L = \frac{1}{\sqrt{k}} A$

- **Lemma 1.** Fix any vector unit vector x. For any $0 < \epsilon, \delta < \frac{1}{2}$. For $k = O(\epsilon^{-2} \log \frac{1}{\delta})$ large enough let $L = \frac{1}{\sqrt{k}}$ be a random variable where A is a $k \times d$ random matrix whose entries are independent zero mean Gaussians ($\sim \mathcal{N}(0,1)$) Then:

$$\Pr_L(|||Lx||^2 - 1| > \epsilon \leq delta \tag{149}$$

- Stated differently, for any unit vector x and values of $0 < \epsilon, \delta < \frac{1}{2}$. If we pick a random matrix as described, then with probability at least $1 - \delta$ the norm of x is distorted by a vector at most $(1 \pm \epsilon)$ by L
  - This generalizes to any non-unit vector since any vector v can be written as scribed, then with probability at least 1 the norm of x is distorted by a factor at most (1 ±) by L.
  - Note that this generalizes to any non-unit vector since a vector v can be written as $v = ||v|| \frac{v}{||v||}$
    - Since embedding is linear

- Result about the Normal Distribution and the χ^2

Fact 1. *For any constants a, b if $X \sim \mathcal{N}(0,1)$, and $Y \sim \mathcal{N}(0,1)$ then $aX + bY \sim \mathcal{N}(0, a^2\sigma_x^2 + b^2\sigma_y^2)$.*

Definition 1. *If $Z_1, ..., Z_k$ are independent, standard normal random variables ($Z_i \sim \mathcal{N}(0,1)$), then the sum of their squares,*

$$Q = \sum_{i=1}^{k} Z_i^2$$

is distributed according to the chi-squared distribution with k degrees of freedom. This is usually denoted as $Q \sim \chi_k^2$.

Lemma 2 ([2])**.** *Let $Y \sim \chi_k^2$. Then*

$$\Pr_Y\left(\left|\frac{Y}{k} - 1\right| \geq x\right) \leq e^{-\frac{3}{16}kx^2}, x \in [0, \frac{1}{2})$$

Lemma 3. *Given any unit vector $x \in \mathbb{R}^d$. Let A be the random variable matrix with each $A_{i,j} \sim \mathcal{N}(0,1)$ independently of the other entries. Then the random variable that is the squared norm of Ax is $\chi_{k,d}$ distributed. To be precise:*

$$\|Ax\|^2 \sim \chi_k^2$$

17 Principal Component Analysis

17.1 Intuition

17.1.1 Problem statement

- The dimensionality reduction is done linearly

- The original data points should be possible to construct from the optimal subspace as well as possible
- The mean squared distance between original and reconstructed data points is the **reconstruction error**
- It is useful and common practice to remove the mean value from the data before doing the dimensionality reduction
- Zero mean data is assumed throughout
- Variance and 2nd moments are the same

17.1.2 Projection and reconstruction error

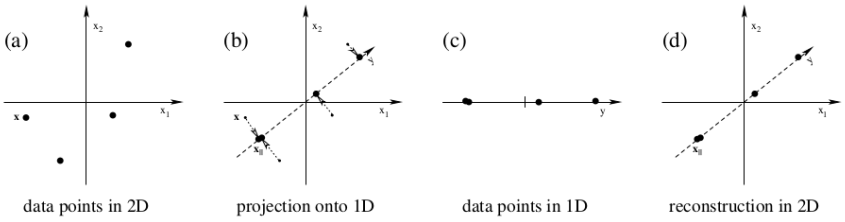


Figure 1: Projection of 2D data points onto a 1D subspace and their reconstruction.

- The task of principal component analysis (PCA) is to reduce the dimensionality of some high-dimensional data points
  - This is done by linearly projecting them onto a lower dimensional space in a way that makes the reconstruction error minimal

17.1.3 Reconstruction error and variance

- Minimizing the construction error is equivalent to maximizing the variance of the projected data

17.1.4 Covariance matrix

- The data points are written as $\boldsymbol{x} = (x_1, x_2)^T$
  - The variance of the first and second component can be written as $C_{11} := \langle x_1 x_1 \rangle$ and $C_{22} := \langle x_2 x_2 \rangle$
    - The angle brackets indicate averaging over all data points
  - If C_{11} is large compared to C_{22} the direction of maximal variance is close to $(1, 0)^T$

- If C_{11} is small the direction of maximal variance is close to $(0,1)^T$
- If C_{11} and C_{22} are similar the covariance between the two components $C_{12} := \langle x_1 x_1 \rangle$ can give additional information
  - A large positive value indicate a strong correlation and then $(1,1)^T$ should be used
  - A negative value would indicate anti-correlation and then $(-1,1)^T$ should be used
  - A small value would indicate no correlation, i.e. no prominent direction of maximal variance
- The variances and covariances are arranged in a matrix with components $C_{ij} := \langle x_i x_j \rangle$
  - It is called **covariance matrix**, assuming zero mean data
  - The components obey the relation: $C_{ij} \leq C_{ii} C_{jj}$
  - Scaling the data by a factor α scales the matrix by a factor α^2

17.1.5 Covariance matrix and higher order structure

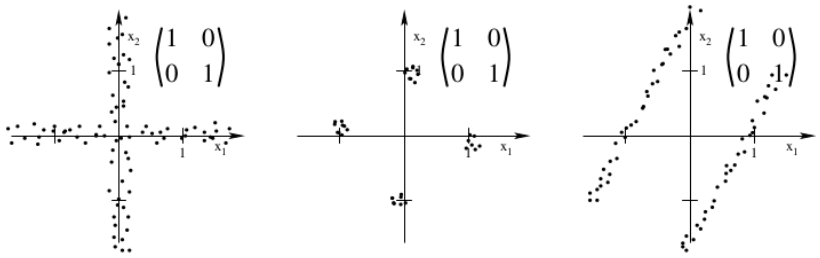


Figure 4: Different data distributions with identical covariance matrices.

- The covariance matrix only gives information about the general extent of the data
  - It does not give any information about the higher order structure of the data cloud

17.1.6 PCA by diagonalizing the covariance matrix

- If the covariance matrix is diagonal the direction of maximal variance is the axis belonging to the largest value of the covariance matrix

- We can make a non-diagonal covariance matrix diagonal by rotating the coordinate system accordingly
- Diagonalizing a matrix can be done by solving the corresponding eigenvalue equation
- The eigenvectors of the covariance matrix point into the directions of maximal and minimal variance
- The eigenvalue are equal to the variances along these direction
- Projecting the data onto the eigenvectors with largest eigenvalues is therefore the optimal linear dimensionality reduction.

17.2 Formalism

17.2.1 Definition of the PCA-optimization problem

- The problem of principal component analysis (PCA) can be stated as follows:

Principal Component Analysis (PCA): Given a set $\{\mathbf{x}^\mu : \mu = 1, ..., M\}$ of I-dimensional data points $\mathbf{x}^\mu = (x_1^\mu, x_2^\mu, ..., x_I^\mu)^T$ with zero mean, $\langle \mathbf{x}^\mu \rangle_\mu = \mathbf{0}_I$, find an orthogonal matrix $\mathbf{U}$ with determinant $|\mathbf{U}| = +1$ generating the transformed data points $\mathbf{x}'^\mu := \mathbf{U}^T \mathbf{x}^\mu$ such that for any given dimensionality P the data projected onto the first P axes, $\mathbf{x}'^\mu_{||} := (x'^\mu_1, x'^\mu_2, ..., x'^\mu_P, 0, ..., 0)^T$, have the smallest

$$\textbf{reconstruction error} \quad E := \langle \|\mathbf{x}'^\mu - \mathbf{x}'^\mu_{||}\|^2 \rangle_\mu \tag{8}$$

among all possible projections onto a P-dimensional subspace. The row vectors of matrix $\mathbf{U}$ define the new axes and are called the *principal components*.

- Remarks about the problem
  1. $\langle \boldsymbol{x}^\mu \rangle_\mu$ indicate the mean over all M data points indexed with μ
  2. If one has non-zero-mean data, one typically removes the mean before applying PCA
  3. Matrix $\boldsymbol{U}$ corresponds to a rotation of the data $\boldsymbol{x}$
     - The shape of the data cloud remains the same
     - The perspective changes
  4. Projecting the data $\boldsymbol{x}'$ onto the P dimensional linear subspace is done by setting all components higher than P to zero

- This can be done, because we have an orthonormal coordinate system
- If this was not the case the projection would become more mathematically complex

5. The reconstruction error has to be minimal for any P

17.2.2 Matrix V^T: Mapping from high-dimensional old coordinate system to low-dimensional new coordinate system

- Assume some data point $\boldsymbol{x}$ are given in a I dimensional space and a linear subspace is spanned by P orthonormal vectors

$$\begin{aligned} \circ \quad \mathbf{v}_p &:= (v_{1p}, v_{2p}, ..., v_{Ip})^T \\ \circ \quad \text{with} \quad \mathbf{v}_p^T \mathbf{v}_q &= \delta_{pq} := \begin{cases} 1 & \text{if } p = q \\ 0 & \text{otherwise} \end{cases} . \end{aligned}$$

- It is assumed that $P < I$ and spear of a high(I) dimensional space and a low(P) dimensional (sub)space
- Arraigning the vectors into a matrix yields

$$\begin{aligned} \bullet \quad \mathbf{V} &:= (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_P) \\ &\overset{(9)}{=} \begin{pmatrix} v_{11} & v_{12} & \dots & v_{1P} \\ v_{21} & v_{22} & \dots & v_{2P} \\ \vdots & & \ddots & \vdots \\ v_{I1} & v_{I2} & \dots & v_{IP} \end{pmatrix} . \end{aligned}$$

- This matrix can be used to map the data points $\boldsymbol{x}$ into the subspace spanned by the vectors v_p yielding

$$\boldsymbol{y} := \boldsymbol{V}^T \boldsymbol{x} \tag{150}$$

- To things are done here
  - The points are moved from the high-dimensional space onto the low-dimensional subspace
  - The points are represented in a new coordinate system
    - It is particular suitable for the low-dimensional subspace
    - The points in the high dimensional space cannot be represented accurately because we does not have enough dimensions

17.2.3 Matrix V: Mapping from low-dimensional new coordinate system to subspace in old coordinate system

- Since the vectors $\boldsymbol{v}_p$ are orthonormal, matrix V can also be used to transform the points back from the new to the old coordinate system
  - The lost dimensions cannot be recovered
  - The mapped $\boldsymbol{y}$ in the new coordinate system become points $\boldsymbol{x}_{||}$ in the old coordinate system and are given by

$$\boldsymbol{x}_{||} := \boldsymbol{V}\boldsymbol{y} = \boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x} \tag{151}$$

- $\boldsymbol{y}$ and $\boldsymbol{x}_{||}$ are equivalent representations
  - The contain the same information just in different coordinate systems

17.2.4 Matrix ($\boldsymbol{V}^T\boldsymbol{V}$): Identity mapping within new coordinate system

- $\boldsymbol{V}^T\boldsymbol{V}$ is a $P \times P$ matrix and performs a transformation from the new coordinate system to and back again
  - Since the all the points in the old coordinate system come from the new coordinate system the mapping does not discard any information
  - $\boldsymbol{V}^T\boldsymbol{V}$ is the identity matrix

17.2.5 Matrix (VV^T): Projection from high- to low-dimensional (sub)space within old coordinate system

- The matrix VV^T maps the points $\boldsymbol{x}$ onto the low-dimensional subspace
  - The mapped points are represented within the old coordinate system
  - This operation does not make a difference whether you apply more than once
  - $\boldsymbol{P} = \boldsymbol{V}\boldsymbol{V}^T$
  - The smaller P there more information is lost
    - The more does $\boldsymbol{P}$ differ from the identity matrix

17.2.6 Variance

- The variance of a multi-dimensional data set is defined to be the sum over the variance of its components

$$\begin{aligned} \bullet \quad \mathrm{var}(\mathbf{x}) &:= \sum_{i=1}^{I} \langle x_i^2 \rangle \\ \circ &= \left\langle \sum_{i=1}^{I} x_i^2 \right\rangle \\ \bullet &= \langle \mathbf{x}^T \mathbf{x} \rangle \end{aligned}$$

- This also holds for the projected data $\mathrm{var}(\boldsymbol{y}) = \langle \boldsymbol{y}^T \boldsymbol{y} \rangle$

17.2.7 Reconstruction error

- The reconstruction error E is defined as the mean square sum over the distances between the orginal data points $\boldsymbol{x}$ and the projected ones $\boldsymbol{x}_{||}$
  - If we define the orthogonal vectors

$$\boldsymbol{x}_{\perp} = \boldsymbol{x} - \boldsymbol{x}_{||} \tag{152}$$

- The following result can be found

$$E = \langle \boldsymbol{x}^T \boldsymbol{x} \rangle - \langle y^T y \rangle \tag{153}$$

- The reconstruction error equals the difference between the variance of the data minus the variance of the projected data.

17.2.8 Covariance matrix

- The covariance matrix can be written as follows in vector notation

$$\bullet \quad \mathbf{C}_x := \left\langle \mathbf{x}\mathbf{x}^T \right\rangle = \frac{1}{M} \sum_{\mu} \mathbf{x}^{\mu} \mathbf{x}^{\mu T} .$$

17.2.9 Eigenvalue equation of the covariance matrix

- The eigenvalues for the covariance matrix are real and a set of orthogonal eigenvectors always exists

- For a given covariance matrix $\boldsymbol{C}_x$ we can always find a complete set of real eigenvalues λ_i and corresponding eigenvectors $\boldsymbol{u}_i$ such that

  - $\mathbf{C}_x\mathbf{u}_i = \mathbf{u}_i\lambda_i$ (eigenvalue equation),
    - $\lambda_i \geq \lambda_{i+1}$ (eigenvalues are ordered),
  - $\mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}$ (eigenvectors are orthonormal).

- If the eigenvectors are combined into an orthogonal matrix $\boldsymbol{U}$ and the eigenvalues into a diagonal matrix $\boldsymbol{\Lambda}$

  - $\mathbf{U} := (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_I)$,
  - $\boldsymbol{\Lambda} := \mathrm{diag}(\lambda_1, \lambda_2, ..., \lambda_I)$,

- The following holds

    - $\mathbf{U}^T\mathbf{U} \overset{(42,43)}{=} \mathbf{1}_I$ (matrix $\mathbf{U}$ is orthogonal),
  - $\iff \mathbf{U}\mathbf{U}^T = \mathbf{1}_I$ (since $\mathbf{U}^{-1} = \mathbf{U}^T$ and $\mathbf{U}$ is quadratic),
    - $\mathbf{C}_x\mathbf{U} \overset{(40,43,44)}{=} \mathbf{U}\boldsymbol{\Lambda}$ (eigenvalue equation),
  - $\overset{(45)}{\iff} \mathbf{U}^T\mathbf{C}_x\mathbf{U} = \boldsymbol{\Lambda}$
  - $\overset{(45,46)}{\iff} \mathbf{C}_x = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$.

17.2.10 Total variance of data $\boldsymbol{x}$

- Given a eigenvector matrix $\boldsymbol{U}$ and the eigenvalue matrix $\boldsymbol{\Lambda}$ it is easy to compute the total variance of the data

$$\langle \boldsymbol{x}^T\boldsymbol{x} \rangle = \sum_i \lambda_i \tag{154}$$

- The total variance of the data is the sum of the eigenvalues of its covariance matrix

17.2.11 Diagonalizing the covariance matrix

- The matrix $\boldsymbol{U}$ can be used to transform the data such that the covariance matrix becomes diagonal
  - Define $X' := \boldsymbol{U}^T \boldsymbol{x}$ and denote the new covariance matrix by $\boldsymbol{C}_x^{'}$
  - The we have $\boldsymbol{C}_x^{'} := \boldsymbol{\Lambda}$

17.2.12 Constraints of matrix $V^{'}$

- Since the vectors $v_p^{'}$ are orthonormal $\boldsymbol{V'}$ can always be completed to an orthogonal $I \times I$ matrix by adding $I - P$ additional orthonormal vectors
- The following holds
  - $\sum_i (v'_{ip})^2 = 1$ (column vectors of $\mathbf{V}'$ have norm one),
  - $\implies \sum_{ip} (v'_{ip})^2 = P$ (square sum over all matrix elements equals P),
  - $\sum (v'_{ip})^2 \leq 1$ (row vectors of $\mathbf{V}'$ have norm less or equal one).

17.2.13 Finding the optimal subspace

- The following holds
  - $$v'_{ip} := \delta_{ip} := \begin{cases} 1 & \text{if } i = p \\ 0 & \text{otherwise} \end{cases},$$

17.2.14 Interpretation of the result

- $\boldsymbol{V}^{'}$ projects the data $\boldsymbol{x'}$ onto the first P axes
  - It is a projection onto the first P eigenvectors of the covariance matrix $\boldsymbol{C}_x$
  - The following is defined
    - $\mathbf{V} := \mathbf{U}\mathbf{V}'$
    - $\overset{(43,\,74)}{=} (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_P)$

- We would set $\boldsymbol{v}_p := \boldsymbol{u}_p$
- The variance of y is $\sum_{i=1}^{P} \lambda_i$
- The reconstruction error is $E = \sum_{i=P+1}^{I} \lambda_i$

17.2.15 Whitening or sphering

- It is sometimes desirable to transform a data set such that it has variance one in all directions
  - It is called **whitening** or **sphering**
  - Sphering requires to stretch and compress the data distribution along the axes of the principal components such they have variance one
  - One first rotates the data into a coordinate system where the covariance is diagonal, then performs stretching along the aces and then rotates the data back into the original coordinate system
  - The eigenvectors of the covariance matrix provide the aces of the new coordinate system
    - The eigenvalues indicate the variances and therefore how much one has to stretch the data
    - Sphering is achieved by multiplying the data with a sphering matrix

$$\bullet \quad \mathbf{W} := \mathbf{U}\,\mathrm{diag}\left(\frac{1}{\sqrt{\lambda_1}}, \frac{1}{\sqrt{\lambda_2}}, ..., \frac{1}{\sqrt{\lambda_I}}\right)\mathbf{U}^T$$

$$\bullet \quad \hat{\mathbf{x}} := \mathbf{W}\mathbf{x}\,.$$

- If the final orientation does not matte this matrix is often defined without its first $\boldsymbol{U}$
  - The sphering matrix is symmetrical

17.2.16 Singular value decomposition

- Sometimes one has fewer data points than dimensions
  - Then doing direct PCA is very inefficient and **singular value decomposition** (SVD) is the very helpful

- Let $\boldsymbol{x}^\mu$, $\mu = 1, \ldots, M$ be the I dimensional data with $M < I$
  - All the data can be written in the one $I \times M$ matrix $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^M)$
  - The second-moment matrix can then be written as

$$\boldsymbol{C}_1 := \boldsymbol{X}\boldsymbol{X}^T/M \tag{155}$$

- It eigenvalue equation and decomposition read

$$\begin{aligned} \mathbf{C}_1\mathbf{U}_1 &= \mathbf{U}_1\mathbf{\Lambda}_1 \\ \iff \quad \mathbf{C}_1 &= \mathbf{U}_1\mathbf{\Lambda}_1\mathbf{U}_1^T . \end{aligned}$$

- The data represented in the coordinate system of the eigenvectors is

$$\boldsymbol{Y}_1 := \boldsymbol{U}_1^T\boldsymbol{X} \tag{156}$$

18 Representative-based Clustering

18.1 General

- Given a dataset with n points in a d dimensional space, $\mathbf{D} = \{\boldsymbol{x}_1\}_{i=1}^n$ and given the number of desired clusters k, the goal of **representative-based clustering** is to divide the dataset into k groups or clusters
  - This is called a **clustering**
  - It is denoted as $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$
  - For each cluster C_i there exists a representative point that summarizes the cluster
    - A common choice being the mean (centroid) $\boldsymbol{\mu}_i$ of all points in the clusters i.e.

$$\boldsymbol{\mu}_i = \frac{1}{n_i}\sum_{x_j \in C_i} \boldsymbol{x}_j \tag{157}$$

- where $n_i = |C_i|$
- A brute-force algorithm for finding a good clustering is simply to generate all possible partitions of n points into k clusters

– A optimization score is evaluated for each of them an the clustering with the best score is used
– The number of ways of partitioning n points into k nonempty and disjoint parts is given by the Stirling number of the second kind given as

$$S(n,k) = \frac{1}{k!}\sum_{t=1}^{k}(-1)^t\binom{k}{t}(k-t)^n \tag{158}$$

18.2 K-Means Algorithm

ALGORITHM 13.1. K-means Algorithm

K-MEANS ($\mathbf{D}, k, \epsilon$):
$t = 0$
Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$
repeat
  $t \leftarrow t+1$
  $C_j \leftarrow \emptyset$ for all $j = 1, \cdots, k$
  `// Cluster Assignment Step`
  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**
    $j^* \leftarrow \arg\min_i \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i^{t-1} \right\|^2 \right\}$ `// Assign` $\mathbf{x}_j$ `to closest centroid`
    $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$
  `// Centroid Update Step`
  foreach $i = 1$ *to* k **do**
    $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|}\sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$
until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

- Given a clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ a the **sum of squared error** scoring function is defined as

$$SSE(\mathcal{C})\sum_{i=1}^{k}\sum_{\boldsymbol{x}_j \in C_i} ||\boldsymbol{x}_j - \boldsymbol{\mu}_i||^2 \tag{159}$$

- The goal is to finding the clustering that minimizes the SSE score

$$C^* = \arg\min_{\mathcal{C}}\{SSE(\mathcal{C})\} \tag{160}$$

- K-means employs a greedy iterative approach to find a clustering that minimizes the SSE objective

- It can converge to a local optima instead of a globally optimal clustering
- K-means initializes the clusters by randomly generating k points in the data space
  - Typically done by generating a value uniformly at random within the range for each dimension
- Each iteration of K-means consists of two steps
  1. Cluster assignment: Each point $\boldsymbol{x}_j \in \mathbf{D}$ is assigned to the closest mean
     - This induces a clustering with each cluster C_i compromising points that are closer to $\boldsymbol{\mu}_i$
     - Each point $\boldsymbol{x}_j$ is assigned to cluster C_{j^*} where $j^* = \arg\min_{i=1}^{k}\{||\boldsymbol{x}_j - \boldsymbol{m}u_i||^2\}$
  2. Centroid update: Given a set of clusters $C_i, i = 1, \ldots, k$ new mean values are computed for each cluster from the points in C_i
- The two steps are carried out iteratively until we reach a fixed point or local minima
  - One can assume that K-means has converged if the centroids do not change from one iteration to the next
- Since the method starts with a random guess it is typically run several times
  - The run with the lowest SSE is chosen to report the final clustering
- It generates convex shaped clusters
- The total time for K-means is given as $O(tnkd)$
  - Where t is the number of iteration
  - d is the number of dimensions

18.3 Expectation-Maximization Clustering

18.3.1 General

- The K-means approach is an example of a **hard assignment clustering**
  - Each point can belong to only one cluster

- **Soft assignment cluster** is where each point has a probability of belonging to each cluster
- Let $\mathbf{D}$ consist of n points $\boldsymbol{x}_j$ in a d-dimensional space $\mathbb{R}^d$
  - Let X_a denote the random variable corresponding to the a'th attribute
  - X_a is also used to denote the a'th column vector corresponding to the n data sample from X_a
  - Let $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ denote the vector random variable across the d attributes, with $\boldsymbol{x}_j$ being a data sample from $\mathbf{X}$
- EM does not always convex

18.3.2 Gaussian Mixture Model

- We assume that each cluster C_i is characterized by the multivariate normal distribution i.e.

$$f_i(\mathbf{x}) = f(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}_i|^{\frac{1}{2}}} \exp\left\{-\frac{(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)}{2}\right\}$$

- where the clusters mean $\boldsymbol{\mu}_i \in \mathbb{R}^d$ and the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$ are both unknown parameters
  - $f_i(\boldsymbol{x})$ is the probability density at $\boldsymbol{x}$ attributable to cluster C_i
  - It is assumed that the probability density function of $\mathbf{X}$ is given as a Gaussian mixture model over all the k cluster normals, defined as

$$f(\mathbf{x}) = \sum_{i=1}^{k} f_i(\mathbf{x})P(C_i) = \sum_{i=1}^{k} f(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)P(C_i)$$

- Here the prior probabilities $P(C_i)$ are called the **mixture parameters** and must satisfy the condition

$$\sum_{i=1}^{k} P(C_i) = 1 \tag{161}$$

- The Gaussian mixture model is characterized by
  - The mean $\boldsymbol{\mu}_i$
  - The covariance matrix $\boldsymbol{\Sigma}_i$
  - The mixture probability $P(C_i)$ for each of the k normal distributions
  - The set of all model parameters are written compactly as

$$\boldsymbol{\theta} = \left\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, P(C_i) \ldots, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, P(C_k)\right\}$$

18.3.3 Maximum Likelihood Estrimation

- Given a data $\mathbf{D}$ the likelihood of θ is defined as the conditional probability of the data $\mathbf{D}$ given the model parameters θ
  - It is denoted as $P(\mathbf{D} \mid \theta)$
  - Since each of the n points $\boldsymbol{x}_j$ is considered to be random sample from $\mathbf{X}$ the likelihood of θ is given as

$$P(\mathbf{D} \mid \theta) = \prod_{j=1}^{n} f(\boldsymbol{x}_j) \tag{162}$$

- The goal of MLE is to maximize the parameters θ
  - It typically done on the log-likelihood function
  - The log-likelihood function is given as

$$\ln P(\mathbf{D}|\boldsymbol{\theta}) = \sum_{j=1}^{n} \ln f(\mathbf{x}_j) = \sum_{j=1}^{n} \ln\left(\sum_{i=1}^{k} f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) P(C_i)\right)$$

- Since maximizing the log-likelihood over θ directly is hard expectation maximization (EM) is used instead
  - It is an approach for finding maximum likelihood estimates for the parameters θ
  - It is a two step iterative approach that starts for an initial guess for the parameters $\boldsymbol{\theta}$

– Given the current estimates for θ the expectation step EM computes the cluster posterior probabilities $P(C_i \mid \boldsymbol{x}_j)$ via the Bayers theorem

$$P(C_i|\mathbf{x}_j) = \frac{P(C_i \text{ and } \mathbf{x}_j)}{P(\mathbf{x}_j)} = \frac{P(\mathbf{x}_j|C_i)P(C_i)}{\sum_{a=1}^{k} P(\mathbf{x}_j|C_a)P(C_a)}$$

- Since the cluster is modeled as a multivariate normal distribution the probability of $\boldsymbol{x}_j$ given cluster C_i can be obtained by considering a small interval $\epsilon > 0$ centered at $\boldsymbol{x}_j$ as follows

$$P(\mathbf{x}_j|C_i) \simeq 2\epsilon \cdot f(\mathbf{x}_j|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = 2\epsilon \cdot f_i(\mathbf{x}_j)$$

The posterior probability of C_i is thus given as

$$P(C_i|\mathbf{x}_j) = \frac{f_i(\mathbf{x}_j) \cdot P(C_i)}{\sum_{a=1}^{k} f_a(\mathbf{x}_j) \cdot P(C_a)}$$

- and $P(C_i \mid \boldsymbol{x}_j)$ can be considered as weight or contribution of the point $\boldsymbol{x}_j$ to cluster C_i
- In the maximization step using the weight $P(C_i \mid \boldsymbol{x}_j)$ EM re-estimates $\boldsymbol{\theta}$
  - i.e. it re-estimates the parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$ and $P(C_i)$ for each cluster C_i
  - The re-estimated means is given as the weighted average of all the points
  - The re-estimated covariance matrix is given as the weighted covariance over all pairs of dimensions
  - The re-estimated prior probability for each is given as the fraction of the weights that contribute to that cluster

18.3.4 EM in one Dimension

1. General
   - A dataset $\mathbf{D}$ is considered which consists of a single attribute X

- Each point $x_j \in \mathbb{R}$ for $j = 1, \ldots, n$ is a random sample from X
- For the mixture model univariate normals it used for each cluster

$$f_i(x) = f(x|\mu_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right\}$$

- with the cluster parameters μ_i, σ_i^2 and $P(C_i)$
- The EM approach consists of three steps initialization, expectation and maximization

2. Initialization

- For each cluster C_i with $i = 1, 2, \ldots, k$ we randomly initialize the cluster parameters μ_i, σ_i^2 and $P(C_i)$
- The mean μ_i is selected uniformly at random from the range of possible values for X
- It is typical to assume that the initial variance is give as σ_i^2
- The cluster probabilities are initialized to $P(C_i) = \frac{1}{k}$

3. Expectation Step

- It is assumed that for each of the k clusters that we have an estimate for the parameters
  - i.e. μ_i, σ_i^2 and $P(C_i)$
- Given the estimated values the posterior probabilities are computed

$$P(C_i|x_j) = \frac{f(x_j|\mu_i, \sigma_i^2) \cdot P(C_i)}{\sum_{a=1}^{k} f(x_j|\mu_a, \sigma_a^2) \cdot P(C_a)}$$

- The notation $w_{ij} = P(C_i \mid x_k)$ is used
- Let

$$\boldsymbol{w}_i = (w_{i1}, \ldots, w_{in})^T \tag{163}$$

- denote the weight vector for cluster i across all the n points

4. Maximization Step

- It is assumed that all posterior probability values or weights $w_{ij} = P(C_i \mid x_k)$ are known
- It computes the ML estimates for the cluster parameters
- The re-estimated value for the cluster mean μ_i is computed as the weighted mean of all the points

$$\mu_i = \frac{\sum_{j=1}^{n} w_{ij} \cdot x_j}{\sum_{j=1}^{n} w_{ij}}$$

- In terms of the weight vector $\boldsymbol{w}_i$ and the attribute vector $X = (x_1, x_2, \ldots, x_n)^T$ it can be written as

$$\mu_i = \frac{\mathbf{w}_i^T X}{\mathbf{w}_i^T \mathbf{1}}$$

- The re-estimated value of the cluster variance is computed as the weighted variance across all the points

$$\sigma_i^2 = \frac{\sum_{j=1}^{n} w_{ij}(x_j - \mu_i)^2}{\sum_{j=1}^{n} w_{ij}}$$

- If we let $Z_i = X - \mu_i \mathbf{1} = (x_1 - \mu_i, x_2 - \mu_i, \ldots, x_n - \mu_i)^T = (z_{i1}, z_{i2}, \ldots, z_{in})^T$ be the centered attribute vector for cluster C_i
  - Let Z_i^s be the squared vector given as $Z_i^s = (z_{i1}^2, z_{i2}^2, \ldots, z_{in}^2)^T$
  - Then the variances can be expressed as

$$\sigma_i^2 = \frac{\mathbf{w}_i^T Z_i^s}{\mathbf{w}_i^T \mathbf{1}}$$

- The prior probability of cluster C_i is re-estimated as the fraction of total weight belonging to C_i computed as

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{\sum_{a=1}^{k}\sum_{j=1}^{n} w_{aj}} = \frac{\sum_{j=1}^{n} w_{ij}}{\sum_{j=1}^{n} 1} = \frac{\sum_{j=1}^{n} w_{ij}}{n} \qquad ($$

- This can be written in vector notation as

$$P(C_i) = \frac{\boldsymbol{w}_i^T \mathbf{1}}{n} \tag{164}$$

5. Iteration

   - Starting for an initial set of values for the cluster parameters the EM algorithm applies the expectation step to compute the weights $w_{ij} = P(C_i \mid x_k)$
   - The weights are used in the maximization step to compute the updated cluster parameters
   - The expectation and maximization steps are iteratively applied until convergence
     - e.g. until there is very little change in the means

18.3.5 EM in d Dimensions

1. General

   - The EM methods in now considered in d dimensions
     - Each cluster is characterized by a multivariate normal distribution with parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}$ and $P(C_i)$
     - For each cluster C_i a d dimension mean vector should be estimated

$$\boldsymbol{\mu}_i = (\mu_{i1}, \mu_{i2}, \ldots, \mu_{id})^T \tag{165}$$

   - and the $d \times d$ covariance matrix

$$\boldsymbol{\Sigma}_i = \begin{pmatrix} (\sigma_1^i)^2 & \sigma_{12}^i & \ldots & \sigma_{1d}^i \\ \sigma_{21}^i & (\sigma_2^i)^2 & \ldots & \sigma_{2d}^i \\ \vdots & \vdots & \ddots & \\ \sigma_{d1}^i & \sigma_{d2}^i & \ldots & (\sigma_d^i)^2 \end{pmatrix}$$

- Since the covariance matrix is symmetric, $\binom{d}{2} = \frac{d(d-1)}{2}$ pairwise covariances and d variances for a total of $\frac{d(d+1)}{2}$ parameters for $\mathbf{\Sigma}_i$
  - It may be too many parameters to estimate them reliably
  - A simplification is to assume that all dimensions are independent, which lead to a diagonal covariance matrix

$$\mathbf{\Sigma}_i = \begin{pmatrix} (\sigma_1^i)^2 & 0 & \dots & 0 \\ 0 & (\sigma_2^i)^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \\ 0 & 0 & \dots & (\sigma_d^i)^2 \end{pmatrix}$$

- Under this assumption d parameters only needs to be estimated for $\mathbf{\Sigma}_i$

2. Initialization
   - For each cluster C_i with $i = 1, 2, \dots, k$ we randomly initialize the mean μ_i by selecting a value μ_{ia} for each dimension X_a uniformly at random from the range X_a
   - The covariance matrix is initialized as the $d \times d$ identity matrix
   - The cluster prior probabilities are initialized to $P(C_i) = \frac{1}{k}$
3. Expectation step
   - The posterior probability of cluster C_i given point $\boldsymbol{x}_j$ is computed using the formula

$$P(C_i|\mathbf{x}_j) = \frac{f_i(\mathbf{x}_j) \cdot P(C_i)}{\sum_{a=1}^{k} f_a(\mathbf{x}_j) \cdot P(C_a)}$$

   - with $i = 1, \dots, k$ and $j = 1, \dots, n$
     - The shorthand $w_{ij} = P(C_i \mid x_k)$ is used
     - The notation $\boldsymbol{w}_i = (w_{i1}, \dots, w_{in})^T$ is used
4. Maximization Step

- Given the weights the parameters $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}$ and $P(C_i)$ are re-estimated
- The mean μ_i for cluster C_i can be estimated as

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}}$$

- This can be expressed in matrix form as

$$\boldsymbol{\mu}_i = \frac{\mathbf{D}^T \mathbf{w}_i}{\mathbf{w}_i^T \mathbf{1}}$$

- Let $\mathbf{Z}_i = \mathbf{D} - \mathbf{1} \cdot \mu_i^T$ be the centered data matrix for the cluster C_i
  - Let $z_{ji} = \boldsymbol{x}_j - \boldsymbol{\mu}_i \in \mathbb{R}^d$ denote the j'th centered point in $\mathbf{Z}_i$
  - $\boldsymbol{\Sigma}_i$ can be described compactly using the following equation

$$\boldsymbol{\Sigma}_i = \frac{\sum_{j=1}^{n} w_{ij} \mathbf{z}_{ji} \mathbf{z}_{ji}^T}{\mathbf{w}_i^T \mathbf{1}}$$

- The covariance between dimension X_a and X_b is estimated as

$$\sigma_{ab}^i = \frac{\sum_{j=1}^{n} w_{ij}(x_{ja} - \mu_{ia})(x_{jb} - \mu_{ib})}{\sum_{j=1}^{n} w_{ij}}$$

- The prior probability $P(C_i)$ for each cluster is the same as in the one-dimensional case given as

$$P(C_i) = \frac{\sum_{j=1}^{n} w_{ij}}{n} = \frac{\mathbf{w}_i^T \mathbf{1}}{n}$$

5. EM Clustering Algorithm

ALGORITHM 13.3. Expectation-Maximization (EM) Algorithm

EXPECTATION-MAXIMIZATION $(\mathbf{D}, k, \epsilon)$:

$t \leftarrow 0$

// Initialization

Randomly initialize $\boldsymbol{\mu}_1^t, \ldots, \boldsymbol{\mu}_k^t$

$\boldsymbol{\Sigma}_i^t \leftarrow \mathbf{I},\ \forall i = 1, \ldots, k$

$P^t(C_i) \leftarrow \frac{1}{k},\ \forall i = 1, \ldots, k$

repeat

  $t \leftarrow t + 1$

  // Expectation Step

  for $i = 1, \ldots, k$ *and* $j = 1, \ldots, n$ **do**

    $w_{ij} \leftarrow \frac{f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \cdot P(C_i)}{\sum_{a=1}^{k} f(\mathbf{x}_j | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) \cdot P(C_a)}$ // posterior probability $P^t(C_i | \mathbf{x}_j)$

  // Maximization Step

  for $i = 1, \ldots, k$ **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{\sum_{j=1}^{n} w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^{n} w_{ij}}$ // re-estimate mean

    $\boldsymbol{\Sigma}_i^t \leftarrow \frac{\sum_{j=1}^{n} w_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^{n} w_{ij}}$ // re-estimate covariance matrix

    $P^t(C_i) \leftarrow \frac{\sum_{j=1}^{n} w_{ij}}{n}$ // re-estimate priors

until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\|^2 \leq \epsilon$

- The total time for the EM clustering algorithm is $O(t(kd^3 + nkd^2))$
- The total time for the EM clustering algorithm with a diagonal covariance matrix is $O(tnkd)$

19 Density-based Clustering

19.1 The DBSCAN Algorithm

- Density-based clustering uses the local density of points to determine the clusters
  - A ball of radius ϵ is defined around a point $\boldsymbol{x} \in \mathbb{R}^d$ called the ϵ neighborhood of $\boldsymbol{x}$ as follows

$$N_\epsilon(\boldsymbol{x}) B_d(\boldsymbol{x}, \epsilon) = \{\boldsymbol{y} \mid \delta(\boldsymbol{x}, \boldsymbol{y}) \leq \epsilon\} \tag{166}$$

- $\delta(\boldsymbol{x}, \boldsymbol{y})$ represents the distance between points $\boldsymbol{x}$ and $\boldsymbol{y}$
  - It is usually assumed to be the Euclidean distance that it $\delta(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_2$

- For any point $\boldsymbol{x} \in \mathbf{D}$ we say that $\boldsymbol{x}$ is a core point if there are at least *minpts* point in its ϵ neighborhood
  - i.e. $\boldsymbol{x}$ is a core point if $|N_\epsilon(\boldsymbol{x})| \geq minpts$ where *minpts* is a user defined local density
  - A border point is defined as a point that does not meat the *minpts* threshold but belongs to the ϵ neighborhood of some core point $\boldsymbol{z}$
    - i.e. $|N_\epsilon(\boldsymbol{x})| < minpts$ and \$x $\in x \in N_\epsilon(\boldsymbol{z})$
  - If a point is neither a core not a border point, then ti is called a noise point or an outlier
  - We say that a point $\boldsymbol{x}$ is **directly density reachable** from another point $\boldsymbol{y}$ if $\boldsymbol{x} \in N_\epsilon(\boldsymbol{y})$ and $\boldsymbol{y}$ is a core point
    - We say that $\boldsymbol{x}$ is **density reachable** if their exists a chain of point $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_l$ such that $\boldsymbol{x} = \boldsymbol{x}_0$ and $\boldsymbol{y} = \boldsymbol{x}_i$ and $\boldsymbol{x}_i$ is directly density reachable from x_{i-1} for all $i = 1, \ldots, l$
    - Density reachability is an asymmetric relationship
    - Two points $\boldsymbol{x}$ and $\boldsymbol{y}$ is density connected if there exists a core point $\boldsymbol{z}$ such that both $\boldsymbol{x}$ and $\boldsymbol{y}$ are density reachable from $\boldsymbol{z}$
    - A **density-based cluster** is defined as the maximal set of density connected points

ALGORITHM 15.1. Density-based Clustering Algorithm

```
DBSCAN (D, ϵ, minpts):
Core ← ∅
foreach x_i ∈ D do // Find the core points
    Compute N_ϵ(x_i)
    id(x_i) ← ∅ // cluster id for x_i
    if N_ϵ(x_i) ≥ minpts then Core ← Core ∪ {x_i}
k ← 0 // cluster id
foreach x_i ∈ Core, such that id(x_i) = ∅ do
    k ← k + 1
    id(x_i) ← k // assign x_i to cluster id k
    DensityConnected (x_i, k)
C ← {C_i}_{i=1}^k, where C_i ← {x ∈ D | id(x) = i}
Noise ← {x ∈ D | id(x) = ∅}
Border ← D \ {Core ∪ Noise}
return C, Core, Border, Noise
DensityConnected (x, k):
foreach y ∈ N_ϵ(x) do
    id(y) ← k // assign y to cluster id k
    if y ∈ Core then DensityConnected (y, k)
```

- DBSCAN computes the ϵ neighborhood $N_\epsilon(\boldsymbol{x}_i)$ for each point $\boldsymbol{x}_i$ in the data set $\mathbf{D}$ and checks if it is a core point
  - It sets the cluster id $id(\boldsymbol{x}_i) = \emptyset$ for all points
    - Indicates that they have not been assigned to any cluster
  - Starting from each unassigned core point, the method recursively finds all its density connected points, which are assigned to the same cluster
  - Some border point may be reachable from core points in more than one cluster
  - They may be assigned to one of the cluster or all of them - if overlapping clusters are allowed
- A limitation of DBSCAN is that is sensitive to the choice of ϵ
  - If ϵ is too small, sparser clusters will be categorized as noise
  - If ϵ is too large denser clusters may be merge together
  - If there are clusters with different local densities a single ϵ value may not suffice

- If the dimensionality is not too high the DBSCAN takes $O(n \cdot \log(n))$ time
  - Worst case it takes $O(n^2)$ time

19.2 Kernel Density Estimation

19.2.1 General

- There is a close connection between density-based clustering and density estimation
  - The goal of density estimation is to find an unknown probability density function by finding the dense regions of points which in turn can be used for clustering
  - It is a nonparametric technique that does not assume any fixed probability model of the clusters
  - It tries to directly infer the underlying probability density at each point in the dataset

19.2.2 Univariate Density Estimation

- Assume that X is a continuous random variable, and let $x_1, x_2, \ldots, x_n$ be a random sample drawn from the underlying probability density function $f(x)$ which is assumed to be unknown
  - The cumulative distribution function can directly estimated by couting how many points are less that or equal to x

$$\hat{F}(x) = \frac{1}{n} \sum_{i=1}^{n} I(x_i \leq x) \tag{167}$$

- I is the indicator function that has value 1 only when its arguments is true and 0 otherwise
  - The density function can be estimated by taking the derivative of $\hat{F}(x)$ by considering a windows of small width h centered at x i.e.

$$\hat{f}(x) = \frac{\hat{F}\left(x + \frac{h}{2}\right) - \hat{F}\left(x - \frac{h}{2}\right)}{h} = \frac{k/n}{h} = \frac{k}{nh}$$

- where k is the number of points that lie in the windows of width h centered at x i.e. within the closed interval $[x - \frac{h}{2}, x + \frac{h}{2}]$
  - The density estimate is the ration of the fraction of the points in the window k/n to the volume of the window h
  - h plays the role of influence
    * A large h estimates the probability density over a large window which is a smoother estimate
    * If h is small then only the proximity to x are considered
    * h should be small but not to small
- **Kernel density** estimation relies on a kernel function K that is non-negative, symmetric and integrates to 1
  - The following should be true for the kernel function $K(x) \geq 0$, $K(-x) = K(x)$ for all values x and $\int K(x)dx = 1$
  - K is essentially a probability density function
- **Discrete Kernel** The density estimate $\hat{f}(x)$ can be rewritten in terms of the kernel function as follows

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

- where the **discrete kernel** function k computes the number of points in a window of width h defined as

$$K(z) = \begin{cases} 1 & \text{If } |z| \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{168}$$

- If $|z| = |\frac{x - x_i}{h}| \leq \frac{1}{2}$ then the point is within a window of width h centered at x
- **Gaussian Kernel** is a more smooth kernel than the discrete kernel where there is a more smooth transition of influence

$$K(z) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{z^2}{2}\right\}$$

- Thus one have

$$K\left(\frac{x-x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{(x-x_i)^2}{2h^2}\right\}$$

- x plays the role of the mean and h acts as the standard derivation

19.2.3 Multivariate Density Estimation

- To estimate the probability density at a d dimensional point $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T$ a d dimensional windows is defined
  - i.e. a hypercube centered at $\boldsymbol{x}$ with edge length h
  - The volume of the hypercube is given as

$$\text{vol}(H_d(h)) = h^d \tag{169}$$

- The density is estimated as the fraction of the point weight which lies within the d dimensional window centered at $\boldsymbol{x}$ divided by the volume of the hypercube

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

- where the multivariate kernel function K satisfies the condition $\int K(\boldsymbol{z})d\boldsymbol{z} = 1$
- **Discrete Kernel** For any d dimensional vector $\boldsymbol{z} = (z_1, z_2, \ldots z_d)^T$, the discrete kernel in a function in$d$ dimensions given as

$$K(z) = \begin{cases} 1 & \text{If } |z| \leq \frac{1}{2} \text{ for all dimensions } j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases} \tag{170}$$

- For $\boldsymbol{z} = \frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}$
  - Each point within the hypercube contributes a weight of $\frac{1}{n}$ to the density estimate
- **Gaussian Kernel** The d dimensional Gaussian kernel is given as

$$K(\mathbf{z}) = \frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{\mathbf{z}^T\mathbf{z}}{2}\right\}$$

- It is assumed that the covariance matrix is the $d \times d$ identity matrix.
  - Setting $\boldsymbol{z} = \frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}$ we have

$$K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) = \frac{1}{(2\pi)^{d/2}} \exp\left\{-\frac{(\mathbf{x}-\mathbf{x}_i)^T(\mathbf{x}-\mathbf{x}_i)}{2h^2}\right\}$$

- Each point contributes a weight to the density estimate inversely proportional to its distance from $\boldsymbol{x}$

19.2.4 Nearest Neighbor Density Estimation

- Another way to do density estimation is to fix a k which is the number of points required to estimate the density and allow the volume of the enclosing region to vary to accommodate those k points
  - This approach is called the k neighbor approach to density estimation
  - It is a nonparametric approach
  - Given k, the number of neighbors we estimate the density at $\boldsymbol{x}$ as follows

$$\hat{f}(\boldsymbol{x}) = \frac{k}{n\text{vol}(S_d(h_{\boldsymbol{x}}))} \tag{171}$$

- Where $h_{\boldsymbol{x}}$ us the distance from $\boldsymbol{x}$ to its kth nearest neighbor and $\text{vol}(S_d(h_{\boldsymbol{x}}))$ is the volume of the d dimensional hypersphere $S_d(h_{\boldsymbol{x}})$ centered at $\boldsymbol{x}$ with radius $h_{\boldsymbol{x}}$

19.3 Density-Based Clustering: DENCLUE

- A point $\boldsymbol{x}^*$ is called a **density attractor** if it is a local maxima of the probability density function f
  - It can be found via a gradient ascent approach starting at some point $\boldsymbol{x}$

- The idea is to compute the density gradient, the direction of the largest increase in density and to most in the direction of the gradient in small steps until we reach a local maxima
- The gradient at a point $\boldsymbol{x}$ can be computed as the multivariate derivative of the probability density estimate given as

$$\nabla \hat{f}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} \hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{x}} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

- For the Gaussian kernel we have

$$\nabla \hat{f}(\mathbf{x}) = \frac{1}{nh^{d+2}} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \cdot (\mathbf{x}_i - \mathbf{x})$$

- We say that $\boldsymbol{x}^*$ is a **density attractor** for $\boldsymbol{x}$ or $\boldsymbol{x}$ is density attracted to $\boldsymbol{x}^*$ if the hill climing process started at $\boldsymbol{x}$ converges to $\boldsymbol{x}^*$
  - That is there exists a sequence of points $\boldsymbol{x} = \boldsymbol{x}_0 \to \boldsymbol{x}_1 \to \cdots \to \boldsymbol{x}_m$ starting from $\boldsymbol{x}$ and ending at $\boldsymbol{x}_m$ such that $||\boldsymbol{x}_m - \boldsymbol{x}^*|| \leq \epsilon$
  - The typically approach is to use the gradient-ascent method to compute $\boldsymbol{x}^*$ starting from $by iteratively update it at each step t$ via the update rule

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \delta \cdot \nabla \hat{f}(\boldsymbol{x}_t) \tag{172}$$

- where $\delta > 0$ is the step size
- Instead of using gradient-ascent the direct update rule can be used

$$\mathbf{x}_{t+1} = \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right) \mathbf{x}_i}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right)}$$

- where t denotes the current iteration and $\boldsymbol{x}_{t+1}$ is updated value for the current vector $\boldsymbol{x}_t$
  - It is much faster to converge than the hill-climbing process

- A cluster $C \subseteq \mathbf{D}$ is called a **center-defined cluster** if all the points $\boldsymbol{x} \in C$ are density attracted to a unique density attractor $\boldsymbol{x}^*$, such that $\hat{f}(\boldsymbol{x}^*) \geq \xi$, where ξ is a user defined minumum
  - In other words the following must hold

$$\hat{f}(\mathbf{x}^*) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x}^* - \mathbf{x}_i}{h}\right) \geq \xi$$

- An arbitrary-shaped cluster $C \subseteq \mathbf{D}$ is called a **density-based cluster** if there exists a set of attractors $\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_m^*$, such that
  1. Each points $\boldsymbol{x} \in C$ is attracted to some attractor $\boldsymbol{x}_i^*$
  2. Each density attractor has density above ξ i.e. $\hat{f}(\boldsymbol{x}_i^*) \geq \xi$
  3. Any two density attractors $\boldsymbol{x}_i^*$ and $\boldsymbol{x}_j^*$ are density reachable i.e. there exists a path from $\boldsymbol{x}_i^*$ to $\boldsymbol{x}_j^*$ such that for all points $\boldsymbol{y}$ on the path, $\hat{f}(\boldsymbol{y}) \geq \xi$

ALGORITHM 15.2. DENCLUE Algorithm

DENCLUE $(\mathbf{D}, h, \xi, \epsilon)$:
$\mathcal{A} \leftarrow \emptyset$
foreach $\mathbf{x} \in \mathbf{D}$ **do** // find density attractors
  $\mathbf{x}^* \leftarrow$ FINDATTRACTOR$(\mathbf{x}, \mathbf{D}, h, \epsilon)$
  if $\hat{f}(\mathbf{x}^*) \geq \xi$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}^*\}$
    $R(\mathbf{x}^*) \leftarrow R(\mathbf{x}^*) \cup \{\mathbf{x}\}$
$\mathcal{C} \leftarrow \{\text{maximal } C \subseteq \mathcal{A} \mid \forall \mathbf{x}_i^*, \mathbf{x}_j^* \in C, \mathbf{x}_i^* \text{ and } \mathbf{x}_j^* \text{ are density reachable}\}$
foreach $C \in \mathcal{C}$ **do** // density-based clusters
  foreach $\mathbf{x}^* \in C$ **do** $C \leftarrow C \cup R(\mathbf{x}^*)$
return $\mathcal{C}$

FINDATTRACTOR $(\mathbf{x}, \mathbf{D}, h, \epsilon)$:
$t \leftarrow 0$
$\mathbf{x}_t \leftarrow \mathbf{x}$
repeat
  $\mathbf{x}_{t+1} \leftarrow \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right) \cdot \mathbf{x}_t}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right)}$
  $t \leftarrow t + 1$
until $\|\mathbf{x}_t - \mathbf{x}_{t-1}\| \leq \epsilon$
return $\mathbf{x}_t$

- **DENCLUE Algorithm**
  - The first step is to compute the density attractor $\boldsymbol{x}^*$ for each point $\boldsymbol{x}$ in the dataset
    - If the density at $\boldsymbol{x}^*$ is above the minimum density threshold ξ the attractor is added to the set of attractors $\mathcal{A}$
    - The data point $\boldsymbol{x}$ is also added to the set of points $R(\boldsymbol{x}^*)$ attracted to $\boldsymbol{x}^*$
  - In the second step DENCLUE finds all the maximal subsets of attractors $C \subseteq \mathcal{A}$ such that any pair of attractors in C in density reachable from each other
    - These form the seed for each density-based clusters
  - Lastly for each attractor $\boldsymbol{x}^* \in C$ we add the cluster all of the points $R(\boldsymbol{x}^*)$ that are attracted to $\boldsymbol{x}^*$
    - This results in the final set of clusters $\mathcal{C}$
  - The `FINDATTRACTOR` method implements the hill-climbing process using the direct update rule
    - Results in fast convergence

20 Clustering Validation

20.1 General

- Cluster validation and assessment encompasses three main tasks
  - **Clustering evaluation** seeks to assess the quality of the clustering
  - **Clustering stability** seeks to understand sensitive the clustering is to the algorithmic parameters
    - e.g. the number of cluster
  - **Clustering tendency** assess the suitability of applying clustering in the first placer
    - i.e. whether the data has any inherent grouping struictire
- Validity measures can be divided into three main types:
  - **External:** External validation measure employ criteria the are not in the dataset

    - ∗ It can be in the from of prior or expert-specified knowledge
    - ∗ e.g. class labels for each point
  - **Internal:** Internal validation measure the employ criteria that are derived from the data itself
  - **Relative:** Relative validation compare different clusterings
    - ∗ It is usually other clusters obtained via different parameter settings for the same algorithm

20.2 External Measures

20.2.1 General

- External measures assume that the correct clustering is known a priori
  - The true cluster labels play the role of external information which is used to evaluate the given clustering
  - In general one would not know the correct cluster
  - External measures can server as way to test and validate different measures
- Let $\mathbf{D} = \{\boldsymbol{x}_i\}_{i=1}^n$ be the dataset consisting of n points in a d dimensional space which is partitioned into k clusters
  - Let $y_i \in \{1, 2, \ldots, k\}$ denote the ground-truth cluster membership
  - The groud truth clustering is given as $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ where the cluster T_j consists of all the points with label j
    - ∗ i.e. $T_j = \{\boldsymbol{x}_i \in \mathbf{D} \mid y_i = j\}$
  - Let $\mathcal{C} = \{C_1, \ldots, C_r\}$ denote a clustering of the same dataset into r clusters which is obtained via some algorithm
  - Let $\hat{y}_i \in \{1, 2, \ldots, r\}$ denote the cluster label for $\boldsymbol{x}_i$
  - $\mathcal{T}$ is refereed to as the ground-truth partitioning and each T_i as a partition
  - $\mathcal{C}$ is called as clustering with each C_i called a clustered
  - Since the ground truth is assumed to be known the algorithm is used with the right number of parameters $r = k$
    - ∗ But one would typically allow them to be different for generality

- External evaluation measure try to capture the extent to which points from same partition appear in the same cluters and different partition are grouped in different clusters
    - There are usually a trade of between the two goal, which is either explicitly captured by a measure or is implicit in its computation
- All of the external measures rely on the $r \times k$ contingency table $\mathbf{N}$ that is induced by a clustering $\mathcal{C}$ and the ground-truth partitioning $\mathcal{T}$, defined as follows

$$\mathbf{N}(i,j) = n_{ij} = |C_i \cap T_j| \tag{173}$$

- The count n_{ij} denotes the number of points that are common to cluster C_i and group-truth partition T_j
    - Let $n_i = |C_i|$ denote the number of points in cluster C_i
    - Let $m_i = |T_j|$ denote the number of points in partition T_j
    - The contingency table can be computed from $\mathcal{T}$ and $\mathcal{C}$ in $O(n)$ time by examining the partition and cluster label y_i and $\hat{y}_i$ for each point $\boldsymbol{x}_i \in \mathbf{D}$ and incrementing the corresponding count $\$n_{y_i}$

y_i

20.2.2 Matching Based Measures

1. Purity
    - **Purity** quantifies the extent to which a cluster C_i contains entities from only one partition
        - How "pure" each cluster is
        - The purity of cluster C_i is defined as

$$purity_i = \frac{1}{n_i} \max_{j=1}^{k} \{n_{ij}\}$$

    - The purity of clutering $\mathcal{C}$ is defined as the weighted sum of the clusterwise purity values

$$purity = \sum_{i=1}^{r} \frac{n_i}{n} purity_i = \frac{1}{n} \sum_{i=1}^{r} \max_{j=1}^{k} \{n_{ij}\}$$

- The maximum value of purity is 1
  - When each clusters comprises points from only one partition
  - When $r = k$ a this indicates a perfect clustering
  - When $r > k$ the each of the clusters is a subset of the ground-truth partitioning
  - When $r < k$ purity can never be one

2. Maximum Matching

- The maximum matching measure selects the mapping between clusters and partitions, such that the sum of common points n_{ij} is maximized
  - This is provided that only one cluster can match with a given partitioning
  - The contingency table is treat as a complete weighted bipartite graph $G = (V, E)$
    - Each partion and cluster is a node i.e. $V = \mathcal{C} \cup \mathcal{T}$
    - There exists an edge $C_i, T_j \in E$ with weight $w(C_i, T_j) = n_{ij}$ for all $C_i \in \mathcal{C}$ and $T_j \in \mathcal{T}$
    - A matching m in G is a subset of E such that the edges in M are pairwise nonadjacent
  - The maximum measure is defined as the maximum weight matching in G

$$match = \arg\max_{M} \left\{ \frac{w(M)}{n} \right\}$$

- It can be computed in $O(k^4)$ time if $r = O(k)$

3. F-Measure

- Given cluster C_i, let j_i denote the partition that contains the maximum number of points from C_i i.e $j_i = \max_{j=1}^{k} n_{ij}$
  - The precision of a cluster C_i is the same as its purity

$$prec_i = \frac{1}{n_i} \max_{j=1}^{k} \{n_{ij}\} = \frac{n_{ij_i}}{n_i}$$

- It measures the fractions of points in C_i from the majority partition T_{j_i}
- The recall of cluster C_i is defined as

$$recall_i = \frac{n_{ij_i}}{|T_{j_i}|} = \frac{n_{ij_i}}{m_{j_i}}$$

- It measure the fraction of point in partition T_{j_i} shared in common C_i
- The F-measure is the harmonic mean of the precision and recall values for each cluster
- The F-measure for cluster C_i is given as

$$F_i = \frac{2}{\frac{1}{prec_i} + \frac{1}{recall_i}} = \frac{2 \cdot prec_i \cdot recall_i}{prec_i + recall_i} = \frac{2\, n_{ij_i}}{n_i + m_{j_i}}$$

- The F-measure for the clustering $\mathcal{C}$ is the mean of the clusterwise F-measure values

$$F = \frac{1}{r} \sum_{i=1}^{r} F_i$$

- F-measure tries to balance the precision and recall values across all the clusters
  - For a perfect clustering, when $r = k$ the maximum value of the F-measure is 1

20.3 Internal Measures

20.3.1 General

- Internal evaluation measures not use the ground-truth partitioning

- To evaluate the quality of the clustering internal measures utilize notions of intracluster similarity or compactness, contrasted with notions of intracluster separation
  - There are usually a trade-off in maximizing these two aims
- The internal measures are based on the $n \times n$ distance matrix which is called the **proximity matrix**

$$\mathbf{W} = \left\{ \delta(\mathbf{x}_i, \mathbf{x}_j) \right\}_{i,j=1}^{n}$$

- where

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \left\| \mathbf{x}_i - \mathbf{x}_j \right\|_2$$

- Since $\mathbf{W}$ is symmetric and $\delta(\boldsymbol{x}_i, \boldsymbol{x}_i) = 0$ only the upper triangular elements of $\mathbf{W}$ are used in calculations
  - It can also be considered as the adjacency matrix of the weighted complete graph G over the n point s with
    - nodes $V = \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in \mathbf{D}\}$
    - edges $E = \{(x_i, x_j) \mid x_i, x_j \in \mathbf{D}\}$
    - edge weights $w_{ij} = \mathbf{W}(i, j)$ for all $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbf{D}$
- For internal measures it is assumed that there is not access to a ground-truth partioning
  - It is assumed that we are given a clustering $\mathcal{C} = C_1, \ldots, C_k$ of $r = k$ clusters
  - Cluster C_i contains $n_i = |C_i|$ points
  - Let $\hat{y}_i \in \{1, 2, \ldots, k\}$ denote the cluster label for point $\boldsymbol{x}_i$
  - The clustering $\mathcal{C}$ can be considered as a k-way cut in G since $C_i \neq \emptyset$ for all i, $C_i \cap C_j = \emptyset$ for all i, j and ${}_iC_i = V$
    - Given any subsets $S, R \subset V$ define $W(S, R)$ as the sum of the weights on all the edges with one vertex in S and the other in R given as

$$W(S, R) = \sum_{\mathbf{x}_i \in S} \sum_{\mathbf{x}_j \in R} w_{ij}$$

- The internal measures are based on various function over the intracluster and intracluster weights
  - The sum of all the intracluster weights over all clusters is given as

$$W_{in} = \frac{1}{2}\sum_{i=1}^{k} W(C_i, C_i)$$

- The sum of all intercluster weights is given as

$$W_{out} = \frac{1}{2}\sum_{i=1}^{k} W(C_i, \overline{C_i}) = \sum_{i=1}^{k-1}\sum_{j>i} W(C_i, C_j)$$

- The number of distinct intracluster edge, denoted N_{in} and intercluster edges denoted N_{out} are given as

$$N_{in} = \sum_{i=1}^{k}\binom{n_i}{2} = \frac{1}{2}\sum_{i=1}^{k} n_i(n_i - 1)$$

$$N_{out} = \sum_{i=1}^{k-1}\sum_{j=i+1}^{k} n_i \cdot n_j = \frac{1}{2}\sum_{i=1}^{k}\sum_{\substack{j=1 \\ j\neq i}}^{k} n_i \cdot n_j$$

- The total number of distinct pair of points is

$$N = N_{in} + N_{out} = \binom{n}{2} = \frac{1}{2}n(n-1)$$

20.3.2 BetaCV Measure

- The BetaCV measure is the ratio of the mean intracluster distance to the mean intercluster distance

$$BetaCV = \frac{W_{in}/N_{in}}{W_{out}/N_{out}} = \frac{N_{out}}{N_{in}} \cdot \frac{W_{in}}{W_{out}} = \frac{N_{out}}{N_{in}} \frac{\sum_{i=1}^{k} W(C_i, C_i)}{\sum_{i=1}^{k} W(C_i, \overline{C_i})}$$

- The smaller the BetaCV ratio, the better the clustering

20.4 Davies–Bouldin Index

- Let μ_i denote the cluster mean, given as

$$\mu_i = \frac{1}{n_i} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_j \tag{174}$$

- Let σ_{μ_i} denote the spread of the points around the cluster mean, which is given

$$\sigma_{\mu_i} = \sqrt{\frac{\sum_{\mathbf{x}_j \in C_i} \delta(\mathbf{x}_j, \mu_i)^2}{n_i}} = \sqrt{var(C_i)}$$

- where $var(C_i)$ is the total variance of cluster C_i
- The Davis-Bloudin measure for a pair of clusters C_i and C_j is defined as the ratio

$$DB_{ij} = \frac{\sigma_{\mu_i} + \sigma_{\mu_j}}{\delta(\mu_i, \mu_j)}$$

- It measures how compact the clustered are compared to the distance between the clusters means
- The Davies-Bloudin index is defined as

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \{DB_{ij}\}$$

- For each cluster C_i we pick the cluster C_j that yields the largest DB_{ij} ration
  - The smaller the DB value the better the clustering
    - It means the clusters are well separated
    - Each cluster is well represented by its mean

20.5 Silhouette Coefficient

- The silhouette coefficient is a measure of both cohesion and separation of clusters
  - It is based on the difference between the average distance to points in the closest cluster and to points in the same cluster
  - For each point $\boldsymbol{x}_i$ we calculate its silhouette coefficient s_i as

$$s_i = \frac{\mu_{out}^{\min}(\mathbf{x}_i) - \mu_{in}(\mathbf{x}_i)}{\max\left\{\mu_{out}^{\min}(\mathbf{x}_i), \mu_{in}(\mathbf{x}_i)\right\}}$$

- where $\mu_{in}(\boldsymbol{x}_i)$ is the mean distance from $\boldsymbol{x}_i$ to points in its own cluster $\hat{y}_i$:

$$\mu_{in}(\mathbf{x}_i) = \frac{\sum_{\mathbf{x}_j \in C_{\hat{y}_i}, j \neq i} \delta(\mathbf{x}_i, \mathbf{x}_j)}{n_{\hat{y}_i} - 1}$$

- and $\mu_{out}^{min}(\boldsymbol{x}_i)$ is the mean of the distances from $\boldsymbol{x}_i$ to points in the closest cluster

$$\mu_{out}^{\min}(\mathbf{x}_i) = \min_{j \neq \hat{y}_i} \left\{ \frac{\sum_{\mathbf{y} \in C_j} \delta(\mathbf{x}_i, \mathbf{y})}{n_j} \right\}$$

- The s_i value of a point lies in the interval $[-1, 1]$
  - A value close to $+1$ indicates that $\boldsymbol{x}_i$ is much closer to points in its own cluster and is far from other clusters
  - A value close to zero indicates that $\boldsymbol{x}_i$ is close to the boundary between two clusters
  - A value close to -1 indicates that $\boldsymbol{x}_i$ is much closer to another cluster than its own cluster
    - The point may be mis-clustered
- The silhouette coefficient is defined as the mean s_i value across all the points:

$$SC = \frac{1}{n} \sum_{i=1}^{n} s_i \tag{175}$$

- A value close to +1 indicates a good clustering

20.6 Relative Measures

- The silhouette coefficient for each point s_j and the average SC value can be used to estimate the number of clusters in the data

  1. The approach consists of plotting the s_j values in descending order for each cluster, and to note overall SC value for a particular value of k
  2. We can then pick the value k that yields the best clustering with many points having high s_j values within each cluster
     - As well as high values for SC and SC_i

21 Hierarchical Clustering

21.1 General

- Given n points d dimensional space the goal of hierarchical clustering is to create a sequence of nested partitions
  - This can be visualized via a tree or hierarchy of clusters which is called the cluster dendrogram
  - The clusters in the hierarchy range from the fine-grained to the coarse grained
    - The lowest level of the tree (the leaves) consists of each point in its own cluster
    - The highest level (the root) consists of all points in one cluster
    - At some intermediate level one may find meaningful clusters
  - If the user supplies k, as the desired number of clusters it can be chosen at which level there are k clusters
- There are two main algorithmic approaches to get hierarchical clusters
  - **Agglomerative** strategies work in a bottom-up manner
    - They start with each of the n points in a separate cluster

    - They repeatedly merge the most similar pair of clusters until all points are members of the same cluster
  - **Divisive** strategies work in a top-down manner
    - They start with all the points in the same cluster
    - They recursively split the clusters until all points are in separate clusters

21.2 Preliminaries

- Given a dataset $\mathbf{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$, a clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$ is a partition of $\mathbf{D}$
  - Each **cluster** is a set of points $C_i \subseteq \mathbf{D}$, such that the clusters are pairwise disjoint $C_i \cap C_j = \emptyset$ for all $i \neq j$ and $\cup_{i=1}^{k} C_i = \mathbf{D}$
  - A clustering $\mathcal{A} = \{A_1, \ldots, A_r\}$ is said to be **nested** in another clustering $\mathcal{B} = \{B_1, \ldots, B_s\}$ if an only if $r > s$ and for each cluster $A_i \in \mathcal{A}$ there exists a cluster $B_j \in \mathcal{B}$ such that $A_i \subseteq B_j$
  - Hierarchical clustering yields a sequence of n nested partitions $C_1, \ldots, C_n$ ranging from the trivial clustering $C_1 = \{\{\boldsymbol{x}_1\}, \ldots, \{\boldsymbol{x}_n\}\}$ where each point is in a separate cluster to the other trivial clustering $\mathcal{C}_n = \{\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}\}$ where all the points are in one cluster
    - In general the clustering $\mathcal{C}_{t-1}$ are nested in the clustering $\mathcal{C}_t$
  - The cluster dendrogram is a rooted binary tree that captures the nesting structure
    - There is edges between cluster $C_i \in \mathcal{C}_{t-1}$ and cluster $C_j \in \mathcal{C}_t$ if C_i is nested in C_j i.e. if $C_i \subset C_j$
  - The number of different clusterings is $(2n-3)!!$
    - This makes a naive approach of enumerating all possible hierarchical clusterings infeasible

21.3 Agglomerative Hierarchical Clustering

21.3.1 General

ALGORITHM 14.1. Agglomerative Hierarchical Clustering Algorithm

```
AGGLOMERATIVECLUSTERING(D, k):
1 C ← {C_i = {x_i} | x_i ∈ D} // Each point in separate cluster
2 Δ ← {δ(x_i, x_j): x_i, x_j ∈ D} // Compute distance matrix
3 repeat
4     Find the closest pair of clusters C_i, C_j ∈ C
5     C_ij ← C_i ∪ C_j // Merge the clusters
6     C ← (C \ {C_i, C_j}) ∪ {C_ij} // Update the clustering
7     Update distance matrix Δ to reflect new clustering
8 until |C| = k
```

- Several distance measures can be used in the algorithm to compute the distance between any two clusters
  - The between cluster distances are ultimately based on the distance between two points which is typically computed using Euclidean distance defined as
- Agglomerative hierarchical clustering begins with each of the n points in a separate cluster
  - The two closest clusters are repeatedly merged until all points are members of the same cluster
  - Since the number of clusters decreases by one in each step, the process result in n nested clusterings
  - If specified the merging process can be stopped when there are k clusters remaining

$$\delta(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \left(\sum_{i=1}^{d} (x_i - y_i)^2\right)^{1/2}$$

- The computational complexity of hierarchical clustering is $O(n^2 \log n)$

21.3.2 Distance between Clusters

- **Since Link** measure

- Given two clusters C_i and C_j the distance between them denoted $\delta(C_i, C_j)$ is defined as the minimum distance between a point in C_i and a point in C_j

$$\delta(C_i, C_j) = \min\{\delta(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$$

- **Complete Link** measure
  - The distance between two clusters is defined as the maximum distance between a point in C_i and a point in C_j

$$\delta(C_i, C_j) = \max\{\delta(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$$

21.3.3 Updating Distance Matrix

- When two clusters C_i and C_j are merged into C_{ij} the distance matrix should be updated by recomputing the distances from the newly creates clusters to other clusters C_r hvor $r \neq i$ and $r \neq j$
  - The Lance-Williams provides a general equation to recompute the distances for all of the cluster proximity measures given as

$$\delta(C_{ij}, C_r) = \alpha_i \cdot \delta(C_i, C_r) + \alpha_j \cdot \delta(C_j, C_r) + \beta \cdot \delta(C_i, C_j) + \gamma \cdot \left|\delta(C_i, C_r) - \delta(C_j, C_r)\right|$$

- The coefficient α_i, α_j, β, and γ differ from one measure to another
  - Let $n_i = |C_i|$ denote the carnality of cluster C_i the coefficients for the different distance meassures is the given as

Measure	α_i	α_j	β	γ
Single link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Mean distance	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i \cdot n_j}{(n_i+n_j)^2}$	0
Ward's measure	$\frac{n_i+n_r}{n_i+n_j+n_r}$	$\frac{n_j+n_r}{n_i+n_j+n_r}$	$\frac{-n_r}{n_i+n_j+n_r}$	0

22 DBSCAN Revisted

22.1 General

- The DBSCAN algorithm requires $O(n^2)$ for $d \geq 3$ not $O(n \log n)$ time as claimed in the original paper
  - This is independent of the parameters ϵ and $MinPts$ for DBSCAN
  - The running time can be dramatically brought down to $O(n)$ by allowing for slight inaccuracies (regardless of d)
- Let $B(p, \epsilon)$ be the d dimensional ball centered at point p with radius ϵ
  - The distance metric is Euclidan distance
  - It is dense if it covers at least $MinPts$ points of P
  - This is used in DBSCAN to form clusters
    - i.e. the ball is the ϵ neighbor hood
- The DBSCAN problem is to find a unique set C of clusters of P
- One can solve the DBSCAN problem in $O(n \log n)$ time dimensionality $d = 2$

22.2 Geometric results

- **Bichromatic Closest Pair (BCP)**
  - Let P_1, P_2 be to sets of points in $\mathbb{R}^d$ for some constant d

- Set $m_1 = |P_1|$ and $m_2 = |P_2|$
- The goal of the BCP problem is to find a pair of points $(p_1, p_2) \in P_1 \times P_2$ with the smalles distance
- In 2D space it can be solved in $O(m_1 \log m_1 + m_2 \log m_2)$ time

LEMMA 2 ([1]). *For any fixed dimensionality $d \geq 4$, there is an algorithm solving the BCP problem in*

$$O\left((m_1 m_2)^{1-\frac{1}{\lceil d/2 \rceil + 1}+\delta'} + m_1 \log m_2 + m_2 \log m_1\right)$$

expected time, where $\delta' > 0$ can be an arbitrarily small constant. For $d = 3$, the expected running time can be improved to

$$O((m_1 m_2 \cdot \log m_1 \cdot \log m_2)^{2/3} + m_1 \log^2 m_2 + m_2 \log^2 m_1)).$$

- **Spherical Emptiness and Hopcroft**
  - Let S_{pt} be a set points and S_{ball} be a set of balls with the same radius all in data space $\mathbb{R}^d$ where d is a constant
  - The objective of USEC is to determine whether there is a point of S_{pt} that is covered by some ball in S_{ball}
  - Set $n = |S_{pt}| + S_{ball}$
  - In 3D space the USEC problem can be solved in $O(n^{4/3} \cdot \log^{4/3} n)$ expected time
  - Finding a 3D USEC algorithm with running time $o(n^{4/3})$ is a big problem in computational geometry
    - It is widely believed to be impossible
  - Strong hardness results are known about USEC when the dimensionality d is higher it has a connection between the problem to the **Hopcroft's problem**
    - Let S_{pt} be a set of points and S_{line} be a set of lines all in data space $\mathcal{R}^2$
    - The goal of the Hopcroft's problem is to determine whether there is a point in S_{pt} that lies on some line S_{line}
  - Hopcroft's problem can be solved in time slightly higher than $O(n^{4/3})$ time
    - Where $n = |S_{pt}| + |S_{line}|$

    - It is believed that $\Omega(n^{4/3})$ is a lower bound on how fast the problem can be solved
  - A problem X is **Hopcroft hard** if an algorithm solving X in $o(n^{4/3})$ time implies an algorithm solving the Hopcrofts problem in $o(n^{4/3})$ time
  - **Lemma 3** The USEC problem in any dimensionality $d \geq 5$ is Hopcroft hard

22.3 DBSCAN in $\geq$ 3 dimensions

THEOREM 1. *The following statements are true about the DBSCAN problem:*

- *It is Hopcroft hard in any dimensionality $d \geq 5$. Namely, the problem requires $\Omega(n^{4/3})$ time to solve, unless the Hopcroft problem can be settled in $o(n^{4/3})$ time.*
- *When $d = 3$ (and hence, $d = 4$), the problem requires $\Omega(n^{4/3})$ time to solve, unless the USEC problem can be settled in $o(n^{4/3})$ time.*

LEMMA 4. *For any dimensionality d, if we can solve the DBSCAN problem in $T(n)$ time, then we can solve the USEC problem in $T(n) + O(n)$ time.*

THEOREM 2. *For any fixed dimensionality $d \geq 4$, there is an algorithm solving the DBSCAN problem in $O(n^{2-\frac{2}{\lceil d/2 \rceil + 1}+\delta})$ expected time, where $\delta > 0$ can be an arbitrarily small constant. For $d = 3$, the running time can be improved to $O((n \log n)^{4/3})$ expected.*

22.4 ρ Approximate DBSCAN

DEFINITION 4. *A point $q \in P$ is* **ρ-approximate density-reachable** *from $p \in P$ if there is a sequence of points $p_1, p_2, ..., p_t \in P$ (for some integer $t \geq 2$) such that:*

- *$p_1 = p$ and $p_t = q$*
- *$p_1, p_2, ..., p_{t-1}$ are core points*
- *$p_{i+1} \in B(p_i, \epsilon(1+\rho))$ for each $i \in [1, t-1]$.*

- **Definition 5.** A ρ approximate cluster C is a non-empty subset of P such that
  - **Maximality:** If a core point $p \in C$ then all points density-reachable from p also belong to C
  - ρ **Approximate Connectivity:** For any points $p_1, p_2 \in C$ there exists a point $p \in C$ such that both p_1 and p_2 are ρ approximate density reachable from p

PROBLEM 2. *The* **ρ-approximate DBSCAN problem** *is to find a set $\mathscr{C}$ of ρ-approximate clusters of P such that every core point of P appears in exactly one ρ-approximate cluster.*

- **The Sandwich Theorem:**
  - $\mathcal{C}_1$ is defined as the set of clusters of DBSCAN with parameters $(\epsilon, MinPts)$
  - $\mathcal{C}_2$ is defined as the set of clusters of DBSCAN with parameters $(\epsilon(1+p), MinPts)$
  - $\mathcal{C}$ is defied as an arpitary set of clusters of that is a legal result of $(\epsilon, MinPts, \rho)$ approx DBSCAN

THEOREM 3 (SANDWICH QUALITY GUARANTEE). *The following statements are true:*

1. *For any cluster $C_1 \in \mathscr{C}_1$, there is a cluster $C \in \mathscr{C}$ such that $C_1 \subseteq C$.*
2. *For any cluster $C \in \mathscr{C}$, there is a cluster $C_2 \in \mathscr{C}_2$ such that $C \subseteq C_2$.*

LEMMA 5. *For any fixed ϵ and ρ, there is a structure of $O(n)$ space that can be built in $O(n)$ expected time, and answers any approximate range count query in $O(1)$ expected time.*

23 Exam

- Tag et billede med af hvordan tavlen skal se ud