Introduction to Machine Learning

GianAndrea Müller

July 19, 2019

Contents

Maths

Probability

Most Important Distributions

Verteilung	$p(x)$	W_X	$\mathbb{E}[X]$	$\text{Var}(X)$
Bernoulli(p)	$p^x(1-p)^{1-x}$	$\{0,1\}$	p	$p(1-p)$
Bin(n,p)	$\binom{n}{k}p^x(1-p)^{n-x}$	$\{0,\ldots,n\}$	np	$np(1-p)$
Mul(n,k,p_i)	$\frac{n!}{\prod_{i=0}^k x_i!}\prod_{i=0}^k p_i^{x_i}$		np_i	$np_i(1-p_i)$
Geom(p)	$p(1-p)^{x-1}$	$\{1,2,\ldots\}$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
Pois(λ)	$e^{-\lambda}\frac{\lambda^x}{x!}$	$\{0,1,\ldots\}$	λ	λ
Uni(a,b)	$\frac{1}{b-a}$	$[a,b]$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
Exp(λ)	$\lambda e^{-\lambda x}$	$\mathbb{R}_+$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
Gamma(α,λ)	$\frac{\lambda^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\lambda x}$	$\mathbb{R}_+$	$\frac{\alpha}{\lambda}$	$\frac{\alpha}{\lambda^2}$
$\mathcal{N}(x;\mu,\sigma^2)$	$\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$	$\mathbb{R}$	μ	σ^2

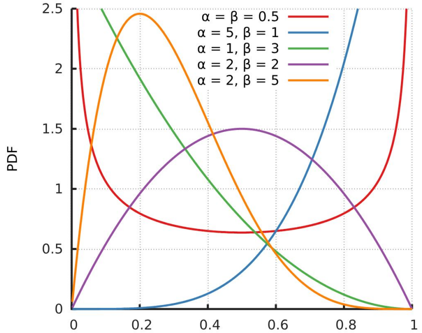
$$\mathcal{N}_k(\underline{x};\underline{\mu},\Sigma) = \frac{\exp(-\frac{1}{2}(\underline{x}-\underline{\mu})^T\Sigma^{-1}(\underline{x}-\underline{\mu}))}{(2\pi)^k\sqrt{\det(\Sigma)}}$$

$$\ln(\mathcal{N}_k) = -\frac{k}{2}\ln(2\pi) - \frac{1}{@}\ln(|\Sigma|) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

Beta Priors

$$\text{Beta}(\theta,\alpha_+,\alpha_-) = \frac{1}{B(\alpha_+,\alpha_-)}\theta^{\alpha_+-1}(1-\theta)^{\alpha_--1}$$

where $B(\alpha_+,\alpha_-)$ is a simple normalization constant.



Calculation Rules

1. $\mathbb{E}[a+bX+cY] = a+b\cdot\mathbb{E}[X]+c\cdot\mathbb{E}[Y]\,,\ a,b,c\in\mathbb{R}$
   no matter if X,Y independent or not
2. $\text{Var}(X) = \mathbb{E}\left[(X-\mu_X)^2\right] = \mathbb{E}\left[X^2\right]-\mathbb{E}[X]^2$
3. $\text{Var}(a+bX) = b^2\text{Var}(X)\,,\ a,b\in\mathbb{R}$
4. X and Y **independent**:
   - $\text{Var}(X+Y) = \text{Var}(X)+\text{Var}(Y)$
   - $\text{Var}(X-Y) = \text{Var}(X)+\text{Var}(Y)$
   - $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$
5. $P(A) = \sum_{i=1}^k P(A|B_i)P(B_i)$
6. $P(B|A) = \frac{P(A|B)P(B)}{P(A)} = \frac{P(A,B)}{P(A)} = \frac{P(A|B)P(B)}{\int P(A|\tilde{B})P(\tilde{B})d\tilde{B}}$
7. Data Normalization: $\hat{x} = \frac{x-\mu}{\sigma}$
8. MLE: $\frac{\partial}{\partial\theta}\sum_{i=0}^n \log(p(y_i|\underline{x}_i,\underline{w})) = 0$
9. MAP: $\frac{\partial}{\partial\theta}\sum_{i=0}^n \log(p(y_i|\underline{x}_i,\underline{w})+\log(p(\underline{w})) = 0$

Algebra

1. $\log(xy) = \log(x)+\log(y)$
2. $\frac{\partial}{\partial x}\ln(x) = \frac{1}{x}$
3. $\int(\ln(x))dx = x\ln(x)-x$
4. De Morgan's Rules
   - Not(A and B) = Not(A) or Not(B)
   - Not(A or B) = Not(A) and Not(B)

Linear Algebra

1. $\frac{\partial}{\partial x}(\underline{x}^TA\underline{x}) = \underline{x}^T(A+A^T)\ \in\mathbb{R}^{1xn}$ where $A\in\mathbb{R}^{nx}$

Numerical Analysis

1. Gradient Descent $\mathcal{O}(nd)$:
   $w_{t+1}\leftarrow w_t-\eta_t\nabla\hat{R}(w_t),\ \hat{R} = \sum(y_i-w^Tx_i)^2$
2. Assuming convex objective for unique solution.
3. **Line search:** Optimize step size every step
   $$\boxed{\eta_t\leftarrow\underset{\eta\in\mathbb{R}}{\text{argmin}}\left(\hat{R}(\underline{w}_t)-\eta\nabla\hat{R}(\underline{w}_t)\right)}$$
4. **Bold driver heuristic:** $c_{acc}>1 \qquad c_{dec}<1$
   $$\boxed{n_{t+1}\begin{cases} n_t\cdot c_{acc} & \text{if } \hat{R}(\underline{W}_{t+1})<\hat{R}(\underline{w}_t)\\ n_t\cdot c_{dec} & \text{if } \hat{R}(\underline{w}_{t+1})>\hat{R}(\underline{w}_t)\end{cases}}$$

0 Definitions

Definition 1. ***Supervised Learning*** *aims to learn the functional relation between two sets of data, based on labelled data.*

Definition 2. ***Unsupervised Learning*** *aims to learn patterns in data sets without existing labels.*

Definition 3. ***Semi-supervised learning*** *is based on both labelled and unlabelled data.*

Definition 4. ***Transfer learning*** *learns on one domain and tests on another.*

Definition 5. ***Active learning*** *acquires most informative data for learning.*

Definition 6. ***Online learning*** *learns from examples as they arrive over time.*

Definition 7. ***Reinforcement learning*** *learns by interacting with an unknown environment.*

Definition 8. $f:\mathbb{R}^d\to\mathbb{R}$ ***convex*** *iff*
$\forall\, x_1,x_2\ \in\ \mathbb{R}^d,\ \forall\lambda\in[0,1]:\quad f(\lambda x_1+(1-\lambda)x_2)\leq\lambda f(x_1)+(1-t)f(x_2)$

1 Regression

1.1 Linear Regression

$$y = ax+b$$

$$\boxed{\underline{w}^* = \underset{\underline{w}}{\text{argmin}}\left(\tilde{R}(\underline{w})\right) = \underset{\underline{w}}{\text{argmin}}\left(\sum_{i=1}^n(y_i-\underline{w}^T\underline{x}_i)^2\right)}$$

Linear Least-Squares Regression

$$\boxed{\underline{w}^* = (\underline{X}^T\underline{X})^{-1}\underline{X}^T\underline{y}}$$ Closed Form Solution $\mathcal{O}(d^3)$

1.2 Choices for Loss Functions

- $l_2(w,x,y) = (y-\underline{w}^Tx)^2$
- $l_1(w,x,y) = |y-\underline{w}^Tx|$
  - $+$ Magnitude of the derivative stays the same.
  - $+$ More emphasis on small deviations.
  - $+$ Less emphasis on large deviations (outliers), thus more robust.
- $l_p(w,x,y) = |y-\underline{w}^Tx|^p$ (convex for $p\geq1$)
- $f(x) = \sum_{i=1}^d w_i\phi_i(\underline{x})$

1.2.1 Kernelized Linear Regression

$$\underline{\hat{w}} = \underset{\underline{w}}{\text{argmin}}\left(\frac{1}{n}\sum_{i=1}^n\left(\underline{w}^t\underline{x}_i-y_i\right)^2+\lambda||\underline{w}||_2^2\right)$$

$$\boxed{\underline{\hat{\alpha}} = \underset{\underline{\alpha}}{\text{argmin}}\left(\frac{1}{n}||\underline{\alpha}^T\underline{K}-\underline{y}||_2^2+\lambda\underline{\alpha}^T\underline{K}\alpha\right)}$$

where $\underline{K} = \begin{pmatrix} k(\underline{x}_1,\underline{x}_1) & \cdots & k(\underline{x}_1,\underline{x}_n)\\ \vdots & & \vdots\\ k(\underline{x}_n,\underline{x}_1) & \cdots & k(\underline{x}_n,\underline{x}_n)\end{pmatrix}$

$$\boxed{\underline{\hat{\alpha}} = (\underline{K}+\lambda\underline{K})^{-1}\underline{y}}$$ Closed form solution

$$\boxed{f(\underline{x}) = \sum_{i=1}^n\alpha_ik(\underline{x}_i,\underline{x}}$$ Regressor

2 Model Validation and Selection

Goal of supervised learning: Find the model that features the lowest prediction error. The prediction error decreases when increasing the model order up to the point where the model is over-fitting the data, in which case the training error further reduces, while the prediction error rises again.

2.1 Probability

Fundamental assumption: I.i.d. data

$$(\underline{x}_i, y_i) \sim P(\underline{X}, Y)$$

The goal is to minimize the expected error under P (true risk)

$$R(\underline{w}) = \int P(\underline{x}, y)(x - \underline{w}^T\underline{x})^2 d\underline{x}dy = \mathbb{E}_{\underline{x},y}\left[(y - \underline{w}^T\underline{x})^2\right]$$

The **true risk** can be estimated by the **empirical risk**

$$\hat{R}_D(\underline{w}) = \frac{1}{|D|}\sum_{(\underline{x},y)\in D}(y - \underline{w}^T\underline{x})^2$$

For large number the estimate approaches the true risk.

$\hat{\underline{w}}$	estimated optimal w
$\underline{w}^*$	true optimal w
$R(w) - \hat{R}(w)$	Generalization error

Under the law of large number the generalization error diminishes and $\hat{\underline{w}}$ approaches $\underline{w}^*$ under the assumption of uniform convergence

$$\sup_{\underline{w}} |R(\underline{w}) - \hat{R}_D(\underline{w})| \to 0 \text{ as } |D| \to \infty$$

In general

$$\mathbb{E}\left[\hat{R}_{train}(\hat{\underline{w}})\right] \ll \mathbb{E}\left[R(\hat{\underline{w}})\right]$$

Thus we obtain an overly optimistic estimate.

2.2 Evaluation for model selection

- Split the same data set into training and validation set.

$$D = D^{(i)}_{train} \uplus D^{(i)}_{val}$$

- Train model: $\hat{\underline{w}}_i = \operatorname{argmin}_{\underline{w}}\left(\hat{R}^{(i)}_{train}(\underline{w})\right)$
- Estimate error: $\hat{R}^{(i)}_m = \hat{R}^{(i)}_{val}(\hat{\underline{w}}_i)$
- Select model: $\hat{m} = \operatorname{argmin}_m\left(\frac{1}{k}\sum_{i=1}^{k}\hat{R}^{(i)}_m\right)$

2.2.1 Monte Carlo Cross-Validation

- Pick training set of given size uniformly at random
- Validate on remaining points
- Estimate prediction error by averaging the validation error over multiple random trials

2.2.2 K-Fold Cross-Validation

- Partition the data into k folds
- Train on $k-1$ folds evaluating on remaining fold.
- Estimate prediction error by averaging the validation error obtained while varying the validation fold.

Choice of k

- Too small
  - Risk of overfitting to test set
  - Using too little data for training
  - Risk of underfitting to training set
- Too large
  - In general better performance
    $k = n$ is perfectly fine (leave-one-out cross-validation)
  - Higher computational complexity.
- In practice: $k \in \{5, 6, 7, 8, 9, 10\}$

! **This only works if the data is i.i.d**

! **Be careful of temporal trends or other dependencies**

Nonlinear Transformations In certain cases transforming the data makes fitting easier.

$$x \to \log(x + c)$$
$$x \to x^\alpha$$
$$x \to \arcsin\sqrt{x}$$

Regularization Encourage small weights via penalty function to avoid overfitting.

$$\boxed{\min_{\underline{w}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \underline{w}^T\underline{x}_i)^2 + \lambda||\underline{w}||_2^2} \text{ Ridge Regression}$$

This can be optimized using gradient descent (convex) or in a closed form:

$$\underline{w}^* = (X^TX + \lambda\underline{I})^{-1}X^T\underline{y})$$

For this regularized version of the problem the solution depends on the magnitudes of $\underline{x}_i$. For that reason data is normalized first:

$$\tilde{x}_{i,j} = (x_{i,j} - \hat{\mu}_j)/\hat{\sigma}_j$$

where $x_{i,j}$ is the value of the j-th feature of the i-th data point. Also:

$$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^{n}x_{i,j} \qquad \hat{\sigma}_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{i,j} - \hat{\mu}_j)^2$$

Pick λ logarithmically spaced and apply cross-validation to find the optimal one.

Gradient descent for ridge regression

$$\underline{w}_{t+1} \leftarrow \underline{w}_t - \eta_t\nabla_w\hat{R}(\underline{w}_t) - \eta_t\lambda 2\underline{w}_t$$

3 Linear Classification

3.1 Binary Classification

- **Input:** Labelled data set with positive and negative examples.

$$D = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\} \quad \text{Data}$$

- **Output:** Decision rule.

$$h : \mathbb{R}^d \to \{+1, -1\},\ h(x) = \operatorname{sign}(\underline{w}^T\underline{x}) \quad \text{Classifier}$$

  where $\underline{w}$ is a vector which is perpendicular to the linear classifier.

This can be formulated as an optimization problem

$$\underline{w}^* = \operatorname{argmin}_{\underline{w}}\left(\sum_{i=1}^{n}[y_i \neq \operatorname{sign}(\underline{w}^T\underline{x}_i)]\right) = \operatorname{argmin}_{\underline{w}}\left(\sum_{i=1}^{n}l_{0/1}(\underline{w}; y_i, \underline{x}_i)\right)$$

3.1.1 Surrogate Loss Function

$l_{0,1} = \begin{cases}0 & \text{if } x > 0\\ 1 & \text{if } x \leq 0\end{cases}$ is intractable. For that reason we replace it with a perceptron

$$l_p(\underline{w}, x, y) = \max(0, -y\underline{w}^T\underline{x})$$

This allows solving

$$\underline{w}^* = \operatorname{argmin}_{\underline{w}}\left(\sum_{i=1}^{n}l_P(\underline{w}; y_i, \underline{x}_i)\right)$$

which now is a convex problem which can be solved using gradient descent.

3.1.2 Gradient Descent

$$\hat{R}_p(\underline{w}) = \sum_{i=1}^{n}\max(0, -y_i\underline{w}^T\underline{w}_i)$$

$$\nabla_{\underline{w}}\hat{R}_p(\underline{w}) = \sum_{i=1}^{n}\underbrace{\nabla_{\underline{w}}\max(0, -y_i\underline{w}^T\underline{x}_i)}_{A}$$

$$A = \begin{cases}0 & \text{if } y_i\underline{w}^T\underline{x}_i \geq 0\\ -y_i\underline{x}_i & \text{if } y_i\underline{w}^T\underline{x}_i < 0\end{cases}$$

$$\Rightarrow \nabla_{\underline{w}}\hat{R}_p(\underline{w}) = -\sum_{i:\ y_i\neq\operatorname{sign}(\underline{w}^T\underline{x}_i)} y_i\underline{x}_i$$

$$\underline{w}_{t+1} \leftarrow \underline{w}_t + \eta_t\sum_{incorrect}y_i\underline{x}_i$$

3.1.3 Stochastic Gradient Descent

- Gradient computation requires summing over all data, which is inefficient for large data sets.
- Instead the gradient for a **single randomly chosen point** is calculated.

1. Start at an arbitrary $\underline{w}_0 \in \mathbb{R}^d$

2. For $t = 1, 2, \ldots$ do
   a) Pick data point $(\underline{x}', y') \in D$ from training set uniformly at random (with replacement).
   b) $\underline{w}_{t+1} = \underline{w}_t - \eta_t \nabla l(\underline{w}_t; \underline{x}', y')$

This algorithm is guaranteed to converge under mild conditions if $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$.

Definition 9. *The* ***Perceptron Algorithm*** *is a stochastic gradient descent on the perceptron loss function.*

Possible improvements on the perceptron algorithm are:

- **Mini-batches**: Instead of a single point a small batch of points is used for the evaluation of the gradient.
- **Adaptive learning rates**

$$\boxed{\text{Hint: For} \leq 2D \text{ plot data points!}}$$

Kernelized Perceptron

1. Initialize $\alpha_i = 0$
2. for $t = 1, 2, \ldots$
   a) Pick data point $(\underline{x}_i, y_i)$ uniformly at random
   b) Predict

$$\hat{y} = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j k(\underline{x}_j, \underline{x}_i)\right)$$

   c) If $\hat{y} \neq y_j$ set $\alpha_i \leftarrow \alpha_i + \eta_t$

3.1.4 Support Vector Machines

The support vector machine assures that the classifier has a maximal margin between classifier and closest data points. Differently formulated one places two hyperplanes encompassing the classifier (parallel to it) and then maximizes their distance, placing the optimal classifier midway between.
The solution to the problem above is replacing the perceptron loss function with the hinge loss function.

$$\boxed{l_H(\underline{w}; \underline{x}, y) = \max\{0, 1 - y\underline{w}^T \underline{x}\}} \text{ Hinge loss}$$

$$\boxed{\underline{w}^* = \underset{\underline{w}}{\text{argmin}} \left(\sum_{i=1}^{n} \max\{0, 1 - y_i \underline{w}^T \underline{x}_i\} + \lambda ||\underline{w}||_2^2\right)} \text{ SVM}$$

The regularization is added in order to prevent an increase of the weights, since increasing the weights would minimize the influence of the hinge loss in comparison to the perceptron loss.

SGD for SVM

$$\underline{w}^* = \underset{\underline{w}}{\text{argmin}} \left(\sum_{i=1}^{n} \underbrace{\max\{0, 1 - y_i \underline{w}^T \underline{x}_i\}}_{l_H(\underline{w}; \underline{x}_i, y_i)} + \lambda ||\underline{w}||_2^2\right)$$

$$\nabla l_H(\underline{w}; \underline{x}_i, y_i) = \begin{cases} 0 & \text{if } y_i \underline{w}^T \underline{x}_i \geq 1 \\ -y_i \underline{x}_i & \text{otherwise} \end{cases}$$

$$\nabla ||\underline{w}||_2^2 = 2\lambda \underline{w}$$

$$\boxed{\eta_t = \frac{1}{\lambda t}} \text{ Best learning rate for SVM}$$

$$\boxed{\underline{w}_{t+1} \leftarrow \underline{w}_t (1 - w\eta_t \lambda) + y_i \underline{x}_i \eta_t [y_i \underline{w}^T \underline{x}_i \geq 1]}$$

¡

- The selection of the right λ is made using cross validation.
- The validation is done using the target performance metric (number of mistakes) instead of hinge loss.

Kernelized SVM

$$\boxed{\min_{\alpha} \sum_{I=1}^{n} \max\{0, 1 - y_i \alpha^T \underline{k}_i\} + \lambda \alpha^T D_{\underline{y}} \underline{K} D_{\underline{y}} \alpha} \text{ Kernelized SVM}$$

$$\underline{k}_i = [y_1 k(\underline{x}_i, \underline{x}_1), \ldots, y_n k(\underline{x}_i, \underline{x}_n)]$$

$$\boxed{\hat{y} = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j k(\underline{x}_j, \underline{x}_i)\right)} \text{ Prediction}$$

4 Feature Selection

4.1 Greedy Feature Selection

$$V = \{1, \ldots, d\} \quad \text{Set of all features}$$

The cross-validation error of using features in $S \subset V$ only is

$$\hat{L}(S)$$

4.1.1 Greedy Forward Selection

1. Start with $S = \{\}$ and $E_0 = \infty$
2. For $i = 1 : d$
   a) Find best element to add: $s_i = \arg\min_{j \in V n S} \left(\hat{L}(S \cup \{j\})\right)$
   b) Compute error $E_i = \hat{L}(S \cup \{s_i\})$
   c) If $E_i > E_{i-1}$ break, else set $S \leftarrow S \cup \{s_i\}$

Problems:

- When the data is spread around a line and a linear classifier is used, the classifiers based on a subset of the features are both just horizontal or vertical lines which will score the same, thus the algorithms stops without a result.

4.1.2 Greedy Backward Selection

1. Start with $S = V$ and $E_{d+1} = \infty$
2. For $i = d : -1 : 1$
   a) Find best element to remove: $s_i = \underset{j \in S}{\text{argmin}} \left(\hat{L}(Sn\{j\})\right)$
   b) Compute error: $E_i = \hat{L}(Sn\{s_i\})$
   c) If $E_i > E_{i+1}$ break, else set $S \leftarrow Sn\{s_i\}$

4.1.3 Comparison

- Forward
  + Usually faster
  - Struggles with certain data sets
- Backward
  + Can handle dependent features
- Both
  + Apply to any prediction method
  - Computationally expensive
  - Potentially suboptimal
  - Slower than lasso
- L1 regularization (Lasso)
  + Faster (training and features selection happen jointly)
  - Only works for linear models

4.2 Joint Features Selection and Training

$$\underset{\underline{w}}{\text{argmin}} \left(\sum_{i=1}^{n} (y_i - \underline{w}^T \underline{x}_i)^2\right) \text{ s.t. } ||\underline{w}||_0 \leq k$$

where $||\underline{w}||_0$ is the number of non-zeros in $\underline{w}$.
Alternatively we can penalize the number of nonzero entries

$$\hat{w} = \underset{\underline{w}}{\text{argmin}} \left(\sum_{i=1}^{n} (y_i - \underline{w}^T \underline{x}_i)^2 + \lambda ||\underline{w}||_0\right)$$

To make the problem tractable we can replace l_0 with the l_1, this is called the **sparsity trick** and the resulting sparse regression is called **the lasso**:

$$\boxed{\min_{\underline{w}} \lambda ||\underline{w}||_1 + \sum_{i=1}^{n} (y_i - \underline{w}^T \underline{x}_i)^2}$$

- This encourages coefficients to be exactly zero which allows automatic feature selection.

5 Kernels

5.1 Polynomials in Higher Dimensions

- We wish to use nonlinear features in order to fit more complicated data.
- Avoid feature explosion: $d = 10000$, $k = 2$ $\rightarrow$ need $\sim 100M$ dimensions.

The solutions of linear classifiers and regressions $\underline{\hat{w}}$ can be written as a linear combination of the data points and feature vectors:

$$\underline{\hat{w}} = \sum_{i=1}^{n} \alpha_i y_i \underline{x}_i$$

Performing gradient descent for such a problem shows that the update rule $\underline{w}_{t+1} = f(\underline{w}_t, \eta_t, y, \underline{x})$ constructs $\underline{\hat{w}}$ in the above described way.
The above observation allows to reformulate the perceptron such that its solution can be described in terms of inner products of pairs of data points, which allows an implicit calculation of higher dimensional spaces.

5.1.1 Reformulating the Perceptron

$$\underline{\hat{w}} = \underset{\underline{w}}{\text{argmin}} \left(\underbrace{\frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i \underline{w}^T \underline{x}_i)}_{*} \right)$$

$$\underline{w} = \sum_{i=1}^{n} \alpha_i y_i \underline{x}_i \quad \text{Ansatz}$$

$$\begin{aligned} * &= \frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i (\sum_{j=1}^{n} y_j \alpha_j \underline{x}_j)^T \underline{x}_i) \\ &= \frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i \sum_{j=1}^{n} y_j \alpha_j (\underline{x}_j^T \underline{x}_i)) \end{aligned}$$

Thus the problem can be formulated as

$$\boxed{\hat{\alpha} = \arg\min_{\alpha \in \mathbb{R}^n} \left(\frac{1}{n} \sum_{i=1}^{n} \max(0, -y_i \sum_{j=1}^{n} y_j \alpha_j (\underline{x}_j^T \underline{x}_i)) \right)}$$

Key observation: Objective only depends on inner products of pairs of points of data. Therefore this approach is efficient, we can work in high-dimensional spaces, as long as the inner product can be calculated easily.

Definition 10. *$< f, g >$ is an **inner product** if:*

- $< \alpha_1 f_1 + \alpha_2 f_2, g >_H = \alpha_1 < f_1, g >_H + \alpha_2 < f_2, g >_H$
- $< f, g >_H = < g, f >_H$
- $< f, f >_H \geq 0 \ \forall \ f \in H$
- $< f, f >_H = 0 \Leftrightarrow f = 0$

Definition 11. ***Kernels** can be understood as* efficient *inner products.*

- Often $k(\underline{x}, \underline{x}')$ can be computed much more efficiently than $\phi(\underline{x})^T \phi(\underline{x}')$. Formulated differently we can formulate a function that operates on the data space to calculate the result of the inner product calculated in the higher dimensional feature space without actually going there.
- k must be symmetric
  Thus if $k = \underline{x}^T M \underline{x}'$, M must be symmetric
- Taking any finite subset of the data $S = \{\underline{x}_1, \ldots, \underline{x}_n\} \subseteq X$ and calculate the kernel/gram matrix

$$K = \begin{pmatrix} k(\underline{x}_1, \underline{x}_1) & \cdots & k(\underline{x}_1, \underline{x}_n) \\ \vdots & & \vdots \\ k(\underline{x}_n, \underline{x}_1) & \cdots & k(\underline{x}_n, \underline{x}_n) \end{pmatrix} = \begin{pmatrix} \phi(\underline{x}_1)^T \phi(\underline{x}_1) & \cdots & \phi(\underline{x}_1)^T \phi(\underline{x}_n) \\ \vdots & & \vdots \\ \phi(\underline{x}_n)^T \phi(\underline{x}_1) & \cdots & \phi(\underline{x}_n)^t \phi(\underline{x}_n) \end{pmatrix}$$

which has to be positive semidefinite, which is due to the kernel implementing an inner product.

$$\boxed{k(\underline{x}, \underline{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\underline{x}) \phi_i(\underline{x}')} \text{ Mercers Theorem}$$

5.2 Examples of Kernels on $\mathbb{R}^d$

- Linear kernel: $k(\underline{x}, \underline{x}') = \underline{x}^T \underline{x}$
- Polynomial kernel of degree m:
  $k(\underline{x}, \underline{x}') = (\underline{x}^T \underline{x}')^m$
- Polynomial kernel up to degree m:
  $k(\underline{x}, \underline{x}') = (\underline{x}^T \underline{x}' + 1)^m$
- Gaussian (RBF, squared exp. kernel):
  $k(\underline{x}, \underline{x}') = \exp(-||\underline{x} - \underline{x}'||_2^2 / h^2)$
  where h denotes the bandwidth parameter.
- Laplacian kernel: $k(\underline{x}, \underline{x}') = \exp(-||\underline{x} - \underline{x}'||_1 / h)$
- $k(\underline{x}, \underline{x}') = f(\underline{x}) f(\underline{x}')$
- $k(X, Y) = |X \cup Y|$

5.3 Kernel Composition rules

Given two kernels

$$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

The following are valid kernels as well:

$$\begin{aligned} k(\underline{x}, \underline{x}') &= k_1(\underline{x}, \underline{x}') + k_2(\underline{x}, \underline{x}') \\ k(\underline{x}, \underline{x}') &= k_1(\underline{x}, \underline{x}') k_2(\underline{x}, \underline{x}') \\ k(\underline{x}, \underline{x}') &= c k_1(\underline{x}, \underline{x}') \quad \text{ for } c > 0 \\ k(\underline{x}, \underline{x}') &= f(k_1(\underline{x}, \underline{x}')) \\ k(\underline{x}, \underline{x}') &= k(\phi(\underline{x}), \phi(\underline{x}')) \end{aligned}$$

where f is a polynomial with positive coefficients or the exponential function.

- Every symmetric positive definite function is a kernel.
- Given $\mathcal{X}$, $\tilde{\mathcal{X}}$, $A : \mathcal{X} \to \tilde{\mathcal{X}}$ and $k(A(x), A(x'))$ kernel on $\mathcal{X}$, if k kernel on $\tilde{\mathcal{X}}$.

5.4 Kernelization

$$\underline{w} = \sum_{j=1}^{n} \alpha_j y_j \underline{x}_j$$

1. Kernelize the objective
2. Kernelize the regularizer

5.5 k Nearest Neighbours

For data point $\underline{x}$ predict majority of labels of k nearest neighbours. Thus classifying a data point based on what the classification of its closest neighbours are. The closeness is measured with some metric like euclidean distance.

$$\boxed{y = \text{sign} \left(\sum_{i=1}^{n} y_i [\underline{x}_i \text{ amog } k \text{ nearest neighbours of } \underline{x}] \right)}$$

- k-NN
  - \+ No training necessary
  - \- Depends on all data and is thus inefficient
- Kernelized Perceptron
  - \+ Optimized weights can lead to improved performance, can capture global trends with suitable kernels, depends on wrongly classified examples only.
  - \- Training requires optimization.

5.6 Semi-Parametric Regression

Often parametric models are too „rigid“and non-parametric models fail to extrapolate. Thus we use an additive combination of linear and non-linear kernel functions.

$$k(\underline{x}, \underline{x}') = c_1 \exp(-||\underline{x} - \underline{x}'||_2^2 / h^2) + c_2 \underline{x}^T \underline{x}'$$

5.7 Imbalanced Data

Issues:

- Accuracy is not a good metric, since the classifier might prefer some mistakes over others (trading false positives and false negatives)
- Minority class instances contribute little to the empirical risk and may be ignored during optimization.

Solutions:

- **Subsampling:** Remove training examples from the majority class.
  - \+ Smaller data set, thus faster
  - \- Available data is wasted, may lose information about majority class.
- **Upsampling:** Reat data points from minority class.
  - \+ Makes use of all data
  - \- Slower, adding perturbation requires arbitrary choices.

5.8 Cost Sensitive Classification

Modify the loss function to be cost sensitive:

- Perceptron: $l_{CS-P}(\underline{w}; \underline{x}, y) = c_y \max(0, -y\underline{w}^T\underline{x})$
- SVM. $l_{CS-H}(\underline{w}; \underline{x}, y) = c_y \max(0, 1 - y\underline{w}^T\underline{x})$

with parameters $c_+, c_- > 0$ controlling tradeoff.
The tradeoff can be obtained in two ways:

1. Cost sensitive classifiers
2. Use single classifier and vary classification threshold

$$y = \text{sign}\left(\underline{w}^T\underline{x} - \tau\right)$$

$$\boxed{R(\underline{w}; c_+, c_-) = \frac{1}{n}\sum_{y=+1} c_+ l(\cdot) + \frac{1}{n}\sum_{y=-1} c_- l(\cdot) = R(\underline{w}; \frac{c_+}{c_-}, 1)}$$

5.9 Metrics for Imbalanced Data

$\boxed{\frac{TP+TN}{TP+TN+FP+FN}}$ Accuracy

$\boxed{\frac{FP}{TN+FP}}$ False Positive Rate

$\boxed{\frac{TP}{TP+FP}}$ Precision

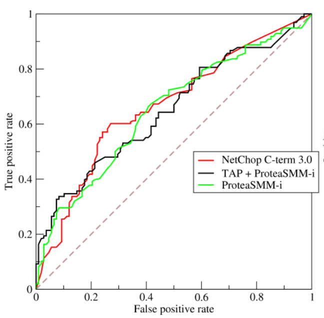
Harmonic mean of Precision and Recall:

$\boxed{\frac{TP}{TP+FN}}$ Recall
True Positive Rate

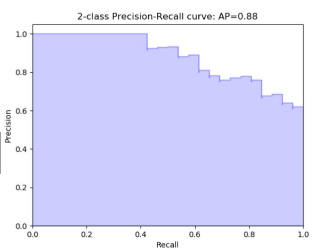
$\boxed{\frac{2TP}{2TP+FP+FN}}$ F1 Score

5.9.1 Area under the Curve

Receiver Operator Characteristics (ROC)



Precision Recall Curve



To compare the ability of classifiers to provide imbalanced classification we can compute the area under the ROC or Precision Recall curve. An AOC of 0.5 means random prediction.

5.10 Multiple Classes

Given

$$\mathcal{D} = \{(\underline{x}_1, y_1), \dots (\underline{x}_n, y_n)\} \qquad y_i \in \mathcal{Y} = \{1, \dots, c\} \qquad \underline{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$$

Want

$$f: \mathcal{X} \to \mathcal{Y}$$

5.10.1 One-vs-all classifiers

- Solve c binary classifiers, one for each class.
- Classify using the classifier with the largest confidence. (Let all classifiers compete for data points and assign to that classifiers with the largest confidence $f^{(i)}(\underline{x}) = \underline{w}^{(i)T}\underline{x}$)

Challenges

- Only works well if classifiers produce confidence scores on the same scale.
- Individual classifiers see imbalanced data, even if the whole data set is balanced.
- One class might not be linearly separable from all other classes.

5.10.2 One-vs-one classifiers

- Train $c(c-1)/2$ binary classifiers, one for each pair of classes.
- Apply voting scheme, class with highest number of positive predictions wins.

5.10.3 Comparison

- One-vs-all
  + Only c classifiers needed (faster)
  - Requires confidence in prediction / leads to class imbalance
- One-vs-one
  + No confidence needed
  - Slower (need to train $c(c-1)/2$ models

5.10.4 Alternative Methods

- Binary encoding
- Error correcting output codes

5.11 Multi-Class SVMs

Key idea: Maintain c weight vectors, one for each class

$$\underline{w}^{(1)}, \dots, \underline{w}^{(c)}$$

Given each data point we want to achieve that

$$\boxed{\underbrace{\underline{w}^{(y)T}\underline{x}}_{\text{Score for class } y} > \underbrace{\max_{i \neq y} \underline{w}^{(i)T)}\underline{x}}_{\text{Score for any other class}} + \underbrace{1}_{\text{margin}} \qquad (*)}$$

$$\boxed{l_{MC-H}(\underline{w}^{(i)}; \underline{x}, y) = \max\left(0, 1 + \max_{j \in \{1,..,y-1,y+1,..,c\}} \underline{w}^{(j)T}\underline{w} - \underline{w}^{(y)T}\underline{x}\right)}$$

Multi-class Hinge Loss

$$\nabla_{\underline{w}^{(i)}} l_{MC-H}(\underline{w}^{(i)}; \underline{x}, y) = \begin{cases} 0 & \text{if } (*) \text{ or } j \notin \{y, \hat{y}\} \\ -\underline{x} & \text{if } \neg(*) \text{ and } j = y \\ \underline{x} & \text{otherwise} \end{cases}$$

6 Neural Networks

6.1 Features

+ Invariance to Rotation
+ Invariance to Scaling
- Features can be hand-designed to a specific task by a domain expert.
- Kernels present a rich set of feature maps and can fit any function with infinite data but the choice of the right kernel can be difficult and the computational complexity grows with the size of the data.

6.1.1 Learning Features

$$\underline{w}^* = \underset{\underline{w}}{\text{argmin}} \left(\sum_{i=1}^n l\left(y_i; \sum_{j=1}^m w_j \phi_j(\underline{x}_i)\right)\right)$$

Idea: Parametrize the feature maps and optimize over the parameters

$$\boxed{\underline{w}^* = \underset{\underline{w}, \theta}{\text{argmin}} \left(\sum_{i=1}^n l\left(y_i; \sum_{j=1}^m w_j \phi(\underline{x}_i, \theta_j)\right)\right)}$$

6.1.2 Activation Functions

Simplest approach:

$$\phi(\underline{x}, \theta) = \phi(\underbrace{\theta^T\underline{w}}_{z})$$

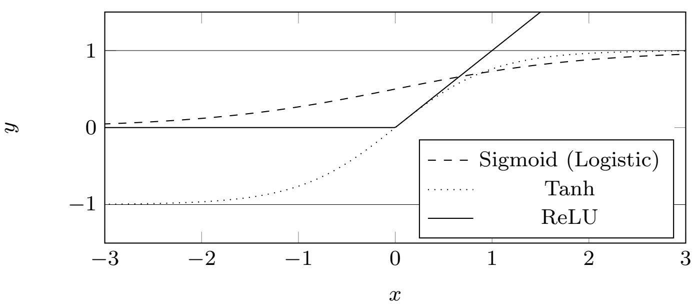
Thus $\phi: \mathbb{R} \to \mathbb{R}$ is called the **activation function**.

$$\phi(z) = \frac{1}{1+\exp(-z)} \qquad \phi'(z) = (1-\phi(z))\phi(z) \qquad \text{Sigmoid AF}$$

$$\phi(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)} \qquad \text{Tanh AF}$$

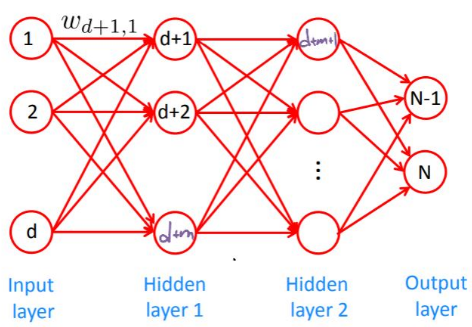
$$\phi(z) = \max(z, 0) \qquad \phi'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Rectified linear units (ReLU)



6.2 Artificial Neural Networks (ANNs)

$$f(\underline{x}; \underline{w}, \theta) := \sum_{j=1}^{m} w_j \underbrace{\phi(\theta_j^T \underline{x})}_{v_j}$$



6.2.1 Forward Propagation

1. For each unit j on input layer, set its value $v_j = x_j$
2. For each layer $l = 1 : L-1$
   - For each unit j on layer l set its value

$$v_j = \phi\left(\underbrace{\sum_{i \in \text{Layer}_{l-1}} w_{j,i} v_i}_{z_j}\right)$$

3. For each unit j on output layer, set its value

$$f_j = \sum_{i \in \text{Layer}_{L-1}} w_{j,i} v_i$$

4. Predict

$$\begin{aligned} y_j &= f_j \text{ for regression} \\ y_j &= \text{sign}(f_j) \text{ for classification} \\ y_j &= \text{argmax}(f_j) \text{ for multiclass classification} \end{aligned}$$

In short:

1. For input layer: $\underline{v}^{(0)} = \underline{x}$
2. For each hidden layer $l = 1 : L-1$

$$\begin{aligned} \underline{z}^{(l)} &= \underline{W}^{(l)} \underline{v}^{(l-1)} \\ \underline{v}^{(l)} &= \phi\left(\underline{z}^{(l)}\right) \end{aligned}$$

3. For output layer: $f = \underline{W}^{(l)} \underline{v}^{(L-1)}$
4. Predict: $\underline{y} = \underline{f}$(regression) or $\underline{y} = \text{sign}\left(\underline{f}\right)$(class.) or $\underline{y} = \text{argmax} f_j$ (multiclass.)

6.2.2 Universal Approximation Theorem

Theorem 1. *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)$ of the above form for which

$$|G(x) - f(x)| < \epsilon \qquad \forall \quad x \in I_n$$

Thus the ANN can approximate any continuous function.

6.3 How to train

Given $D = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$ we want to optimize the weights: $\underline{W} = (\underline{W}^{(1)}, \ldots, \underline{W}^{(L)})$.

- Apply loss function

$$l(\underline{W}; \underline{y}, \underline{x}) = l(\underline{y} - f(\underline{x}, \underline{W}))$$

- Optimize the weights to minimize loss over D

$$\underline{W}^* = \underset{\underline{W}}{\text{argmin}} \left(\sum_{i=1}^{n} l(\underline{W}; \underline{y}_i, \underline{x}_i) \right)$$

  for multiple outputs, define loss as sum of per-output losses.

This problem is non-convex, thus there is no guarantee for finding the globally optimal solution.

6.3.1 Stochastic Gradient Descent

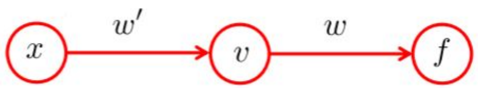
$$\underline{W}^* = \underset{\underline{W}}{\text{argmin}} \left(\sum_{i=1}^{n} l(\underline{W}; \underline{y}_i, \underline{x}_i) \right)$$

1. Initialize weights $\underline{W}$
2. for $t = 1, 2, \ldots$
   - Pick data point $(\underline{x}, \underline{y}) \in D$ uniformly at random
   - Take step in negative gradient direction

$$\underline{W} \leftarrow \underline{W} - \eta_t \nabla_{\underline{W}} l(\underline{W}; \underline{y}, \underline{x})$$

How to compute the gradient? Simple example featuring one input, one hidden and one output unit:

$$f(x, \underline{W}) = w \overbrace{\phi(\underbrace{w' x}_{z})}^{v}$$



$$\begin{aligned} D &= \{(x, y)\} \\ L(w', w) &= l_y(f) = (f - y)^2 \\ \frac{\partial L}{\partial w} &= \underbrace{\frac{\partial L}{\partial f}}_{\delta} \frac{\partial f}{\partial w} = \underbrace{l_y'(f)}_{\delta} v = \underbrace{2(f-y)}_{\delta} v \\ \frac{\partial L}{\partial w'} &= \underbrace{\frac{\partial L}{\partial f}}_{\delta} \underbrace{\frac{\partial f}{\partial v}}_{w} \frac{\partial v}{\partial w'} = \underbrace{\delta w \phi'(z)}_{\delta'} x \end{aligned}$$

Where the only two things not computed in forward propagation are the error signal from the output layer and the derivatives of the loss and the activation functions.

More complicated example

$$L = \sum_i^2 l_i \left(\underbrace{\sum_j^2 w_{i,j} \phi \overbrace{\left(\sum_k^2 \underbrace{w'_{j,k} x_k}_{z_k} \right)}^{v_j}}_{f_i} \right)$$

$$\frac{\partial L}{\partial w_{i,j}} = \underbrace{\frac{\partial L}{\partial f_i}}_{\delta_i} \frac{\partial f_i}{\partial w_{ij}} = \underbrace{l_i'(f_i)}_{\delta_i} v_j$$

$$\frac{\partial L}{\partial w'_{jk}} = \sum_i \underbrace{\frac{\partial L}{\partial f_i}}_{\delta_i} \frac{\partial f_i}{\partial v_j} \frac{\partial v_j}{\partial w'_{jk}} = \underbrace{\sum_{i=1}^{2} \delta_i w_{i,j} \phi'(z_j)}_{\delta'_j} x_k$$

Backpropagation

1. For each unit on the output layer
   - Compute error signal $\delta_j = l_j'(f_j)$
   - For each unit i on layer L, $\frac{\partial L}{\partial w_{j,i}} = \delta_j v_i$
2. For each unit j on hidden layer $l = L-1 : -1 : 1$
   - Compute error signal $\delta_j = \phi'(z_j) \sum_{i \in \text{Layer}_{l+1}} w_{i,j} \delta_i$
   - For each unit i on layer $l-1$, $\frac{\partial L}{\partial w_{j,i}} = \delta_j v_i$

This can be formulated in matrix form:

1. For the output layer
   - Compute error: $\delta^{(L)} = \underline{l}'(\underline{f}) = [l'(f_1), \ldots, l'(f_p)]$
   - Gradient $\nabla_{\underline{W}^{(L)}} l(\underline{W}; \underline{y}, \underline{x}) = \delta^{(L)} \underline{v}^{(L-1)T}$
2. For each hidden layer $l = L-1 : -1 : 1$
   - Compute error $\delta^{(l)} = \phi'\left(\underline{z}^{(l)}\right) \odot \left(\underline{W}^{(l+1)T} \delta^{(l+1)}\right)$ where $\odot$ is pointwise multiplication.
   - Gradient $\nabla_{\underline{W}^{(l)}} l(\underline{W}; \underline{y}, \underline{x}) = \delta^{(l)} \underline{v}^{(l-1)T}$

6.4 Initializing Weights

- Non-convex problem, thus initialization matters
- Random initialization usually works well
  - Glorot (tanh):
$$w_{i,j} \sim \mathcal{N}(0, 1/(n_{in}))$$
$$w_{i,j} \sim \mathcal{N}(0, 2/(n_{in} + n_{out}))$$
  - He (ReLU):
$$w_{i,j} \sim \mathcal{N}(0, 2/n_{in})$$

6.5 Learning Rate

$$\underline{W} \leftarrow \underline{W} - \eta_t \nabla_{\underline{W}} l(\underline{W}; \underline{y}; \underline{x})$$

- Start with a fixed small learning rate and decrease slowly after some iterations.
$$\eta_t = \min(0.1, 100/t)$$
- Or a learning schedule, a piecewise constant learning rate, decreasing over time.

6.5.1 Learning with Momentum

- Idea: Move not only into a direction of gradient, but also in direction of last weight update.
$$a \leftarrow ma + \eta_t \nabla_{\underline{W}} l(\underline{W}; \underline{y}; \underline{x})$$
$$\underline{W} \leftarrow \underline{W} - \underline{a}$$

a	Previous direction
m	Friction („forgetting"previous a

- This can help prevent oscillations.

6.6 Weight-space Symmetries

- Multiple distinct weights compute the same predictions.
- Therefore multiple local minima can be equivalent in terms of input-output mapping.

6.7 Avoiding overfitting

- **Early stopping:** Don't run SGD until convergence.
- **Regularization:** Add penalty term to keep weights small.
- **Dropout:** Randomly ignore hidden units during each iteration of SGD with probability 1/2. After training half the weights to compensate.

6.8 Batch normalization [Ioffe & Szegedy 2015]

- Idea: normalize inputs to each layer according to mini-batch statistics.
- Reduces internal covariate shift
- Enables larger learning rates
- Helps with regularization

6.9 Convolutional Neural Networks

The motivation for constructing CNNs is to use the invariant properties of convolution operations for ANNs such that they achieve a certain independency towards scaled or rotated features. The main idea is to implement a convolution layer by limiting interaction to nearby nodes.

Definition 12. ***Pooling*** *describes the process of reducing the number of parameters of a neural network by gathering subsets of nodes to form a single new node with a certain process (for example* max()*).*

$$\boxed{\left(\frac{n+2p-f}{s}, m\right)} \text{ Output dimension}$$

where p is the padding, n is the number of pixels in $I \in \mathbb{R}^{n \times n}$, s is the stride, f describes the filter size of $F \in \mathbb{R}^{f \times f}$ and m describes the number of filters applied.

7 Clustering

Idea: group data points into clusters such that similar points are in the same clusters and dissimilar points are in different clusters.

Definition 13. *The* ***hierachical*** *approach builds a clustering tree (bottom-up or top down) representing distances among data points.*

Definition 14. *The* ***partitional*** *approach defines an optimizes a notion of „cost"defined over partitions. Basically you build a graph between points and then cut it (for example you make a cut to receive non-trivial partitions and try to minimize the number of edges cut).*

Definition 15. *The* ***model based*** *approach maintains cluster „models"and infers cluster membership, for example assigning each point to the closest cluster-center.*

7.1 K-Means Clustering

- Represent each cluster by a single point.
- Assign points to closest center
- Assumes points are in Euclidean space $\underline{x}_i \in \mathbb{R}^d$
- Represents clusters as centers $\mu_j \in \mathbb{R}^d$

$$\boxed{\begin{gathered}\hat{R}(\mu) = \hat{R}(\mu_1, \dots, \mu_k) = \sum_{i=1}^{n} \min_{j \in \{1,\dots,k\}} ||\underline{x}_i - \mu_j||_2^2 \\ \hat{\mu} = \arg\min_{\mu} \hat{R}(\mu)\end{gathered}}$$

- K-Median clustering is particularly robust to outliers.

7.1.1 Lloyd's Heuristic

1. Initialize cluster centers $\mu^{(0)} = [\mu^{(0)}, \dots, \mu_k^{(0)}]$
2. while not converged
   - Assign each point $\underline{x}_i$ to the closest center
$$z_i \leftarrow \arg \min_{j \in \{1,\dots,k\}} ||\underline{x}_i - \mu_j^{(t-1)}||_2^2$$
   - Update center as mean of assigned data points
$$\mu_j^{(t)} \leftarrow \frac{1}{n_j} \sum_{i: z_i = j} \underline{x}_i$$

7.1.2 Properties

- Guaranteed to monotonically decrease average squared distance in each iteration.
- Converges to a local optimum, which means it is sensitive to initialization.
- Complexity per iteration $\mathcal{O}(nkd)$
- The number of iterations required can be exponential.
- Determining the number of clusters is hard.

7.2 Adaptive Seeding (K-Means++)

Random seeding can result in two problems: Clusters without centers and clusters with multiple centers. For that reason adaptive seeding is used:

1. Start with a random data point as center.
2. Add centers 2 to k randomly, proportionally to the squared distance to closest selected center. Those points are sampled with the following probability, where $D(x)$ denotes the shortest distance from a data point to the closest center we have already chosen
$$P = \frac{D(x)^2}{\sum_{x \in X} (x)^2}$$
   Thus the probability to choose a new center far away from all others is more likely, and also clusters with many points are more likely to receive a closeby center.

How to determine k (Ellbow method) The heuristic for determining k is based on the point at which the reduction in the k-means cost function begins to reduce slower. This is chosen as the optimal number of clusters. The same result can be achieved by adding a regularization term and find the minimum of that cost function.

8 Dimensionality Reduction

Given data set $D = \{\underline{x}_1, \dots, \underline{x}_n\}$ obtain „embedding"(low dimensional representation $\underline{z}_1, \dots, \underline{z}_n \in \mathbb{R}^k$.

8.1 Approach

- Assume $D = \{\underline{x}_1, \dots, \underline{x}_n\} \subseteq \mathbb{R}^d$
- Obtain mapping $\underline{f} : \mathbb{R}^d \to \mathbb{R}^k$ where $k \ll d$
- Distinguish between
  - Linear dimension reduction: $\underline{f}(\underline{x}) = \underline{Ax}$
  - Nonlinear dimension reduction (parametric or non-parametric.
- The goal in dimension reduction is using a mapping that allows the reconstruction of the original data. The mapping should compress the data.
  - A simple example of a good compression is 2D data that can be fit with a line. A possible dimension reduction is to represent data points by their projection onto the fitted line.

8.2 Linear dimensionality reduction

- $D = \{\underline{x}_1, \ldots, \underline{x}_n\} \subseteq \mathbb{R}^d$
- Want $z_i \underline{w} \approx \underline{x}_i$ e.g. minimizing $||z_i \underline{w} - \underline{w}_i||_2^2$
- To ensure uniqueness normalize $||\underline{w}||_2 = 1$
- Optimize over $\underline{w}, z_1, \ldots, z_n$ jointly:

$$(\underline{w}^*, \underline{z}^*) = \arg \min_{||\underline{w}||_2=1, z_1,\ldots,z_n \in \mathbb{R}} \sum_{i=1}^n ||\underline{x}_i - z_i \underline{w}||_2^2$$

- Then, given a certain direction $\underline{w}$ the optimal z can be found as

$$z_i^* = \underline{w}^T \underline{x}_i$$

Thus we effectively solve a regression problem, interpreting x as features and z as labels.
Inserting the above in the initial optimization results in

$$\underline{w}^* = \arg \max_{||\underline{w}||_2=1} \sum_{i=1}^n (\underline{w}^T \underline{x}_i)^2$$

which can be further simplified to

$$\underline{w}^* = \arg \max_{||\underline{w}||_2=1} \underline{w}^T \Sigma \underline{w}$$

where $\Sigma = \frac{1}{n} \sum_{i=1}^n \underline{x}_i \underline{x}_i^T$ is the **empirical covariance** assuming the data is centered: $\mu = \frac{1}{n} \sum_i \underline{x}_i = 0$.

The closed form solution to the above problem is then given by the principal eigenvector of Σ i.e. $\underline{w}^* = \underline{v}_1$ where λ_1 is the largest eigenvalue.

- This idea can be generalized to $k > 1$, thus projections to more than one dimension:

8.3 Principal Component Analysis (PCA)

- Linear Dimensionality Reduction for multiple orthogonal directions.
- **The data needs to be centered!**

$$\boxed{\mu = \frac{1}{n} \sum_i x_i = 0}$$

$$\Sigma = \frac{1}{n} \sum_{i=1}^n x_j x_i^T = \frac{1}{n} \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} \quad \text{Empirical Covariance}$$

$$(\underline{W}, \underline{z}_1, \ldots, \underline{z}_n) = \arg\min \sum_{i=1}^n ||\underline{W} \underline{z}_i - \underline{x}_i||_2^2$$

where $\underline{W} \in \mathbb{R}^{d \times k}$ is orthogonal, $\underline{z}_1, \ldots, \underline{z}_n \in \mathbb{R}^k$ and is given by $\underline{W} = (\underline{v}_1 | \ldots | \underline{v}_k)$ and $\underline{z}_i = \underline{W}^T \underline{x}_i$ where

$$\Sigma = \sum_{i=1}^n \lambda_i \underline{v}_i \underline{v}_i^T \qquad \lambda_1 \geq \ldots \geq \lambda_d \geq 0$$

This projection is chosen to minimize the reconstruction error (measured in Euclidean norm). The eigenvectors of the covariance matrix can be found using SVD.

$$X^T X = V S^T U^T U S V^T = V \underbrace{S^T S}_{D} V^T$$

where the solution, the new basis can be found as the first k columns of V.

k can be chosen by cross validation. For visualization we can choose k by inspection.

- Comparing K-Means to PCA, one can say that PCA has orthogonal basis vectors, where K-Means has arbitrary basis vectors.

8.4 Nonlinear Dimension Reduction

We can use kernels to reduce nonlinear problems to linear ones.
Thus we apply feature maps to PCA in the following way

- We'd like to solve

$$\arg \max_{||\underline{w}||_2=1} \left(\underline{w}^T \underline{X}^T \underline{X} \underline{w}\right) = \arg \max_{||\underline{w}||_2=1} \left(\sum_{j=1}^n \alpha_j \phi(\underline{x}_j)\right)$$

- Applying features maps: $\underline{w} = \sum_{j=1}^n \alpha_j \phi(\underline{x}_j)$ and thus $||\underline{w}|_2^2 = \alpha^T \underline{K} \alpha$
- This can be simplified to

$$\arg \max_{\alpha^T \underline{K} \alpha = 1} \left(\alpha^t \underline{K}^T \underline{K} \alpha\right)$$

where K is the kernel matrix $k(\underline{x}_j, \underline{x}_i) = \phi(\underline{x}_j)^T \phi(\underline{x}_i)$.

- And the final problem formulates as:

$$\boxed{\alpha^* = \arg \max_{\alpha^T \underline{K} \alpha = 1} \left(\alpha^T \underline{K}^T \underline{K} \alpha\right)}$$

For which the closed form solution can be found as

$$\alpha^* = \frac{1}{\sqrt{\lambda_1}} \underline{v}_1$$

where λ_1 is the principal/largest eigenvalue and $\underline{v}_1$ the corresponding eigenvector.

8.4.1 Kernel PCA (general k)

- For general $k \geq 1$, the Kernel Principal Components are given by $\alpha^{(1)}, \ldots, \alpha^{(k)} \in \mathbb{R}^n$

$$\text{where } \alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} \underline{v}_i$$

$$\underline{K} = \sum_{i=1}^n \lambda_i \underline{v}_i \underline{v}_i^T \qquad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$$

- A new data point is then projected as $\underline{z} \in \mathbb{R}^k$

$$z_i = \sum_{j=1}^n \alpha_j^{(i)} k(\underline{x}, \underline{x}_j)$$

8.5 Autoencoders

Key idea: try to learn the identity function $\underline{x} \approx f(\underline{x}; \theta)$

$$f(\underline{x}; \theta) = f_2(f_1(\underline{x}; \theta_1); \theta_2)$$

$$\begin{array}{ll} f_1 : \mathbb{R}^d \to \mathbb{R}^k & \text{encoder} \\ f_2 : \mathbb{R}^k \to \mathbb{R}^d & \text{decoder} \end{array}$$

- Neural Networt Autoencoders are ANNs with one output unit for each of the d input units, and $k < d$ hidden units.
- The goal is to optimize the weights, such that the output agrees with the input. For example you can minimize the square loss for training:

$$\min_{\underline{W}} \sum_{i=1}^n ||\underline{x}_i - f(\underline{x}_i; \underline{W})||_2^2$$

- If you use linear activation functions and a single hidden layer you end up with PCA.

9 Statistical Perspective

9.1 Minimizing Generalization Error

- Fundamental assumption: Our data set is generated i.i.d.

$$(\underline{x}_i, y_i) \sim P(\underline{X}, Y)$$

- Goal: Find hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the prediction error

$$R(h) = \int P(\underline{x}, y) l(y; h(\underline{x})) d\underline{x} dy = \mathbb{E}_{\underline{x},y} \left[l(y; h(\underline{x}))\right]$$

9.1.1 Example: LSQ

$$R(h) = \mathbb{E}_{\underline{x},y} \left[(y - h(\underline{x}))^2\right] \quad \text{Risk}$$

Which h minimizes the risk? The hypothesis h^* is given by the conditional mean:

$$\boxed{h^*(\underline{x}) = \mathbb{E}\left[Y | \underline{X} = \underline{x}\right]}$$

Thus if we can estimate a predictor from the training data to estimate the conditional distribution

$$\hat{P}(Y | \underline{X})$$

based on which we can predict the label y for point $\underline{x}$

$$\boxed{\hat{y} = \hat{\mathbb{E}}\left[Y | \underline{X} = \underline{x}\right] = \int \hat{P}(y | \underline{X} = \underline{x}) y dy}$$

A common approach is **parametric estimation:**

- Choose a particular parametric form $\hat{P}(Y | \underline{X}, \theta)$
- Find the Maximum (conditional) Likelihood Estimation

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \left(\hat{P}(y_1, \ldots, y_n | \underline{x}_1, \ldots, \underline{x}_n, \theta)\right)$$

9.1.2 MLE for Conditional Linear Gaussian

- The negative log likelihood is given by

$$L(\underline{w}) = -\log P(y_i|\underline{x}_i, \underline{w}) = \frac{n}{2}\log(2\pi\sigma^2) + \sum_{i=1}^{n} \frac{(y_i - \underline{w}^T\underline{x}_i)^2}{2\sigma^2}$$

- Under the conditional linear gaussian assumption (noise is gaussian and iid), maximizing the likelihood is equivalent to LSQ estimation.

9.2 Bias Variance Tradeoff

$$\hat{h}_D = \operatorname*{argmin}_{h\in\mathcal{H}} \left(\sum_{(\underline{x},y)\in D} (y - h(\underline{x}))^2 \right)$$

$$\mathbb{E}_D\left[\mathbb{E}_{\underline{X},Y}\left[(Y - \hat{h}_D(\underline{X})^2\right]\right] = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

$$\boxed{\begin{aligned} \text{Bias}^2 &= \mathbb{E}_{\underline{X}}\left[\mathbb{E}_D\left[\hat{h}_D(\underline{X})\right] - h^*(\underline{X})\right]^2 \\ \text{Variance} &= \mathbb{E}_{\underline{X}}\left[\mathbb{E}_D\left[\left(\hat{h}_D(\underline{X}) - \mathbb{E}_D\left[\hat{h}_D(\underline{X})\right]\right)^2\right]\right] \\ \text{Noise} &= \mathbb{E}_{\underline{X},Y}\left[Y - h^*(\underline{X})^2\right] \end{aligned}}$$

9.3 Bayesian Modelling

$$\boxed{p(\underline{w}|\underline{x}_{1:n}, y_{1:n}) = \frac{p(\underline{w})p(y_{1:n}|\underline{x}_{1:n},\underline{w})}{p(y_{1:n}|\underline{x}_{1:n})}} \text{ Bayes' Rule}$$

where we assume that $\underline{w}$ is independent of x, thus $p(\underline{w}) = p(\underline{w}|\underline{x}_{1:n})$. Note that the Bayes' rule in this case does not take $\underline{x}$ into account, but just deals with y and $\underline{w}$.
Finding the parameters, that are most likely, given $\underline{x}_{1:n}, y_{1:n}$ and some a priori distribution of $\underline{w}$, is done my finding the argmax of $p(\underline{w}|\underline{x}_{1:n}, y_{1:n})$.

$$\boxed{\begin{aligned} &\operatorname*{argmax}_{\underline{w}}\left(p(\underline{w}|\underline{x}_{1:n}, y_{1:n})\right) \\ &= \operatorname*{argmin}_{\underline{w}}\left(\lambda||\underline{w}||_2^2 + \sum_{i=1}^{n}(y_i - \underline{w}^T\underline{x}_i)^2\right) \text{ for } \lambda = \frac{\sigma'^2}{\beta^2} \end{aligned}}$$

- Ridge regression can be understood as finding the MAP parameter estimate for a linear regression problem, assuming that the noise $P(y|\underline{x}, \underline{W})$ is i.i.d. Gaussian and the prior $P(\underline{w})$ on the model parameters $\underline{w}$ is Gaussian.

$$\operatorname*{argmin}_{\underline{w}}\left(\sum_{i=1}^{n}(y_i - \underline{w}^T\underline{w}_i)^2 + \lambda||\underline{w}||_2^2\right) \equiv \operatorname*{argmax}_{\underline{w}}\left(P(\underline{w}\prod_i P(y_i|\underline{x}_i, \underline{w})\right)$$

9.3.1 Regularization vs. MAP inference

A regularized estimation can often be understood as MAP inference.

$$\begin{aligned} \operatorname*{argmin}_{\underline{w}}\left(\sum_{i=1}^{n} l(\underline{w}^T\underline{x}_i; \underline{x}_i, y_i) + C(\underline{w})\right) &= \operatorname*{argmax}_{\underline{w}}\left(\prod_i P(y_i|\underline{x}_i, \underline{w})P(\underline{w})\right) \\ &= \operatorname*{argmax}_{\underline{w}}\left(P(\underline{w}|D)\right) \text{ (Bayes)} \end{aligned}$$

where $C(\underline{w}) = -\log P(\underline{w})$ and $l(\underline{w}^T\underline{x}_i; \underline{x}_i, y_i) = -\log P(y_i|\underline{x}_i, \underline{w})$

Regularization	Prior
$l1$	Laplace

9.4 Bayes' optimal classifier

- Assume the data is generated i.i.d. according to

$$(\underline{x}_i, y_i) \sim P(\underline{X}, Y)$$

- The hypthosis h^* minimizing $R(h) = \mathbb{E}_{\underline{X},Y}\left[[Y \neq h(\underline{X})]\right]$ is given by the most probable class

$$h^*(x) = \operatorname*{argmax}_{y}\left(P(Y = y|\underline{X} = \underline{x})\right)$$

9.5 Logistic Regression

- Assumption: Bernoulli noise
- Idea: Describe the probability of label y using the linear model for classification and combining it with a link function that turns $\underline{w}^T\underline{x}$ into a probability:

$$P(y = +1|\underline{x}) = \sigma(\underline{w}^T\underline{x}) = \frac{1}{1+\exp(-\underline{w}^T\underline{x})}$$

9.5.1 MLE for Logistic Regression

$$\hat{\underline{w}} = \operatorname*{argmax}_{\underline{w}}\left(\prod_{i=1}^{n} P(y_i|\underline{x}_i, \underline{w})\right)$$

Idea: Choose $\underline{w}$ such that for all observations the certainty (probability of getting the correct label y_i given feature vector $\underline{x}_i$) is as high as possible and optimize over all observations.

The negative log likelihood function can then be found as

$$\hat{R}(\underline{w}) = \sum_{i=1}^{n} \log\left(1 + \exp(-y_i\underline{w}^T\underline{x})\right)$$

Since this loss function is convex we can use optimization techniques like SGD.

9.5.2 SGD for L2-Regularized Logistic Regression

1. Intialize $\underline{w}$
2. For $t = 1, 2, \ldots$
   a) Pick data point $(\underline{x}, y)$ uniformly at random from data D
   b) Compute probability of misclassification with current model

   $$\hat{P}(Y = -y|\underline{w}, \underline{x}) = \frac{1}{1+\exp(y\underline{w}^T\underline{x})}$$

   c) Take gradient step

   $$\underline{w} \leftarrow \underline{w}(1 - 2\lambda\eta_t) + \eta_t y\underline{X}\hat{P}(Y = -y|\underline{w}, \underline{x})$$

Now since we'd like to use a regularizer to control the model complexity we estimate MAP instead of MLE:

Prior	Regularizer
Gaussian	L2
Laplace	L1

9.5.3 Kernelized Logistic Regression

- **Learning:** Find optimal weights by minimizing logistic loss and regularizer.

$$\hat{\alpha} = \operatorname*{argmin}_{\alpha}\left(\sum_{i=1}^{n}\log\left(1 + \exp\left(-y_i\alpha^T\underline{K}_i\right)\right) + \lambda\alpha^T\underline{K}\alpha\right)$$

- **Classification:** Use conditional distribution

$$\hat{P}(y|\underline{x}, \alpha) = \frac{1}{1+\exp\left(-y\sum_{j=1}^{n}\alpha_j k(\underline{x}_j, \underline{x})\right)}$$

9.6 Multi-Class Logistic Regression

- Maintain one weight vector per class an model

$$P(Y =)i|\underline{x}, \underline{w}_1, \ldots, \underline{w}_c) = \frac{\exp(\underline{w}_i^T\underline{x})}{\sum_{j=1}^{c}\exp(\underline{w}_j^T\underline{x})}$$

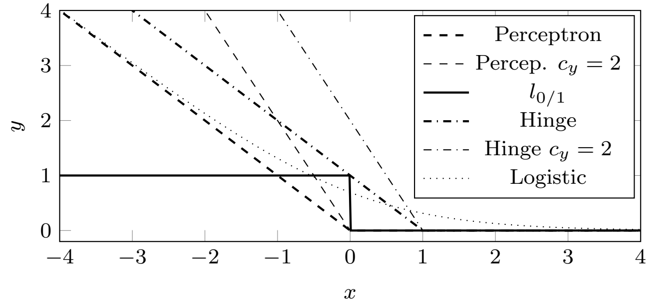
- Not unique - can enforce uniqueness by setting $\underline{w}_c = 0$ (this recovers logistic regression as special case)
- Corrsponding loss function (cross-entropy loss):

$$l(y; \underline{x}, \underline{w}_1, \ldots, \underline{w}_c) = -\log P(Y = y|\underline{x}, \underline{w}_1, \ldots, \underline{w}_c)$$

9.7 SVM vs. Logistic Regression

- SVM / Perceptron
  - \+ Sometimes higher classification accuracy
  - \+ Sparse solutions
  - \- Can't (easily) get class probabilites
- Logistic Regression
  - \+ Can obtain class probabilities
  - \- Dense solutions

9.8 Different Loss Functions



10 Bayesian Decision Theory

- Given:
  - Conditional distribution over labels $P(y|\underline{x})$

- Set of actions $\mathcal{A}$
- Cost function $C : \mathcal{Y} \times \mathcal{A} \to \mathbb{R}$

- Bayesian Decision Theory recommends to pick the action that minimizes the expected cost

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \left(\mathbb{E}_y \left[C(y, a) | \underline{x} \right] \right)$$

- If we had access to the true distribution $P(y|\underline{x})$ this decision implements the **Bayesian Optimal Decision**.

10.1 Example with Logistic Regression

- Est. cond. dist: $\hat{P}(y|\underline{x}) = \text{Ber}(y; \sigma(\underline{\hat{w}}^T \underline{x}))$
- Action set: $\mathcal{A} = \{+1, -1\}$
- Then the action that minimizes the expected cost is the most likely class:

$$a^* = \operatorname*{argmax}_y \left(\hat{P}(y|\underline{x}) = \text{sign}(\underline{w}^T \underline{x}) \right)$$

10.2 Asymmetric Cost

$$C(y, a) = \begin{cases} c_{FP} & \text{if } y = -1 \text{ and } a = +1 \\ c_{FN} & \text{if } y = +1 \text{ and } a = -1 \\ 0 & \text{otherwise} \end{cases}$$

which is equivalent to

$$C(y, a) = c_{FN} \max(y - a, 0) + c_{FP} \max(a - y, 0)$$

Then the action that minimizes the cost is

$$c_+ = \mathbb{E}_Y \left[c(Y, +1) | \underline{x} \right] = c_{FP} P(Y = -1|\underline{x}) = c_{FP}(1 - p)$$
$$c_- = \mathbb{E}_Y \left[c(Y, -1) | \underline{x} \right] = c_{FN} P(Y = +1|\underline{x}) = c_{FN} p$$

Therefore we predict $+1$ if $c_+ < c_-$ and vice versa.

$$\begin{aligned} c_+ &< c_- \\ c_{FP}(1 - p) &< c_{FN} p \\ p &> \frac{c_{FP}}{c_{FP} + c_{FN}} \end{aligned}$$

10.2.1 Doubtful Logistic Regression

Idea: Pick most likely class only if confident enough.

- Est. cond. dist.: $\hat{P}(y|\underline{x}) = \text{Ber}(y; \sigma(\underline{\hat{w}}^T \underline{x}))$
- Action set: $\mathcal{A} = \{+1, -1, D\}$
- Cost functions:

$$C(y, a) = \begin{cases} [y \neq a] & \text{if } a \in \{+1, -1\} \\ c & \text{if } a = D \end{cases}$$

- the the action that minimizes the expected cost is given by

$$a^* = \begin{cases} y & \text{if } \hat{P}(y|\underline{x}) \geq 1 - c \\ D & \text{otherwise} \end{cases}$$

11 Generative vs. Discriminative Modelling

Idea: Model $P(\underline{x})$ to gain the capability of detecting outliers (unusual points for which $P(\underline{x})$ is very small), since models only estimating conditional distributions $P(y|\underline{x})$ can't do that.

Definition 16. ***Discriminative models*** *aim to estimate* $P(y|\underline{x})$.

Definition 17. ***Generative models*** *aim to estimate the joint distribution* $P(y, \underline{x})$.

11.1 Typical Approach on Generative Modelling

1. Estimate prior on labels $P(y)$.
2. Estimate conditional distribution $P(\underline{x}|y)$ for each class.
3. Obtain predictive distribution using Bayes' rule:

$$P(y|\underline{x}) = \frac{1}{Z = P(x)} P(y) P(\underline{x}|y)$$

We can derive the conditional distribution from the joint distribution but not vice versa!

11.2 Gaussian Naive Bayes Model $p(\underline{x}_i, y) = p(\underline{x}_i|y)p(y)$

- **Used for continuous features!** If not use categorical model.
- Model class label as generated from categorical variable

$$P(Y = y) = p_y \qquad q \in \mathcal{Y} = \{1, \ldots, c\}$$

  Assumption: Features are conditionally independent if the class label is known. ($\Sigma_y = \text{diag}(\cdots)$)

- **Learning:** Given data $D = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$
  - MLE for class prior:

$$\hat{P}(Y = y) = \hat{p}_y = \frac{\text{Count}(Y=y)}{n}$$

  - MLE for feature distribution:

$$\begin{aligned} \hat{P}(x_i|y) &= \mathcal{N}(x_i; \hat{\mu}_{y,i}, \sigma^2_{y,i}) \\ \hat{\mu}_{y,i} &= \frac{1}{\text{Count}(Y = y)} \sum_{j : y_i = y} x_{j,i} \\ \sigma^2_{y,i} &= \frac{1}{\text{Count}(Y = y)} \sum_{j : y_j = y} (x_{i,j} - \hat{\mu}_{y,i})^2 \end{aligned}$$

  where $x_{j,i}$ represents the value of feature i for instance j (x_j, y_i).

- **Prediction** given point $\underline{x}$

$$y = \operatorname*{argmax}_{y'} \left(\hat{P}(y'|\underline{x}) \right) = \operatorname*{argmax}_{y'} \left(\hat{P}(y') \prod_{i=1}^{n} \hat{P}(x_i|y') \right)$$

- Decision Rules for binary classification

$$y = \text{sign} \left(\log \frac{P(Y=1|\underline{x})}{P(Y=-1|\underline{x})} \right)$$

  It is easy to verify that the above returns $+1$ if $P(Y = 1|\underline{x}) >= 0.5$.

Definition 18. *The function* $f(\underline{x}) = \log \frac{P(Y=1|\underline{x})}{P(Y=-1|\underline{x})}$ *is called* ***discriminant function***.

- GNB ($c = 2$), in case of shared variance, produces a linear classifier:

$$f(\underline{x}) = \underline{w}^T \underline{x} + w_0$$

  where $w_0 = \log \frac{\hat{p}_+}{1 - \hat{p}_+} + \sum_{i=1}^{d} \frac{\hat{\mu}^2_{-,i} - \hat{\mu}_{+,i}{}^2}{2\hat{\sigma}^2_i}$ and $w_i = \frac{\mu_{+,i} - \mu_{-,i}}{\sigma^2_i}$

- Connection of discriminant function, class probability and link function:

$$P(Y = 1|\underline{x}) = \frac{1}{1 + \exp(-f(\underline{x}))} = \sigma(f(\underline{x}))$$

- GNB with shared variance and $c = 2$ will make the same predictions as logistic regression if the model assumptions are met.

11.2.1 Issue with NBM

- If there is a conditional correlation between class labels, then the assumption of independence between features is violated.
- Due to the independence assumption predictions can become overconfident.
- This is alright if we care only about the most likely class, but not if we want to use probabilities for making decisions (e.g. asymmetric losses etc).

11.3 Gaussian Bayes Classifiers (not naive)

- In contrast to GNB general gaussian bayes classifiers model features as generated by multivariate Gaussians

$$P(\underline{x}|y) = \mathcal{N}(\underline{x}; \mu_y, \Sigma_y)$$

- **Learning** given $D = \{(\underline{x}_1, y_1), \ldots, (\underline{x}_n, y_n)\}$
  - MLE for class label distribution

$$\hat{P}(Y = y) = \hat{p}_y = \frac{\text{Count}(Y=y)}{n}$$

  - MLE for feature distribution

$$\begin{aligned} \hat{P}(\underline{x}|y) &= \mathcal{N}(\underline{x}; \hat{\mu}_y, \hat{\Sigma}_y) \\ \hat{\mu}_y &= \frac{1}{\text{Count}(Y = y)} \sum_{i : y_i = y} \underline{x}_i \\ \hat{\Sigma}_y &= \frac{1}{\text{Count}(Y = y)} \sum_{i : y_i = y} (\underline{x}_i - \hat{\mu}_y)(\underline{x}_i - \hat{\mu}_y)^T \end{aligned}$$

- Discriminant functions for GBCs
  - Want: $f(\underline{x}) = \log \frac{P(Y=1|\underline{x})}{P(Y=-1|\underline{x})}$
  - This is given by

$$f(\underline{x}) = \log \frac{p}{1-p} +$$
$$\frac{1}{2} \left[\log \frac{|\hat{\Sigma}_-|}{|\hat{\Sigma}_+|} + \left((\underline{x} - \hat{\mu}_-)^T \hat{\Sigma}_-^{-1} (\underline{x} - \hat{\mu}) \right) - \left((\underline{x} - \hat{\mu}_+)^T \hat{\Sigma}_+^{-1} (\underline{x} - \hat{\mu}_+) \right) \right]$$

  where $p = P(Y = 1)$.

11.4 Fisher's Linear Discriminant Analysis

Assumptions:

- $p = 0.5, c = 2$
- Equal covariances $\hat{\Sigma}_- = \hat{\Sigma}_+ = \hat{\Sigma}$

$$f(\underline{x}) = \underline{x}^T \underbrace{\hat{\Sigma}^{-1}(\hat{\mu}_+ - \hat{\mu}_-)}_{\underline{w}} + \underbrace{\frac{1}{2}(\hat{\mu}_-^T \hat{\Sigma}^{-1}\hat{\mu}_- - \hat{\mu}_+^T \hat{\Sigma}^{-1}\hat{\mu}_+)}_{w_0}$$

Under these circumstances the prediction is

$$y = \text{sign}(f(\underline{x})) = \text{sign}(\underline{w}^T \underline{x} + w_0)$$

- If the model assumptions are met, LDA will make the same predictions as Logistic Regression.
- **LDA vs. PCA**: LDA can be viewed as a projection to a 1-dimensional subspace that maximizes the ratio of between-class an within-class variances (very little variance within classes, alot across classes), where in contrast PCA ($k = 1$) maximizes the variance of the resulting 1-dimensional projection. PCA does not separate the classes.

11.5 Quadratic Discriminant Analysis

In the general case

$$f(\underline{x}) = \log\left(\frac{p}{1-p}\right) +$$
$$\frac{1}{2}\left[\log\left(\frac{|\hat{\Sigma}_-|}{|\hat{\Sigma}_+|}\right) + \left((\underline{x} - \hat{\mu}_-)^T \hat{\Sigma}_-^{-1}(\underline{x} - \hat{\mu}_-)\right) - \left((\underline{x} - \hat{\mu}_+)^T \hat{\Sigma}_+^{-1}(\underline{x} - \hat{\mu}_1)\right)\right]$$

and we predict

$$y = \text{sign}(f(\underline{x}))$$

which is called **quadratic discriminant analysis.**

11.6 General Comparison

- Fisher's LDA
  - – Generative models, i.e. models $P(\underline{X}, Y)$
  - \+ Can be used to detect outliers: $P(\underline{X}) < t$
  - • Assumes normality of $\underline{X}$
  - \- Not very robust against violation of this assumption
- Logistic Regression
  - • Discriminative model, i.e. models $P(Y|\underline{X})$ only
  - \- Cannot detect outliers
  - • Makes no assumptions on $\underline{X}$
  - \+ More robust
- Gaussian Naive Bayes Models
  - \- Conditional independence assumption may lead to overconfidence
  - \+ Predictions might still be useful
  - \+ Number of parameters $= O(cd)$
  - • Complexity (memory + interface) is linear in d
- General Gaussian Bayes Models
  - \+ Captures correlations among features
  - \+ Avoids overconfidence
  - \- Number of parameters $= O(cd^2)$
  - \- Complexity quadratic in d

11.7 Adaptation to Discrete Features

- Suppose X_i take discrete values
- Since generative models allow to swap the distribution easily we can use different models like
  - Bernoulli
  - Categorical
  - Multinominal

11.7.1 Categorical Naive Bayes Classifier

- Model class labels as generated from categorical variable.

$$P(Y = y) = p_y \qquad y \in \mathcal{Y} = \{1, \dots, c\}$$

- Model features by (conditionally) independent categorical random variables

$$P(X_i = c|Y = y) = \theta^{(i)}_{c|y}$$

- MLE for CNBC
  - Given $D\{(\underline{x}_1, y_1), \dots (\underline{x}_n, y_n)\}$
  - MLE for class label distribution $\hat{P}(Y = y) = \hat{p}_y$

$$\hat{p}_y = \frac{\text{Count}(Y=y)}{n}$$

  - MLE for distribution of feature i $\hat{P}(X_i = c|y) = \theta^{(i)}_{c|y}$

$$\theta^{(i)}_{c|y} = \frac{\text{Count}(X_i=c, Y=y)}{\text{Count}(Y=y)}$$

  - Prediction given new point $\underline{x}$

$$y = \underset{y'}{\text{argmax}}\left(\hat{P}(y'|\underline{x})\right) = \underset{y'}{\text{argmax}}\left(\hat{P}(y')\prod_{i=1}^{d}\hat{P}(x_i|y')\right)$$

11.7.2 Combination of Discrete and Continuous Features

- The (N)BC does not require each feature to follow the same type of conditional distribution
- Training and prediction is the same as before

11.8 Avoiding Overfitting of MLE

- Restrict Model Class (assumptions on covariance structure, e.g. GNB) thus using fewer parameters
- Regularization is equivalent to putting a prior on the weights to describe class probabilities.

11.8.1 Prior over parameters ($c = 2$)

- Prior on class probabilities: $P(Y = 1) = \theta$
- MLE: $\hat{\theta} = \frac{\text{Count}(Y=1)}{n}$
- Extreme case: $n = 1$, in which case the empirical frequencies are not informative and the MLE collapses.
- Solution: Assume prior knowledge in the form of a probability distribution → use beta priors.

Conjugate Distributions

Definition 19. *A pair of prior distributions and likelihood functions is called **conjugate** if the posterior distribution remains in the same family as the prior.*
This enables a simple and efficient calculation of prior distributions.

Example

- Prior: $\text{Beta}(\theta; \alpha_+, \alpha_-)$
- Observations: Suppose we observe n_+ positive and n_- negative labels.
- Posterior: $\text{Beta}(\theta; \alpha_+ + n_+, \alpha_- + n_-)$
- thus α_+, α_- act as pseudo-counts.

MAP estimate

$$\hat{\theta} = \underset{\theta}{\text{argmax}}\,(P(\theta|y_1, \dots, y_n; \alpha_+, \alpha_-)) = \frac{\alpha_+ + n_+ - 1}{\alpha_+ + n_+ + \alpha_- + n_- - 2}$$

Conjugate Priors

Prior/Posterior	Likelihood function
Beta	Bernoulli/Binomial
Dirichlet	Categorical/Multinomial
Gaussian (fixed covariance)	Gaussian
Gaussian-inverse Wishart	Gaussian
Gaussian process	Gaussian

12 Dealing with Missing Data

12.1 Gaussian Mixtures

$$P(\underline{x}|\theta) = P(\underline{x}|\mu, \Sigma, \underline{w}) = \sum_{i=1}^{k} w_i \mathcal{N}(\underline{x}; \mu_i, \Sigma_i)$$

where $w_i \geq 0$ and $\sum_i w_i = 1$ and $P(\underline{x}|\theta)$ is a convex combination of gaussian distributions.

$$(\mu^*, \Sigma^*, w^*) = \text{argmin}\left(-\sum_i \log \sum_{j=1}^{k} w_j \mathcal{N}(\underline{x}_i|\mu_j, \Sigma_j)\right) \quad \text{MLE}$$

- This objective function is nonconvex.
- Challenges for stochastic descent:
  - Convariance matrices must remain symmetric positive definite, which as constraints might be difficult to maintain

- The joint distribution $P(z, \underline{x}) = w_z \mathcal{N}(\underline{x}|\mu_z, \Sigma_z)$ is identical to the generative model used by the GBC.
- If we have the labels to our training data, we can find the parameters by computing the MLE in closed form as done for GBC.
- **Initialization:**
  - For weights: Uniform Distribution
  - For means: Random initialization or k-means++
  - For variances: Initialize as spherical, e.g. according to empirical variance in the data.
- **Selecting** k: Similar challenge to selecting number of clusters, in contrast to k-means, here cross-validation works well. Aim to maximize log-likelihood on validation set.
- **Degeneracy**: Given a single data point the loss converges to $-\infty$ as $\mu = x$, $\sigma \to 0$. Thus the optimal GMM chooses $k = n$ and puts one Gaussian around each data point with variance tending to 0. The solution lies in adding a small term to the diagonal of the MLE:

$$\Sigma_j^{(t)} \leftarrow \frac{\sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)(\underline{x}_i - \mu_j^{(t)})(\underline{x}_i - \mu_j^{(t)})^T}{\sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)} + \nu^2 \mathbb{I}$$

12.1.1 Hard-EM

1. Initialize the parameters $\theta^{(0)}$ where $\theta^{(*)} = \left[w_{1:c}^{(*)}, \mu_{1:c}^{(*)}, \Sigma_{1:c}^{(*)}\right]$.
2. For $t = 1, 2, \dots$
   a) E-Step: Predict most likely class for each data point

$$z_i^{(t)} = \underset{z}{\operatorname{argmax}} \left(P(z|\underline{x}_i, \theta^{(t-1)}) \right)$$

$$= \underset{z}{\operatorname{argmax}} \left(\underbrace{P(z|\theta^{(t-1)})}_{w_z^{(t-1)}} \underbrace{P(\underline{x}_i|z, \theta^{(t-1)})}_{\mathcal{N}(\underline{x}_i|\mu_z^{(t-1)}, \Sigma_z^{(t-1)})} \right)$$

   b) Now we got complete data $D^{(t)} = \{(\underline{x}_1, z_1^{(t)}), \dots, (\underline{x}_n, z_n^{(t)})\}$
   c) M-Step: Compute MLE as for the GBC

$$\theta^{(t)} = \underset{\theta}{\operatorname{argmax}} \left(P(D^{(t)}|\theta) \right)$$

Problems with Hard EM

- Points are assigned a fixed label, even though the model is uncertain.
- Intuitively, this tries to extract too much information from a single point.
- In practice, this may work poorly if clusters are overlapping.

k-Means Algorithm vs. EM for GMM

- Can understand k-Means Algorithm (Lloyd's heuristic) as special case of Hard-EM for GMMs
  - Uniform weights over mixture components
  - Assuming identical, spherical covariance matrices
- Can also understand k-Means Algorithm as limiting case of Soft-EM for GMM
  - Assumptions same as above, with additionally variances tending to 0

$$\gamma_j(\underline{x}) = \begin{cases} 1 & \text{if } \mu_j \text{ i closest to } x \\ 0 & \text{else} \end{cases}$$

12.1.2 Posterior Probabilities

- Suppose we knew $P(z|\theta)$, the distribution of the labels given the model parameters and $P(\underline{x}|z, \theta)$, the distribution of the features given the labels and parameters.
- Compute a posterior distribution over cluster membership, thus inferring distributions over latent (hidden) variables z.

$$\gamma_j(\underline{x}) = P(z|\underline{x}, \Sigma, \mu, \underline{w})$$

$$= \frac{p(\underline{x}|z, \theta)p(z|\theta)}{p(\underline{x}|\theta)} = \frac{w_j P(\underline{x}|\Sigma_j, \mu_j)}{\Sigma_l w_l P(\underline{x}|\Sigma_l, \mu_l}$$

- Hard EM takes the argmax of the above distribution, where Soft EM records the likelihood of each class for each point.

12.2 Expectation-Maximization (Soft-EM)

- While not converged
  1. E-Step: Calculate cluster membership weights („Expected sufficient statistics"for each point (aka „responsibilities"))
     Calculate $\gamma_j^{(t)}(\underline{x}_i)$ for each i and j given estimates of $\mu^{(t-1)}, \Sigma^{(t-1)}, \underline{w}^{(t-1)}$ from previous iteration
  2. M-Step: Fit clusters to weighted data points (closed form Maximum likelihood solution)

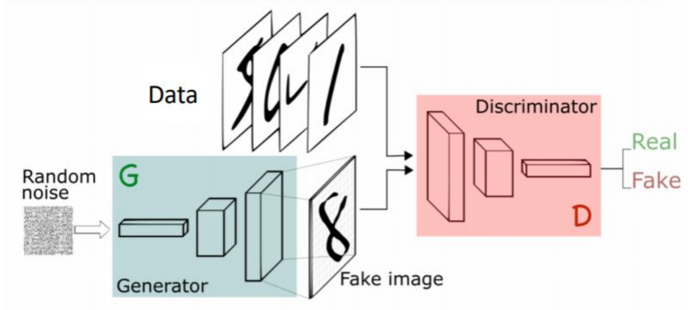
$$w_j^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)$$

$$\mu_j^{(t)} \leftarrow \frac{\sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)\underline{x}_i}{\sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)}$$

$$\Sigma_j^{(t)} \leftarrow \frac{\sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)(\underline{x}_i - \mu_j^{(t)})(\underline{x}_i - \mu_j^{(t)})^T}{\sum_{i=1}^{n} \gamma_j^{(t)}(\underline{x}_i)}$$

12.3 Implicit Generative Models

- Given sample of unlabelled points $\underline{x}_1, \dots, \underline{x}_n$
- Goal: Learn model $\underline{X} := G(\underline{Z}, \underline{w})$ where Z is a simple distribution (e.g. lowdimensional Gaussian) and G some flexible nonlinear function (neural net)
- Key challenge: Hard to compute likelihood of the data.
- Possible solution: Generative adversarial networks



- Simultaneously train two neural networks
  - Generator G tries to produce realistic examples
  - Discriminator D tries to detect 'fake' examples
- Can view as a game

$$D : \mathbb{R}^d \to [0, 1] \text{ wants } D(x) = \begin{cases} \approx 1 & \text{if } x \text{ is 'real'} \\ \approx 0 & \text{if } x \text{ is 'fake'} \end{cases}$$

$$G : \mathbb{R}^m \to \mathbb{R}^d \text{ wants } D(G(z)) \approx 1 \text{ for samples } z$$

$$\min_{\underline{w}_G} \max_{\underline{w}_D} \underbrace{\mathbb{E}_{X \sim \text{Data}} \log D(\underline{x}; \underline{w}_D) + \mathbb{E}_{Z \sim \text{Noise}} \log(1 - D(G(\underline{z}; \underline{w}_G)))}_{M(G,D)}$$

- Training a GAN requires finding a saddle point rather than a (local) minimum.

$$\underline{w}_G^{(t+1)} \leftarrow \underline{w}_G^{(t)} - \eta_t \Delta_{\underline{w}_G} M(\underline{w}_G, \underline{w}_D^{(t)})$$

$$\underline{w}_D^{(t+1)} \leftarrow \underline{w}_D^{(t)} + \eta_t \Delta_{\underline{w}_D} M(\underline{w}_G^{(t)}, \underline{w}_D)$$