Introduction to Machine Learning

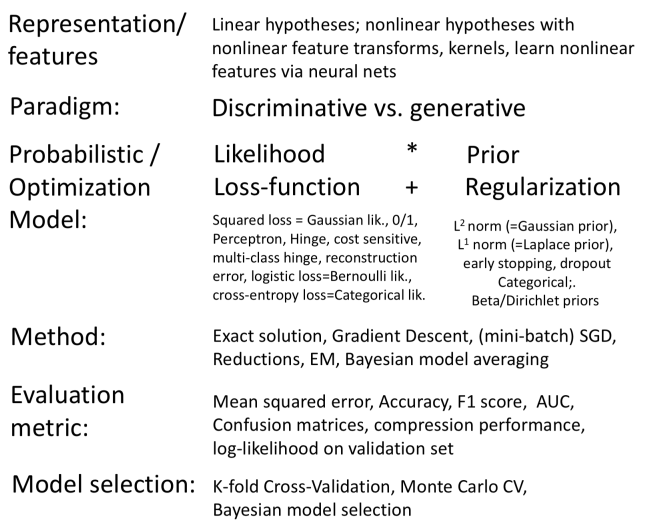
ETH Zurich

Janik Schuettler
Marcel Graetz

FS18

Contents

1 Overview



Representation/ features	Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms, kernels, learn nonlinear features via neural nets		
Paradigm:	Discriminative vs. generative		
Probabilistic / Optimization Model:	Likelihood Loss-function Squared loss = Gaussian lik., 0/1, Perceptron, Hinge, cost sensitive, multi-class hinge, reconstruction error, logistic loss=Bernoulli lik., cross-entropy loss=Categorical lik.	* +	Prior Regularization L^2 norm (=Gaussian prior), L^1 norm (=Laplace prior), early stopping, dropout Categorical;. Beta/Dirichlet priors
Method:	Exact solution, Gradient Descent, (mini-batch) SGD, Reductions, EM, Bayesian model averaging		
Evaluation metric:	Mean squared error, Accuracy, F1 score, AUC, Confusion matrices, compression performance, log-likelihood on validation set		
Model selection:	K-fold Cross-Validation, Monte Carlo CV, Bayesian model selection		

I Supervised Learning

2 Regression and Gradient Descent

We try to fit a function to training data (learning) to make predictions. Our goal is to learn real-valued mapping $f : \mathbb{R}^d \to \mathbb{R}$.

The general model is

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i + b = \mathbf{w}^T\mathbf{x} + b = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$$

with $\tilde{\mathbf{x}} = (x_1, ..., x_d, 1)$ and $\tilde{\mathbf{w}} = (w_1, ..., w_d, b)$.

Model error We measure goodness of a model (i.e. fit) using a p-loss function $l_p(w, x, y)$,

$$\hat{R}(w) = \sum_{i=1}^{n} l_p(w, x, y) = \sum_{i=1}^{n} |y_i - w^T x_i|^p, \quad p \geq 1.$$

For $p = 2$, we get the least squares measure $\hat{R}(w) = \sum_{i=1}^{n}(y_i - w^T x_i)^2$.

Optimization problem We want to find the optimal weight vector

$$w^* = \operatorname*{argmin}_w(\hat{R}) = \operatorname*{argmin}_w \sum_{i=1}^{n}(y_i - w^T x_i)^2.$$

2.1 Closed-form Solution: Linear Least Squares

Closed-form solution is $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

Complexity for solving in closed form is $\mathcal{O}(nd^2 + d^3)$.

2.2 Optimization: Gradient Descent

Theorem 2.1 (Gradient descent) *Let f be convex with w^* the global minimizer. Assume $\|w_1 - w^*\| \leq D$ and $\|\nabla f(w)\| \leq G\ \forall w \in \mathbb{R}^d, B_D(w^*)$. If we choose $\eta_t = \frac{D}{G\sqrt{t}}$, then*

$$f(\overline{w}_T) - f(w^*) \leq \frac{GD}{\sqrt{T}}.$$

Least squares function is convex. Gradient descent finds an optimal solution with better complexity.

Complexity for one iteration of gradient descent is $\mathcal{O}(dn)$.

Problem: For low steps size very slow, but for high step size this can diverge.

Adaptive step size Examples of how to update the step size adaptively.

Algorithm 1 Gradient Descent

```
1: $w_0 \in \mathbb{R}$                                  ▷ start with arbitrary $w_0$
2: for $t = 1, 2, \ldots, T$ do
3:     $\nabla_w \hat{R}(w_t) = -2\sum_{i=1}^{n}(y_i - w_t^T x_i)x_i$
4:     $w_{t+1} = w_t - \eta_t \nabla \hat{R}(w_t)$      ▷ $\eta_t$ is the learning rate
5: return $w_T$
```

- **Line search**: Optimizing step size every step
$$\eta_t \leftarrow \operatorname*{argmin}_{\eta \in \mathbb{R}^+} \hat{R}(w_t - \eta \nabla \hat{R}(w_t))$$
$$w_{t+1} \leftarrow w_t - \eta_t \nabla \hat{R}(w_t)$$
- **Bold driver heuristic**: If function decreases, increase step size, else decrease step size
$$\eta_{t+1} \leftarrow \begin{cases} c_{inc}\eta_t, & \text{if } \hat{R}(w_{t+1}) < \hat{R}(w_t) \\ c_{dec}\eta_t, & \text{if } \hat{R}(w_{t+1}) > \hat{R}(w_t) \end{cases}$$

whut?

GD convergence Stop if either

- gradient is small enough, or
- difference in objective between subsequent iterations is small enough.

2.3 Non-linear Regression via Linear Regression

Non-linear basis functions are used to fit non-linear data via linear regression

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x})$$

In $2D$, ϕ could be $\phi(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1x_2, \ldots)$.

2.4 Model selection

We would like to choose the model that optimizes trade-off between model complexity and training error, i.e. under- and overfitting data. Mathematically, we try to minimize the true risk

$$R(\mathbf{w}) = \mathbb{E}_{\mathbf{x},y}[(y - \mathbf{w}^T\mathbf{x})^2] = \int \mathbb{P}(x, y)(y - w^T x)^2 \, dx \, dy,$$
$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} R(\mathbf{w})$$

However, we can only compute the empirical risk

$$\hat{R}_D(\mathbf{w}) = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D}(y - \mathbf{w}^T\mathbf{x})^2, \quad \hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \hat{R}_{train}(\mathbf{w}).$$

Theorem 2.2 (Law of large numbers (LLN)) $\hat{R}_D(w) \to R(w)$ *for any fixed w almost surely as $|D| \to \infty$.*

iid assumnption assumes that the data set is generated independently and identically distributed (iid), $(\mathbf{x}_i, y_i) \sim P(\mathbf{X}, Y)$.

Convergence of learning For learning via empirical risk minimization to be successful, we need uniform convergence $\sup_{\mathbf{w}} |R(\mathbf{w}) - \hat{R}_D(\mathbf{w})| \to 0$ for $|D| \to \infty$, which is not implied by LLN, but depends on model class.

Splitting data set Do not test a model on the training data, because

$$\mathbb{E}[\hat{R}_{train}(\hat{\mathbf{w}})] < \mathbb{E}[R(\hat{\mathbf{w}})].$$

Best practice is to split the data set into a training set D_{train} and a testing set D_{test}. Optimize $\mathbf{w}$ on D_{train}

$$\mathbf{w} = \operatorname*{argmin}_{\mathbf{w}} \hat{R}_{train}(\mathbf{w})$$

and evaluate it on D_{test}

$$\hat{R}_{test}(\hat{\mathbf{w}}) = \frac{1}{|D_{test}|}\sum_{(\mathbf{x},y)\in D_{test}}(y - \hat{\mathbf{w}}^T\mathbf{x})^2.$$

Then $\mathbb{E}_{D_{train},D_{test}}[\hat{R}_{D_{test}}(\hat{w}_{D_{train}})] = \mathbb{E}_{D_{test}}[R(\hat{w}_{D_{train}})]$.

2.5 Cross validation

Test error $\hat{R}_{test}$ is itself random. Variance usually increases for more complex models. Idea is to use and average over multiple test sets to reduce the bias. Note that this only works for i.i.d. data.

Algorithm 2 Model selection with cross validation

```
1: for each candidate model $m$ do
2:     for $i = 1 : k$ do
3:         $D = D^{(i)}_{train} \uplus D^{(i)}_{validation}$          ▷ split data set
4:         $\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} \hat{R}^{(i)}_{train}(\mathbf{w})$          ▷ train model
5:         $\hat{R}^{(i)}_m = \hat{R}^{(i)}_{val}(\hat{\mathbf{w}}_i)$          ▷ estimate error
6: $\hat{m} = \text{argmin}_m \frac{1}{k}\sum_{i=1}^k \hat{R}^{(i)}_m$          ▷ select model
```

Splitting test sets may be obtained by different procedures.

- **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.
- **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. This is the default.

Number of folds k is hard too choose.

- 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 fine), but higher computational complexity
- in practice mostly $k = 5$ or $k = 10$

2.6 Regularization

For non-linear models, there might not be a natural order of complexity like with monomials, i.e. higher order monomial account for more complex models. Too complex models often manifest in large weights. The idea of regularization is to keep models simple by punishing large weights.

Ridge regression problem adds the term $\lambda\|\mathbf{w}\|_2^2$

$$\mathbf{w}^* = \underset{w}{\text{argmin}}(\hat{R}) + \lambda\|\mathbf{w}\|_2^2 = \underset{w}{\text{argmin}} \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda\|\mathbf{w}\|_2^2.$$

for some $\lambda \in \mathbb{R}$. Using the homogeneous representation, the constant term is not being regularized. λ balances

$\lambda \to \infty$, the optimization problem tries to minimize $\mathbf{w}$ only,

$\lambda \to 0$, optimization problem with no regularization.

Closed-form solution is $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$. Matrix $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$ is always invertible and better conditioned.

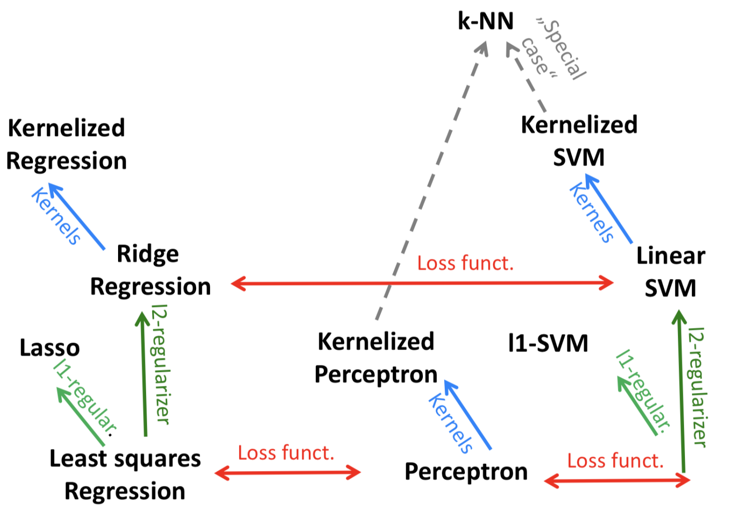
Renormalizing data ensures that each feature has zero mean and unit variance, because scaling does matter for regularization.

$$\tilde{x}_{i,j} = (x_{i,j} - \hat{\mu}_j)/\hat{\sigma}_j^2, \quad \text{where}$$
$$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^n x_{i,j} \quad \hat{\sigma}_j^2 = \frac{1}{n}\sum_i (x_{i,j} - \hat{\mu}_j)^2$$

Algorithm 3 Gradient Descent with regularization

```
1: $w_0 \in \mathbb{R}$          ▷ start with arbitrary $w_0$
2: for $t = 1, 2, \ldots, T$ do
3:     $\nabla_w \hat{R}(w_t) = -2\sum_{i=1}^n (y_i - w_t^T x_i) x_i$
4:     $w_{t+1} = w_t(1 - 2\lambda\eta_t) - \eta_t \nabla \hat{R}(w_t)$          ▷ $\eta_t$ is the learning rate
5: return $w$
```

Choosing λ is mostly done using cross-validation for logarithmically spaced $\lambda \in \{10^{-6}, 10^{-5}, \ldots, 10^5, 10^6\}$.



3 Classification

We would like to assign data points X a label Y, i.e. Y is discrete.

3.1 Linear Classification

Linear Classification seems restrictive, but using more dimensions and the right features often works quite well. Prediction is typically efficient.

Loss functions We want to find the optimal weight vector $\mathbf{w}^* = \text{argmin}_w \sum_{i=1}^n l(\mathbf{w}, \mathbf{x}_i, y_i)$. Possible loss-functions are

- **0/1-loss**, minimizing the number of errors, an intractable (non-convex) loss-function

$$l(\mathbf{w}, \mathbf{x}_i, y_i) = l_{0/1}(\mathbf{w}, \mathbf{x}_i, y_i) = \begin{cases} 1 & \text{if } y_i \neq \text{sign}(\mathbf{w}^T\mathbf{x}_i) \\ 0 & \text{else} \end{cases},$$

- **perceptron-loss**, a tractable surrogate-loss function

$$l(\mathbf{w}, \mathbf{x}_i, y_i) = l_p(\mathbf{w}, \mathbf{x}_i, y_i) = \max(0, -y_i\mathbf{w}^T\mathbf{x}_i),$$

- **hinge-loss (SVM)**

$$l(\mathbf{w}, \mathbf{x}_i, y_i) = l_h(\mathbf{w}, \mathbf{x}_i, y_i) = \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i).$$

was bedeutet immer dieses traceable? Du meinst tractable?

3.1.1 Perceptron and Stochastic Gradient Descent

Perceptron optimization problem

$$\mathbf{w}^* = \underset{w}{\text{argmin}} \sum_{i=1}^n l_p(\mathbf{w}, \mathbf{x}_i, y_i) = \underset{w}{\text{argmin}} \sum_{i=1}^n \max(0, -y_i\mathbf{w}^T\mathbf{x}_i).$$

Perceptron Gradient

$$\nabla_w \hat{R}_p(\mathbf{w}) = -\sum_{i:y_i \neq \text{sign}(\mathbf{w}^T\mathbf{x}_i)} y_i\mathbf{x}_i$$

Stochastic Gradient Descent (SGD) picks uniformly and at random m data points to compute the gradient (mini-batch SGD)

$$\nabla\hat{R}(w) = \frac{1}{n}\sum_I \nabla l(w; x_I, y_I) = \mathbb{E}_{I \sim Unif\{1,\ldots,n\}}[\nabla l(w; x_I, y_I)].$$

Algorithm 4 Stochastic Gradient Descent

```
1: $w_0 \in \mathbb{R}$          ▷ start with arbitrary $w_0$
2: for $t = 1, 2, \ldots$ do
3:     pick $(x'_i, y'_i)_{i=1}^m \in D$          ▷ Random data points
4:     $w_{t+1} = w_t - \eta_t \nabla l(w_t, x', y')$
5: return $w$
```

Algorithm 5 Perceptron with Stochastic Gradient Descent

```
1: $w_0 \in \mathbb{R}$                         ▷ start with arbitrary $w_0$
2: for $t = 1, 2, \ldots$ do
3:     pick $i_t \sim \text{Unif}\{1, \ldots, n\}$     ▷ one random data point
4:     if $y_{i_t} \neq \text{sign}(w_t^T x_{i_t})$ then    ▷ Perceptron gradient
5:         $w_{t+1} = w_t + \eta_t y_{i_t} x_{i_t}$
6:     else
7:         $w_{t+1} = w_t$
```

Robbins-Monro Conditions keep the learning rate η_t such that the algorithm will not terminate before converging, i.e. $\sum_t \eta_t = \infty$, but with bound variance, i.e. $\sum_t \eta_t^2 < \infty$. These conditions are sufficient for convergence. For example $\eta_t = 1/t$ or $\eta_t = \min(c_1, c_2/t)$.

Remark 3.1 *This is **the** perceptron algorithm. (Says probelm set 3.)*

Adaptive learning rates are used by algorithms such as AdaGrad, RMSProp, Adam.

Theorem 3.2 *If the data is linearly separable, the Perceptron will obtain a linear separator.*

SGD convergence criteria Stop if either

- a fixed number of iterations have passed,
- GD conditions would suggest convergence (occasionally, say every n-th iteration, compute full objective value/ gradient magnitude),
- error on separate validation data set is small enough (direct monitoring). This is a special form of regularization called early stopping.

3.1.2 Support Vector Machines (SVM)

The hinge-loss encourages not only correct classification, but correct classification with maximal margin to the decision boundary.

Can this lead to non-optimal decisions in case of e.g. separability?

SVM optimization problem

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

Theorem 3.3 *SVM finds solution with max margin to decision boundary.*

Choosing λ is mostly done using cross-validation. Note that instead of using the hinge-loss for validation, one would use the target performance metric (e.g. the number of mistakes).

3.2 Feature Selection

In many high-dimensional problems, we may prefer not to work with potentially available features, because of

- interpretability and generalization: simple models may be understood and generalize better to more complex tasks,
- storage/ computation/ cost: it is unnecessary to compute with and store unused or less important features or features that depend upon another. Also, acquiring features might be expensive, so one might prefer not to acquire a feature if it is not needed.

3.2.1 Greedy feature selection

Greedily add or remove features to maximize cross-validated prediction accuracy, mutual information, or other notions of informativeness.

For a set of features $V = \{1, \ldots, d\}$, define a cost function $\hat{L} : \mathcal{P}(V) \to \mathbb{R}$ to be the cross-validation error using features in subsets of V only.

Comparison Forward is usually fast (if few relevant features), backward can handle "dependent" features.

Problems Both algorithms are computationally expensive (need to retrain models many times for different feature combinations) and can be suboptimal. As an extreme counter example consider a setting in which all features are uninformative on their own, but informative altogether.

Algorithm 6 Greedy forward selection

```
1: $S = \emptyset$, $E_0 = \infty$
2: for $t = 1 : d$ do
3:     $s_i = \text{argmin}_{j \in V \setminus S} \hat{L}(S \cup \{j\})$     ▷ Find best element to add
4:     $E_i = \hat{L}(S \cup \{s_i\})$                  ▷ Compute error
5:     if $E_i > E_{i-1}$ then
6:         break
7:     else
8:         $S \leftarrow S \cup \{s_i\}$
9: return S
```

Algorithm 7 Greedy backward selection

```
1: $S = V$, $E_{d+1} = \infty$
2: for $t = d : 1 : -1$ do
3:     $s_i = \text{argmin}_{j \in S} \hat{L}(S \setminus \{j\})$     ▷ Find best element to remove
4:     $E_i = \hat{L}(S \setminus \{s_i\})$                ▷ Compute error
5:     if $E_i > E_{i+1}$ then
6:         break
7:     else
8:         $S \leftarrow S \setminus \{s_i\}$
9: return S
```

3.2.2 Linear models

We want to solve the learning and feature selection problem simultaneously via a single optimization.

Sparse regression The key idea is to replace feature selection with setting unimportant features to 0, i.e. working with sparse feature representations, encoded within the pseudo-norm $\|\mathbf{w}\|_0$ = number of non-zero entries in $\mathbf{w}$. The 0-norm penalty encourages coefficients to be exactly 0 and therefore automatic feature selection, however, the optimization problem is hard to solve. We instead use the 1-norm $\|\cdot\|_1$ for optimization to keep the convex optimization problem.

L1-regularized regression problem (Lasso)

$$\mathbf{w}^* = \underset{w}{\text{argmin}} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_1$$

L1-SVM optimization problem

$$\mathbf{w}^* = \underset{w}{\text{argmin}} \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i) + \lambda \|\mathbf{w}\|_1$$

Comparison FW/BW applies to any prediction method but takes time. L1-Regularization is faster (training and feature selection happen jointly) but only works for linear models.

3.3 Non-linear Classification

Theorem 3.4 (Representer Theorem) *The normal to the optimal hyperplane lives in the span of the data $\hat{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$.*

Reformulation using inner-products $\mathbf{x}_i^T \mathbf{x}_j$

$$\hat{R}(\alpha) = \min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, -\underbrace{\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j}_{=y_i w^T x_i}\}$$

3.3.1 Kernels

Kernel-trick Express problem such that it only depends on inner products and replace these inner products with kernels $\mathbf{x}_i^T \mathbf{x}_j \to k(\mathbf{x}_i, \mathbf{x}_j)$.

Definition 3.5 (Kernel) *A kernel is a function $k : X \times X \to \mathbb{R}$ satisfying symmetry and positive semi-definiteness, i.e. for any n, any set $S = \{x_1, \dots, x_n\} \subseteq X$, the kernel (Gram) matrix*

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

is positive semi-definite.

Theorem 3.6 (Kernel Composition) *Let $k_1 : \chi \times \chi \to \mathbb{R}$, $k_2 : \chi \times \chi \to \mathbb{R}$ be defined on data space χ. Then the following are valid kernels*

- $k(x, x') = k_1(x, x') + k_2(x, x')$
- $k(x, x') = k_1(x, x')k_2(x, x')$
- $k(x, x') = ck_1(x, x')$ *for* $c > 0$
- $k(x, x') = f(k_1(x, x'))$, *where f is a polynomial with positive coefficients or the exponential function.*

Theorem 3.7 (Mercer's Theorem) *Let X be a compact subset of $\mathbb{R}^n$ and $k : X \times X \to \mathbb{R}^n$ a kernel function. Then one can expand k in a uniformly convergent series of bounded functions ϕ such that*

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(x').$$

Kernels in $\mathbb{R}^d$

- **Linear Kernel**: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- **Polynomial Kernel**: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$ implicitly represents all monomials of up to degree m. In d-D, there are $\binom{d+m}{m}$ such monomials.
- **Gaussian/ RBF Kernel**: $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2 / h^2)$ maps to infinite dimensional space.
- **Laplacian Kernel**: $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|_1 / h)$
- **? Kernel**: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{M} \mathbf{x}'$ for symmetric positive definite matrix $\mathbf{M}$.
- **ANOVA Kernel**: $k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} k_i(\mathbf{x}^{(i)}, \mathbf{x}'^{(i)})$ with $k_i(x, x') = \exp(-(x - x')^2 / h_i^2)$.

3.3.2 k-Perceptron

Kernelized perceptron optimization problem

$$\underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^{n} \max\{0, -y_i \alpha^T \mathbf{k}_i\} = \min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

for $\mathbf{k}_i = (y_1 k(\mathbf{x}_i, \mathbf{x}_1), \dots, y_n k(\mathbf{x}_i, \mathbf{x}_n))^T$.

Algorithm 8 Kernelized Perceptron

```
1: $\alpha_1 = \ldots = \alpha_n = 0$
2: for $t = 1, 2, \ldots$ do
3:     Pick $i_t \sim \text{Unif}\{1, \ldots, n\}$                ▷ Random data points
4:     $\hat{y} = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_{i_t})\right)$      ▷ Predict
5:     if $y_{i_t} \neq \hat{y}$ then                             ▷ Perceptron gradient
6:         $\alpha_i \leftarrow \alpha_i + \eta_t$
7: $\hat{y} = \text{sign}(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}))$              ▷ Prediction for x
```

3.3.3 k nearest Neighbors (k-NN)

k-nearest Neighbor Label depending on the k nearest neighbor of all data points

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i \{\mathbf{x}_i \text{ among } k \text{ nearest neighbors of } \mathbf{x}\}\right).$$

Choose k using cross validation.

Comparison of Perceptron and k-NN For k-NN, no training is necessary, but it depends on all data, which may render it inefficient. The kernelized perceptron may have improved performance due to optimized weights, can capture "global trends" with suitable kernels, and it depends on wrongly classified examples only, but training requires optimization.

Parametric vs nonparametric Models Parametric models have finite set of parameter (e.g. linear regression, linear perceptron), nonparametric models grow in complexity with the size of the data (e.g. kernelized perceptron, k-NN) and are thus potentially much more expressive and computationally expensive. Kernels provide a principled way of deriving non-parametric models from parametric ones.

3.3.4 Kernelized SVM

Kernelized SVM optimization problem

$$\underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^{n} \max(0, 1 - y_i \alpha^T \mathbf{k}_i) + \lambda \alpha^T \mathbf{D}_y \mathbf{K} \mathbf{D}_y \alpha$$

for $\mathbf{k}_i = (y_1 k(\mathbf{x}_i, \mathbf{x}_1), \dots, y_n k(\mathbf{x}_i, \mathbf{x}_n))^T$ and $\mathbf{D}_y = \text{diag}(y_1, \dots, y_n)$.

3.3.5 Kernelized Regression

Kernelized Linear Regression optimization problem

$$\underset{\alpha}{\operatorname{argmin}} \|\alpha^T \mathbf{K} - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

with closed form solution $\alpha^* = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$ and predictor $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$.

3.4 Class Imbalance

		True label		
		Positive	Negative	$\sum =$
Predicted	Positive	TP	FP	p_+
	Negative	FN	TN	p_-
$\sum =$		n_+	n_-	

		T		
		1	0	
P	1	TP	FP	p_+
	0	FN	TN	p_-
		n_+	n_-	

Metrics to measure goodness of fit

$$\frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{n} \quad \text{(Accuracy)}$$

$$\frac{TP}{TP + FP} = \frac{TP}{p_+} \in [0, 1] \quad \text{(Precision)}$$

$$\frac{TP}{TP + FN} = \frac{TP}{n_+} \in [0, 1] \quad \text{(Recall (TPR))}$$

$$\frac{FP}{TN + FP} = \frac{FP}{n_-} \in [0, 1] \quad \text{(False positive rate (FPR))}$$

$$\frac{2TP}{2TP + FP + FN} = \frac{2}{\frac{TP+FP}{TP} + \frac{TP+FN}{TP}} \quad \text{(F1 Score)}$$

Accuracy is often not meaningful for imbalanced data set, because it may prefer certain mistakes over others (trade false positives and false negatives). Minority class instances contribute little to the empirical risk and may be ignored during optimization.

Upsampling Repeat data points from minority class (possibly with small random perturbations) to obtain balanced data set. This method makes us of all data, but is slower and adding perturbations requires arbitrary choices.

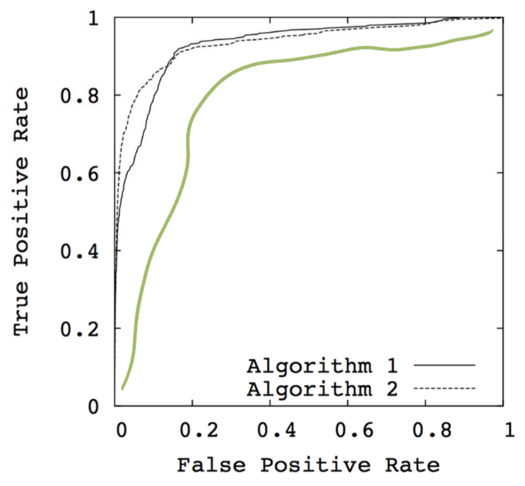
Downsampling Remove training examples from the majority class (e.g. uniformly at random) such that the resulting data set is balanced. This method is faster, because it reduces the test set size, but available data is wasted and information about the majority class.

Cost sensitive classification Modify Perceptron/ SVM by using a cost-sensitive loss function $l_{CS}(\mathbf{w}; \mathbf{x}, y) = c_y l(\mathbf{w}; \mathbf{x}, y)$ to take class imbalance into account.

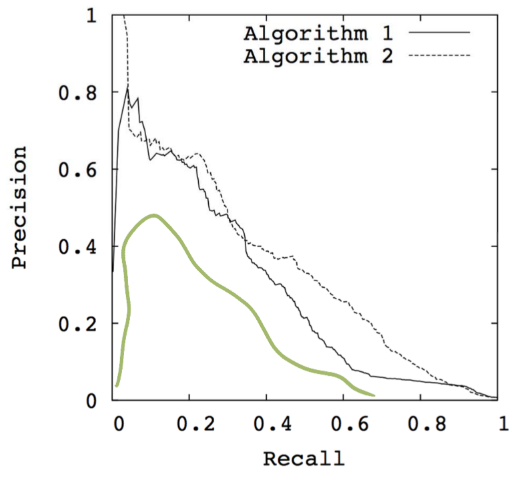
$$l_{CS-P}(\mathbf{w};\mathbf{x},y) = c_y \max(0, -y\mathbf{w}^T\mathbf{x}) \qquad \text{(Perceptron)}$$
$$l_{CS-H}(\mathbf{w};\mathbf{x},y) = c_y \max(0, 1 - y\mathbf{w}^T\mathbf{x}) \qquad \text{(SVM)}$$

with parameters $c_+, c_- > 0$.

Receiver operator characteristic (ROC) curve draws true positive rate vs. false positive rate.



Area under the curve of ROC or Precision Recall as ability of a classifier to provide imbalanced classification.



Theorem 3.8 *Algorithm 1 dominated algorithm 2 in terms of ROC curve if and only if algorithm 1 dominates algorithm 2 in terms of precision recall curves.*

3.5 Multi-class Problems

One-vs-All Solve c binary classifiers for each class. Classify using the classifier with the largest confidence, i.e. predict

$$\hat{y} = \underset{i\in\{1,\ldots,c\}}{\operatorname{argmax}}\, {w^{(i)}}^T x.$$

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

One-vs-One Train $c\frac{c-1}{2}$ binary classifiers, one for each pair of classes i, j. Apply voting scheme, class with highest number of positive prediction wins.

Discussion One-vs-One One-vs-One does not rely on confidence, but is slower than One-vs-All.

Multi-class methods Maintain c weight vectors $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(c)}$, one for each class, and predict $\hat{y} = \operatorname{argmax}_i {\mathbf{w}^{(i)}}^T\mathbf{x}$. Given each data point $(\mathbf{x}, y)$, we want to archive that

$$\mathbf{w}^{(y)}\mathbf{x} > \max_{i\neq y} \mathbf{w}^{(i)T}\mathbf{x} + 1. \qquad (*)$$

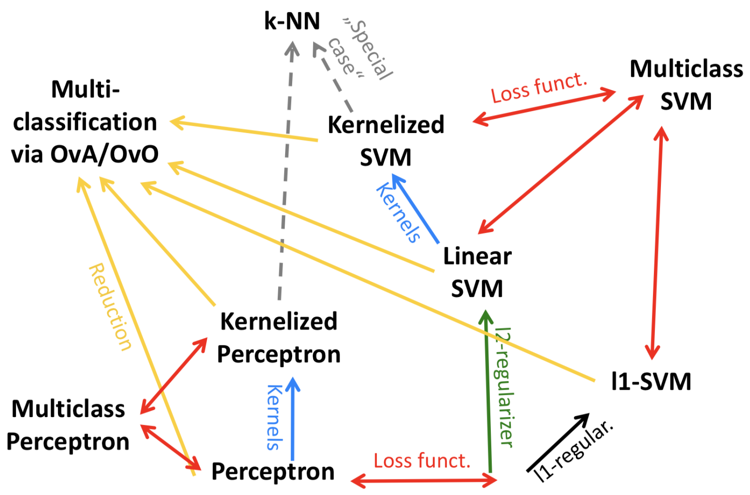
Multi-class hinge-loss

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

$$\nabla_{\mathbf{w}^{(j)}} l_{MC-H}(\mathbf{w}^{(1:c)};\, \mathbf{x},y) = \begin{cases} 0 & (*) \text{ or } j \notin \{y,\hat{y}\} \\ -\mathbf{x} & \text{not } (*) \text{ and } j = y \\ \mathbf{x} & \text{not } (*) \text{ and } j = \hat{y} \end{cases}$$

Confusion Matrices are often used to evaluate multi-class classifiers.

		True label		
		Cat	Dog	Elefant
Predicted	Cat	5	2	0
	Dog	3	7	0
	Elefant	1	0	6



4 Neural Networks

What are good features?

Neural Networks optimization problem

$$\mathbf{w}^* = \underset{\mathbf{w},\theta}{\operatorname{argmin}} \sum_{i=1}^{n} l(y_i; \sum_{j=1}^{m} w_j \phi(x_i, \theta_j))$$

Feature maps, activation function For example $\phi(x,\theta) = \varphi(\theta^T x)$

Activation functions

- **Sigmoid** $\varphi(z) = \frac{1}{1+\exp(-z)}$
- **Tanh** $\varphi(z) = \tanh z$
- **ReLU** $\varphi(z) = \max(0, z)$

Artificial Neural Networks (ANN) are functions of the form

$$f(\mathbf{x}; w, \theta) = \sum_{j=1}^{m} w_j \varphi(\theta_j^T \mathbf{x}).$$

Algorithm 9 Forward Propagation

```
1: $\mathbf{v}^{(0)} = \mathbf{x}$                                   ▷ Input layer
2: for each hidden layer $l = 1 : L-1$ do
3:     $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{v}^{(l-1)}$
4:     $\mathbf{v}^{(l)} = \varphi(\mathbf{z}^{(l)})$
5: $\mathbf{f} = \mathbf{W}^{(L)}\mathbf{v}^{(L-1)}$
6: $\mathbf{y} = \mathbf{f}$                                        ▷ Prediction for regression, or
7: $y = \text{sign}(\mathbf{f})$                               ▷ Prediction for classification, or
8: $y = \text{argmax}_j \, \mathbf{f}_j$                       ▷ Prediction for multiclass classification
```

Theorem 4.1 *Let σ be any continuous sigmoidal function. Then finite sums of the form*

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

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

$$|G(x) - f(x)| < \varepsilon \quad \forall x \in I_n.$$

Aber solche Funktionen kriegen wir doch nicht aus neuronalen Netzwerken, oder?

4.1 Training: Momentum SGD, Backpropagration

Training Given data set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, we want to optimize weights $\mathbf{W} = (\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)})$ using any loss function $l(\mathbf{W}; \mathbf{y}, \mathbf{x})$ (Perceptron loss, multi-class hinge loss, squared loss, etc.)

$$\mathbf{W}^* = \underset{\mathbf{W}}{\text{argmin}} \sum_{i=1}^{n} l(\mathbf{W}; y_i, \mathbf{x}_i).$$

When predicting multiple outputs at the same time, usually define loss as sum per-output losses

$$l(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \sum_{i=1}^{p} l_i(\mathbf{W}; \mathbf{y}_i, \mathbf{x}).$$

This optimization problem is not convex.

Algorithm 10 SGD for ANNs

```
1: Initialize weights $\mathbf{W}$
2: for $t = 1, 2, \ldots$ do
3:     Pick data point $(x, y) \in D$ uniformly at random
4:     Take step in negative gradient direction
5:     $\mathbf{W} = \mathbf{W} - \eta_t \nabla_{\mathbf{W}} l(\mathbf{W}; y, x)$
```

Computing the gradient To compute $\nabla_{\mathbf{W}} l(\mathbf{W}; \mathbf{y}, \mathbf{x})$, we use backpropagation exploiting the chain-rule and the weight-specific gradients $\nabla_{w_{i,j}} l(\mathbf{W}; \mathbf{y}, \mathbf{x})$.

Algorithm 11 Backpropagation

```
1: for the output layer do
2:     $\delta^{(L)} = \text{D}l(\mathbf{f}) = (l'_1(f_1), \ldots, l'_p(f_p))$   ▷ compute "error" True?
3:     $\nabla_{\mathbf{W}^{(L)}} l(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \delta^{(L)} \mathbf{v}^{(L-1)T}$      ▷ Compute gradient matrix
4: for each hidden Layer $l = L-1 : -1 : 1$ do
5:     $\delta^{(l)} = \varphi'(\mathbf{z}^{(l)}) \odot (\mathbf{W}^{(l+1)T}\delta^{(l+1)})$            ▷ compute "error"
6:     $\nabla_{\mathbf{W}^{(l)}} l(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \delta^{(l)} \mathbf{v}^{(l-1)T}$              ▷ Compute gradient
```

Learning rate often initially chosen with a fixed (small) learning rate and decreased slowly after some iterations, e.g. $\eta_t = \min(0.1, 100/t)$. It is also possible to monitor the ratio of weight change (gradient) to weight magnitude. If the ratio is too small, increase learning rate, otherwise decrease learning rate.

Learning with momentum can help to escape local minima by not only moving into direction of gradient, but also in direction of last weight update

$$a = m \cdot a + \eta_t \nabla_{\mathbf{W}} l(\mathbf{W}; \mathbf{y}, \mathbf{x}),$$
$$\mathbf{W} = \mathbf{W} - a,$$

where m denotes a parameter of friction. This method can help to prevent oscillations.

Weight-space symmetries cause 'degenerate' local minima, i.e. multiple local-minima can be equivalent in terms of input-output mapping.

Derivatives of activation functions

- $\varphi'(z) = (\frac{1}{1+e^{-z}})' = \frac{e^z}{(1+e^z)^2} = (1 - \varphi(z))\varphi(z)$. Properties: Differentiable and non-zero everywhere, but $\varphi'(z) \approx 0$ almost everywhere except for $z \approx 0$.
- $\varphi'(z) = (\max(0, z))' = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \end{cases}$. Properties: not differentiable at 0 (in practice just set to 0, doesn't really matter), efficient and > 0 in $\mathbb{R}_+$.

wie bringen wir dieses concept am besten unter? Wie oder wo?

4.2 Initialization and Termination

Initialization of weights Matters, because problem is non-convex. Random initialization usually works well, e.g. $w_{i,j} \sim \mathcal{N}(0, 0.1)$, $w_{i,j} \sim \mathcal{N}(0, 1/\sqrt{|Layer_{l+1}|})$. However, incorrect initialization can lead to bad results. Might want to repeat training multiple times to avoid getting stuck in a poor local optimum. Less deep architectures are more prone to get stuck in a local optimum.

Termination

4.3 Choosing parameters

In principle, one could use cross validation to compare models, however, training ANNs is usually expensive.

Choosing parameters like number of units/ layers/ activation functions/ learning rate/ ...

Type of activation function Sigmoid and tanh are differentiable and were popular in the past. ReLUs are currently used extensively. They are not differentiable (not a problem), fast to compute and their gradients do not vanish (important).

Number of hidden layers [1] In most tasks, one hidden layer is sufficient. More generally:

- 0: only capable of representing linear separable functions or decisions.
- 1: Can approximate any function that contains a continuous mapping from one finite space to another.
- 2: Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

Wo genau ist der Unterschied zwischen 1 und 2?

Number of hidden units [1] The optimal size of the hidden layer is usually between the size of the input and size of the output layers. Some rules of thumb are

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

An upper bound for the number of hidden units is given by

$$N_{\text{hidden units}} \leq \frac{N_{\text{sample}}}{\alpha(N_{\text{input}} + N_{\text{output}})}, \quad 2 \leq \alpha \leq 10.$$

Woher kommt das?

Regularization method

Learning rate schedule

Weight initialization

Number of convolution/ pooling layers

4.4 Regularization

Neural networks are prone to overfitting due to their large number of parameters.

Early stopping doesn't let the neural net converge. Monitor prediction performance on validation set and stop training once validation error starts to increase.

Regularization adds the usual regularization term to the optimization problem

$$\mathbf{W}^* = \operatorname*{argmin}_{\mathbf{W}} \sum_{i=1}^{n} l(\mathbf{W}; y_i, \mathbf{x}_i) + \lambda \|\mathbf{W}\|_2^2.$$

Dropout regularization Randomly ignore hidden units during each iteration of SGD with probability p. After the training, half the weights to compensate.

4.5 Invariances

Predictions should be unchanged under some transformations of the data, e.g. translation, rotation, scale, pitch, speed, etc. Invariances can be learned from data: SIFT (scale invariant feature transformation), Ceptum (speech recognition).

Encourage learning of invariances e.g. by

- augmentation of the training set
- special regularization terms
- invariance built into pre-processing
- implement invariance into structure of ANN (e.g. CNN)

Hat er dazu mehr gesagt?

4.6 Convolutional Neural Networks (CNN)

CNNs are ANNs for specialized applications like image recognition. They are robust against transformations (translation, rotations, scaling). The hidden layer(s) closest to the input layer shares parameters: each hidden unit only depends on all "closeby" inputs (e.g. pixels), and weights constrained to be identical across all units on the layer. This reduces number of parameters and enourages robustness against (small amounts of) translation. The weights can still be optimized via backprobagation.

Output dimension of CNNs when applying m different $f \times f$ filters to an $n \times n$ image with padding p and stride s is given by

$$L = \frac{n + 2p - f}{s} + 1.$$

Pooling aggregates several units to decrease the with of the network and hence the number of parameters. Usually, either average or maximum values are considered.

4.7 ANNs vs. tanh-kernels

Class of functions that can be modeled with ANNs or tanh-kernels are the same. This does not mean the trained models are the same. Kernel optimization problems are linear and hence convex, whereas the ANN optimization problem optimizes θ and $\mathbf{w}$, which makes it non-convex. Robustness for kernels that ANNs do not exhibit . However, ANNs have more degrees of freedom. Noisy data better with kernels because of their robustness.

II Unsupervised Learning

Learning without labels. Typically used for exploratory data analysis, e.g. finding patterns, visualizations.

5 Clustering: k-means

Unsupervised analog to classification.

Clustering Given data points, group them into clusters such that similar points are in the same cluster and dissimilar points are in different clusters. Points are typically represented either in (high-dimensional) Euclidean space or with distances specified by a metric or kernel. Clustering is related to anomaly/ outlier detection.

Standard approaches to clustering

- **Hierarchical clustering**: Build a tree (bottom-up or top-down), representing distances among data points. Examples include single-, average-linkage clustering.
- **Partitional approaches**: Define and optimize a notion of "cost" defined over partitions. Examples include spectral clustering, and graph-cut based approaches.
- **Model based approaches**: Maintain cluster "models" and infer cluster membership (e.g. assign each point to closest center). Examples include k-means, and Gaussian mixture models.

k-means optimization problem assumes points are in Euclidean space $\mathbf{x}_i \in \mathbb{R}^d$, represents clusters as centers $\mu_j \in \mathbb{R}^d$, and each point is assigned to its closest center (Voronoi partition). The goal is to minimize the average squared distance, i.e.

$$\hat{R}(\mu) = \hat{R}(\mu_1, \dots, \mu_k) = \sum_{i=1}^{n} d(\mathbf{x}_i, \mu) = \sum_{i=1}^{n} \min_{j \in \{1,\dots,k\}} \|\mathbf{x}_i - \mu_j\|_2^2,$$
$$\hat{\mu} = \operatorname*{argmin}_{\mu} \hat{R}(\mu).$$

This optimization problem is non-convex and NP-hard.

Algorithm 12 k-means algorithm (Lloyd's heuristic)

```
1: $\mu^{(0)} = \{\mu_1^{(0)}, \dots, \mu_k^{(0)}\}$                ▷ Initialize cluster centers
2: while not converged, $t = 1$, $t = t + 1$ do
3:     $z_i = \operatorname{argmin}_{j \in \{1,\dots,k\}} \|\mathbf{x}_i - \mu_j^{(t-1)}\|_2^2$
4:                                        ▷ Assign each point $\mathbf{x}_i$ to closest center
5:     $\mu_j^{(t)} = \frac{1}{n_j} \sum_{i: z_i = j} \mathbf{x}_i$
6:                              ▷ update center as mean of assigned data points
```

Properties and challenges of k-means Guaranteed to monotonically decrease average squared distance in each iteration. It converges to a local minimum. Complexity per iteration is $\mathcal{O}(nkd)$.

Initializing k-means different approaches: multiple random restarts, farthest points heuristic (often works well, but prone to outliers), seeding with k-means++.

Algorithm 13 k-means++ algorithm for initializing centers

```
1: $i_1 \sum \text{Unif}\{1, \dots, n\}$
2: $\mu_1 = x_{i_1}$
3: for $j = 2 : k$ do
4:     $i_j = i$ with probability $\frac{d(x_i, \mu_{1:j-1})^2}{\sum_{i=1}^{n} d(x_i, \mu_{1:j-1})^2} = \frac{\min_{1 \le l \le j-1} \|x_i - \mu_l\|_2}{\sum_{i=1}^{n} d(x_i, \mu_{1:j-1})^2}$
5:     $\mu_j = x_{i_j}$
6: Continue with standard k-means algorithm
```

Model selection (i.e. determining the number of clusters) is difficult. Approaches include heuristic quality measures, regularization (favor "simple" models with few parameters by penalizing complex models), information theoretic basis (tradeoff between robustness (stability) and informativeness). One heuristic for determining k is to pick k such that increasing k leads to negligible decrease in loss.

Challenges with k-means Generally, the algorithm only converges to local minimum (dependence on initialization), the number of iterations needed may be exponential (however, practically not often), determining the number of clusters k is difficult, and models of arbitrary shape cannot be modeled well.

Don't do crossvalidation, because there is a strong correlation between train accuracy and test/validation accuracy. Explain, please.

Nonlinear k-means Applying k-means on kernel-principal components is sometimes called **Kernel-k-means** or **Spectral Clustering**.

6 Dimension Reduction

Unsupervised analog to regression. Given data set $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, obtain "embedding" (low-dimensional representation) $\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathbb{R}^k$.

Typical approaches Assume $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$, obtain mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ such that $k \ll d$. One can distinguish linear dimension reduction $f(\mathbf{x}) = \mathbf{A}\mathbf{x}$ and nonlinear dimension reduction, parametric and non-parametric.

6.1 Linear Dimension Reduction: PCA

PCA optimization problem

$$(\mathbf{W}^*, \mathbf{z}_1^*, \dots, \mathbf{z}_n^*) = \operatorname{argmin} \sum_{i=1}^{n} \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2,$$

such that $\mathbf{W} \in \mathbb{R}^{d \times k}$ is orthogonal and $\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathbb{R}^k$.

Theorem 6.1 (PCA) *Let $\Sigma = \frac{1}{n}\sum_{i=1}^n x_i x_i^T = \sum_{i=1}^d \lambda_i v_i v_i^T$, $\lambda_1 \geq \dots \geq \lambda_d \geq 0$ be the empirical covariance. Assume that $\mu = \frac{1}{n}\sum_i x_i = 0$. The linear dimension reduction optimization problem is equivalent to*

$$W^* = \operatorname{argmax} W^T \Sigma W, \quad z_i^* = (W^*)^T x_i,$$

and its solution is given by

$$W^* = (v_1 | \dots | v_k), \quad z_i^* = (W^*)^T x_i.$$

Proof?

PCA via SVD From SVD it follows that $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{U} \in \mathbb{R}^{n \times n}$ and $\mathbf{V} \in \mathbb{R}^{d \times d}$ orthogonal, $S \in \mathbb{R}^{n \times d}$ diagonal. The top k principal components are the first k columns of $\mathbf{V}$ and $\Sigma = \mathbf{V}\mathbf{S}^T\mathbf{S}\mathbf{V}^T$

What?

6.2 Nonlinear Dimension Reduction

6.2.1 Kernel PCA

Theorem 6.2 *For w^* there exist α_i such that $w^* = \sum_{i=1}^n \alpha_i x_i$.*

1D Kernel PCA The 1D Kernel-PCA problem requires solving

$$\alpha^* = \operatorname*{argmax}_{\alpha^T \mathbf{K} \alpha = 1} \alpha^T \mathbf{K}^T \mathbf{K} \alpha$$

with closed-form solution from the eigendecomposition of $\mathbf{K} = \sum_{i=1}^n \lambda_1 \mathbf{v}_i \mathbf{v}_i^T$, $\lambda_1 \geq \dots \geq \lambda_d \geq 0$

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

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

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

is obtained from $\mathbf{K} = \sum_{i=1}^n \lambda_i \mathbf{v}_i \mathbf{v}_i^T$, $\lambda_1 \geq \dots \lambda_d \geq 0$. Given this, a new point $\mathbf{x}$ is projected as $\mathbf{z} \in \mathbb{R}^k$

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

Notes on PCA

- Complexity grows with the number of data points.
- Cannot easily "explicitly" embed high-dimensional data (unless we have an appropriate kernel).
- Kernel-PCA corresponds to applying PCA in the feature space induced by the kernel k.
- Can be used to dissolve non-linear feature maps in closed form. This can be used as inputs, e.g. SVMs given "multilayer SVMs" was heisst das?
- May want to center the kernel $\mathbf{E} = \frac{1}{n}[1, \dots, 1][1, \dots, 1]^T$, $\mathbf{K}' = \mathbf{K} - \mathbf{K}\mathbf{E} - \mathbf{E}\mathbf{K} + \mathbf{E}\mathbf{K}\mathbf{E}$ ... und das?.

6.2.2 Autoencoders

Idea is to learn the identity function $\mathbf{x} \approx f(\mathbf{x}; \theta) = f_2(f_1(\mathbf{x}; \theta_1); \theta_2)$, where $f_1 : \mathbb{R}^d \to \mathbb{R}^k$ is the encoder and $f_2 : \mathbb{R}^k \to \mathbb{R}^d$ is the decoder.

Neural network autoencoders are ANNs where there is one output unit for each of the d input units and the number k of hidden units is usually smaller than the number of inputs (compression effect). The goal is to optimize the weights such that the output agrees with the input, for example by minimizing the square loss.

Training autoencoders For example, minimize the square loss

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

using SGD (backpropagation). Initialization matters and is challenging, c.f. work on pretraining, e.g. layerwise training of restricted Boltzmann machines.

6.2.3 Other

There are other nonlinear methods like locally linear embedding (LLE) or multi-dimensional scaling.

6.3 Autoencoders vs. PCA

If the activation function is the identity $\phi(z) = z$, then fitting a NN autoencoder is equivalent to PCA.

6.4 PCA vs. k-Means

After reformulating the optimization problems, both differ mostly in constraints.

PCA Problem

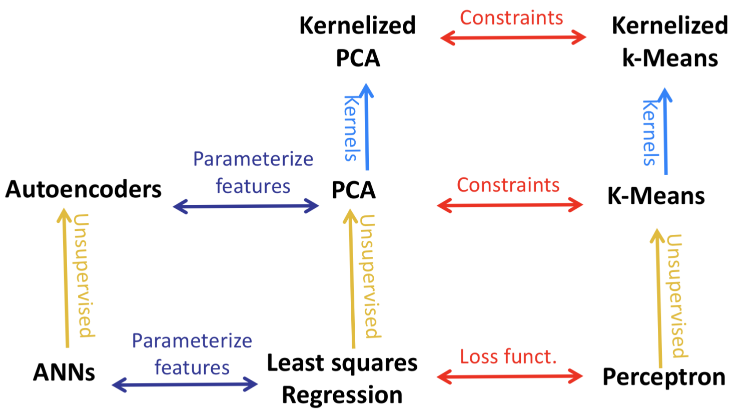
$$(\mathbf{W}, \mathbf{z}_1, \dots, \mathbf{z}_n) = \operatorname{argmin} \sum_{i=1}^n \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is orthogonal, $\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathbb{R}^k$.

k-Means Problem

$$(\mathbf{W}, \mathbf{z}_1, \dots, \mathbf{z}_n) = \operatorname{argmin} \sum_{i=1}^n \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ arbitrary, $\mathbf{z}_1, \dots, \mathbf{z}_n \in E_k$ and $E_k = \{(1, 0, \dots, 0)^T, \dots, (0,$
is the set of unit vectors in $\mathbb{R}^k$.



III Probabilistic modeling

General approach to probabilistic modeling

(i) Start with statistical assumption on data, mostly data points modeled as i.i.d. (can be relaxed).

(ii) Choose likelihood function (e.g. Gaussian, student-t, logistic, exponential). This defines the loss function.

(iii) Choose a prior (e.g. Gaussian, Laplace, exponential). This defines the regularizer.

(iv) Optimize for MAP parameters.

(v) Choose hyperparameters (i.e., variance) through cross-validation.

(vi) Make predictions via Bayesian Decision Theory.

7 Probabilistic Modeling, Bias-variance tradeoff

We would like to take a statistical perspective on supervised learning and statistically model the data, e.g. quantify uncertainty or express prior knowledge/ assumptions about the data. Many of the previous approaches from part 1 can be interpreted as fitting probabilistic models.

Fundamental assumption: i.i.d Our data is independently and identically distributed, i.e. $(\mathbf{x}_i, y_i) \sim P(\mathbf{X}, Y)$.

Problem formulation We would like to identify a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the prediction error (risk)

$$R(h) = \int \mathbb{P}(\mathbf{x}, y) l(y; h(\mathbf{x}))\, \mathrm{d}\mathbf{x}\, \mathrm{d}y = \mathbb{E}_{\mathbf{x},y}[l(y; h(\mathbf{x}))]$$

defined in terms of a loss function.

Risk In least squares regression, the risk is $R(h) = \mathbb{E}_{\mathbf{x},y}[(y - h(\mathbf{x}))^2]$

7.1 Parametric Estimation

Bayes' optimal predictor for the squared loss assuming we knew $\mathbb{P}(\mathbf{X}, Y)$. Minimizing least squares risk leads to the hypothesis h^* given by the conditional mean

$$h^*(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]. \qquad \textbf{(Bayes optimal predictor)}$$

Estimating conditional distributions We know which h^* minimizes the risk, thus one strategy for estimating a predictor from training data is to estimate the conditional distribution

$$\hat{\mathbb{P}}(Y|\mathbf{X}) \qquad \textbf{(Conditional distribution)}$$

and then for a test point $\mathbf{x}$ to predict the label

$$\hat{y} = \hat{\mathbb{E}}[Y|\mathbf{X} = \mathbf{x}] = \int y \hat{\mathbb{P}}(y|\mathbf{X} = \mathbf{x})\, \mathrm{d}y.$$

Parametric Estimation $\hat{\mathbb{P}}(Y|\mathbf{X}, \theta)$ is a common approach. Choose a particular parametric form

$$\hat{\mathbb{P}}(Y|\mathbf{X}, \theta) \qquad \textbf{(Parametric cond. distribution)}$$

with parameters θ, then optimize the parameters. This is done for example by Maximum Likelihood estimation.

7.2 Least Squares Regression = Gaussian Maximum Likelihood Estimation (MLE)

Maximum (conditional) Likelihood Estimation is a method for optimizing the parameters, i.e.

$$\theta^* = \operatorname*{argmax}_\theta \hat{\mathbb{P}}(y_1, \dots, y_n | \mathbf{x}_i, \dots, \mathbf{x}_n, \theta) \overset{i.i.d}{=} \operatorname*{argmax}_\theta \prod_{i=1}^n \hat{\mathbb{P}}(y_i|\mathbf{x}_1, \theta) \iff$$

$$\theta^* = \operatorname*{argmax}_\theta \log \hat{\mathbb{P}}(y_{i:n}|\mathbf{x}_{i:n}, \theta) = \operatorname*{argmax}_\theta \sum_{i=1}^n \log \hat{\mathbb{P}}(y_i|\mathbf{x}_i, \theta)$$

MLE = Least squares With the assumption of Gaussian noise, i.e. $y_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \sigma^2) \equiv y_i = \mathbf{w}^T\mathbf{x}_i + \varepsilon_i,\ \varepsilon_i \sim \mathcal{N}(0, \sigma^2)$, maximizing the likelihood is equivalent to least squares estimation

$$\operatorname*{argmax}_{\mathbf{w}} \mathbb{P}(y_1, \dots, y_n|\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^n (y_i - \mathbf{w}^T\mathbf{x}_i)^2.$$

MLE for i.i.d. Gaussian noise Suppose $\mathcal{H} = \{h : \mathcal{X} \to \mathbb{R}\}$ is a class of functions. Assuming that $\mathbb{P}(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y|h^*(\mathbf{x}), \sigma^2)$ was heisst das argument für den mean, also $y|h^*(\mathbf{x})$? for some function $h^* : \mathcal{X} \to \mathbb{R}$ and some $\sigma^2 > 0$ the MLE for data $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ in $\mathcal{H}$ is given by

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{i=1}^n (y_i - h(\mathbf{x}_i))^2.$$

MLE properties for $n \to \infty$ (for finite n we must still avoid overfitting)

- **consistency**, parameter estimate converges to true parameters in probability,
- **asymptotic efficiency**, smallest variance among all "well-behaved" estimators for large n,
- **asymptotic normality**.

7.3 Bias Variance Tradeoff

Prediction error descomposition states that prediction error = $\mathbb{E}_D\mathbb{E}_{\mathbf{X},Y}[($ $\hat{h}_D(\mathbf{X}))^2]$ = bias^2 + variance + noise, or mathematically

$$\mathbb{E}_D\mathbb{E}_{\mathbf{X},Y}[(Y - \hat{h}_D(\mathbf{X}))^2] = \mathbb{E}_\mathbf{X}[\mathbb{E}_D\hat{h}_D(\mathbf{X}) - h^*(\mathbf{X})]^2 + \mathbb{E}_\mathbf{X}\mathbb{E}_D[\hat{h}_D(\mathbf{X}) - \mathbb{E}_{D'}\hat{h}_{D'}(\mathbf{X})]^2 + \mathbb{E}_{\mathbf{X},Y}[Y - h^*(\mathbf{X})]^2$$

Bias, Variance, Noise in estimation MLE solution depends on training data $\hat{h} = \hat{h}_D = \operatorname{argmin}_{g \in \mathcal{H}} \sum_{(\mathbf{x},y) \in D}(y - h(\mathbf{x}))^2$, but training data D itself is random, drawn i.i.d from $\mathbb{P}(\mathbf{X}, Y)$. We thus use $\mathbb{E}_D[\hat{h}_D(\mathbf{X})]$.

Bias is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting). Bias is expressed as $\mathbb{E}_\mathbf{X}[\mathbb{E}_D\hat{h}_D(\mathbf{X}) - h^*(\mathbf{X})]^2$

Variance is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data rather than the intended outputs (overfitting). Variance is expressed as $\mathbb{E}_\mathbf{X}\mathbb{V}_D[\hat{h}_D(\mathbf{X})]^2 = \mathbb{E}_\mathbf{X}\mathbb{E}_D[\hat{h}_D(\mathbf{X}) - \mathbb{E}_{D'}\hat{h}_{D'}(\mathbf{X})]^2$

Noise is the risk error incurred by optimal model, i.e. the irreducible error, and constant with respect to the model complexity. Nose is expressed as $\mathbb{E}_{\mathbf{X},Y}[Y - h^*(\mathbf{X})]^2$.

J: ich glaube wir sollten noch mal über diese formalen formulierungen von bias, variance und noise sprechen.

Bias and variance in regression The MLE (= least-squares fit) for linear regression is unbiased (if h^* in class $\mathcal{H}$) and the minimum variance estimator among all unbiased estimators. However, least-squares solutions may be overfit. Thus, trade (a little bit of) bias for a (potentially dramatic) reduction in variance, i.e. regularization (ridge, Lasso, etc.).

7.4 Ridge Regression = Maximum A Posteriori (MAP) Estimation

A posteriori estimate Introduce bias by expressing assumptions on parameters through a Bayesian prior, e.g. $\theta \sim \mathcal{N}(0, \beta^2 Id)$. Then, the posterior distribution of θ using Bayes' rule is given by

$$\mathbb{P}(\theta|\mathbf{x}, y) = \frac{\mathbb{P}(\theta)\mathbb{P}(y|\mathbf{x}, \theta)}{\mathbb{P}(y|\mathbf{x})}. \qquad \textbf{(Posterior)}$$

Maximizing a posteriori estimate parameters θ leads to the ridge regression problem

$$\operatorname*{argmax}_{\theta} \log \mathbb{P}(\theta|\mathbf{x}, y) = -\log \mathbb{P}(\theta) - \log \mathbb{P}(y_{1:n}|x_{1:n}, \theta)$$
$$+ \log \mathbb{P}(y|x) = \operatorname*{argmin}_{\theta} \frac{1}{2\beta^2}\|\theta\|_2^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta^T\mathbf{x}_i)^2.$$

Ridge Regression = MAP Ridge regression can be understood as finding the Maximum A Posteriori (MAP) parameter estimate for a linear regression problem, assuming that the noise $\mathbb{P}(y|\mathbf{x}, \theta)$ is i.i.d. Gaussian and the prior $\mathbb{P}(\theta)$ on the model parameters θ is Gaussian.

Regularization and MAP interference More generally, regularized estimation can often be understood as MAP inference

$$\operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{n} l(\mathbf{w}^T\mathbf{x}_i; \mathbf{x}_i, y_i) + C(\mathbf{w}) = \operatorname*{argmax}_{\mathbf{w}} \prod_i \mathbb{P}(y_i|\mathbf{x}_i, \mathbf{w})\mathbb{P}(\mathbf{w})$$
$$= \operatorname*{argmax}_{\mathbf{w}} \mathbb{P}(\mathbf{w}|D)$$

where $C(\mathbf{w}) = -\log \mathbb{P}(\mathbf{w})$ and $l(\mathbf{w}^T\mathbf{x}_i; \mathbf{x}_i, y_i) = -\log \mathbb{P}(y_i|\mathbf{x}_i, \mathbf{w})$. This perspective allows changing **priors (= regularizers)** and **likelihoods (= loss functions)**.

7.5 Examples for other Priors and Likelihood Functions

Laplace prior = l1-regularization

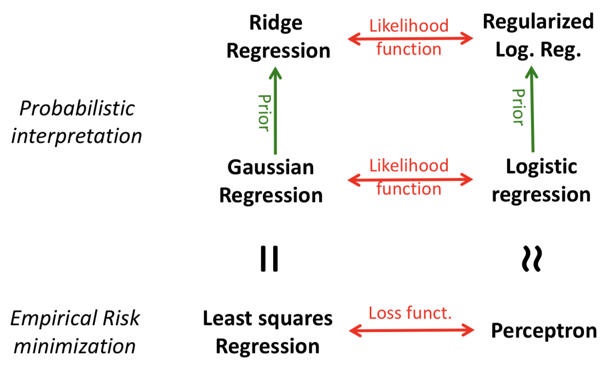
$$\mathbb{P}(x; \mu, b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$$

One can introduce robustness by changing the likelihood (=loss) function.

Student-t likelihood

$$\mathbb{P}(y|\mathbf{x}, \mathbf{w}, \nu, \sigma^2) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu\sigma^2}\Gamma(\frac{\nu}{2})}\left(1 + \frac{(y - \mathbf{w}^T\mathbf{x})^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}}$$

Compared with the Gaussian distribution, outliers are encouraged, because the student-t likelihood decreases algebraically ($\mathbb{P}(|y-\mu| > t) = \mathcal{O}(t^{-\alpha})$), whereas Gaussian likelihood decreases exponentially ($\mathbb{P}(|y-\mu| > t) = \mathcal{O}(e^{-t})$). Thus, student-t likelihood might be better for data with extreme outliers.



8 Classification: Logistic regression

There are no natural statistical models for classification.

Link function for logistic regression

$$\sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}.$$

what is a link function?

Logistic regression replaces the assumption of Gaussian noise (squared loss) by i.i.d. Bernoulli noise

$$\mathbb{P}(y|\mathbf{x}, \mathbf{w}) = \mathrm{Ber}(y; \sigma(\mathbf{w}^T\mathbf{x})).$$

The parameters $\mathbf{w}$ can be estimated via MLE or MAP estimation.

MLE for logistic regression is the convex optimization problem

$$\hat{\mathbf{w}} = \operatorname*{argmax}_{w} \mathbb{P}(y_{i:n}|\mathbf{w}, \mathbf{x}_{1:n}) = \operatorname*{argmax}_{w} \sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)).$$

Logistic loss gradient is given by

$$\nabla_w l(\mathbf{w}) = \frac{-yx}{\exp(y\mathbf{w}^T\mathbf{x}) + 1} = -y\mathbf{x}\hat{\mathbb{P}}(Y = -y|\mathbf{w}, \mathbf{x}),$$

i.e. the gradient is large if model $\mathbf{w}$ is 'surprised' by y.

Algorithm 14 SGD for logistic regression

```
1: Initialize $\mathbf{w}$
2: for $t = 1, 2, \ldots$ do
3:     Pick data point $(\mathbf{w}, y)$ uniformly at random from data D
4:     Compute probability of misclassification with current model
           $\hat{\mathbb{P}}(Y = -y|\mathbf{w}, \mathbf{x}) = \frac{1}{1 + \exp(y\mathbf{w}^T\mathbf{x})}$
5:     Take gradient step
           $\mathbf{w} \leftarrow \mathbf{w} + \eta_t y\mathbf{x}\hat{\mathbb{P}}(Y = -y|\mathbf{w}, \mathbf{x})$
```

Regularizers can be introduced by estimating MAP instead of solving the MLE problem. For the respective priors, we get the following optimization problems

- **L2 (Gaussian)** $\operatorname{argmin}_w \sum_{i=1}^n \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda\|\mathbf{w}\|_2^2$
- **L1 (Laplace)** $\operatorname{argmin}_w \sum_{i=1}^n \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda\|\mathbf{w}\|_1$

Algorithm 15 SGD for l2-regularized logistic regression

```
1: Initialize $\mathbf{w}$
2: for $t = 1, 2, \ldots$ do
3:     Pick data point $(\mathbf{w}, y)$ uniformly at random from data D
4:     Compute probability of misclassification with current model
           $\hat{\mathbb{P}}(Y = -y|\mathbf{w}, \mathbf{x}) = \frac{1}{1 + \exp(y\mathbf{w}^T\mathbf{x})}$
5:     Take gradient step
           $\mathbf{w} \leftarrow \mathbf{w}(1 - 2\lambda\eta_t) + \eta_t y\mathbf{x}\hat{\mathbb{P}}(Y = -y|\mathbf{w}, \mathbf{x})$
```

8.1 Regularized Logistic Regression

Regularized logistic regression Find optimal weights by minimizing logistic loss + regularizer

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{w} \sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda\|\mathbf{w}\|_2^2 \qquad \textbf{(Learning)}$$
$$= \operatorname*{argmax}_{w} \mathbb{P}(\mathbf{w}|\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n)$$

Use conditional distribution

$$\mathbb{P}(y|\mathbf{x}, \hat{\mathbf{w}}) = \frac{1}{1 + \exp(-y\hat{\mathbf{w}}^T\mathbf{x})} \qquad \textbf{(Classification)}$$

to for example predict more likely class.

Generalizations Logistic regression may be kernelized, there exist natural multi-class variants, and one can apply logistic loss function to neural networks in order to have them output probabilities.

8.2 Kernelized Logistic Regression

Kernelized logistic regression Find optimal weights by minimizing logistic loss + regularizer

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \alpha^T \mathbf{K}_i)) + \lambda \alpha^T \mathbf{K} \alpha \qquad \textbf{(Learning)}$$

Use conditional distribution

$$\hat{\mathbb{P}}(y|\mathbf{x}, \hat{\alpha}) = \frac{1}{1 + \exp(-y \sum_{j=1}^{n} \alpha_j k(\mathbf{x}_j, \mathbf{x}))} \qquad \textbf{(Classification)}$$

to for example predict more likely class.

8.3 Multi-class Logistic Regression

Multi-class Maintain one weight vector per class and model

$$\mathbb{P}(Y = y, \mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_c) = \frac{\exp(\mathbf{w}_i^T \mathbf{x})}{\sum_{j=1}^{c} \exp(\mathbf{w}_j^T \mathbf{x})}$$

By setting $\mathbf{w}_c = 0$ we force uniqueness and can recover logistic regression as a special case.

Cross-entropy loss is given by

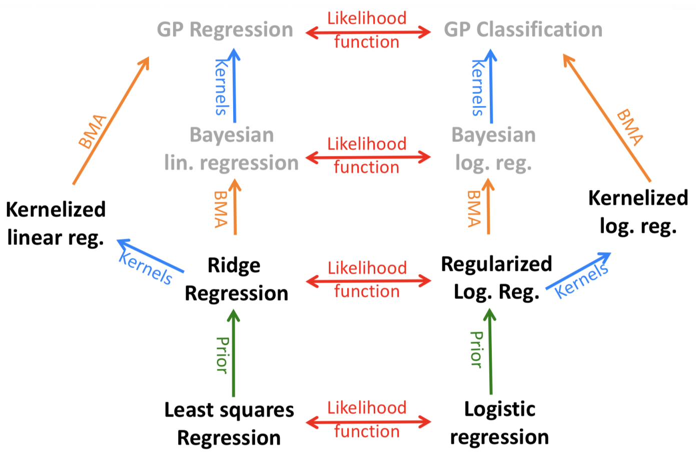
$$l(y; \mathbf{x}; \mathbf{w}_1, \dots, \mathbf{w}_c) = -\log \mathbb{P}(Y = y, \mathbf{x}, \mathbf{w}_1, \dots, \mathbf{w}_c)$$

8.4 Comparison

SVM vs. Logistic regression SVM/ Perceptron sometimes has higher classification accuracy and produces sparse solutions, but cannot easily give class probabilities. Logistic regression can obtain class probabilities, but it produces dense solutions.

Outlook: Bayesian Learning

- **Optimization based learning** such as MAP or MLE, i.e. $\hat{\mathbf{w}} = \operatorname{argmax}_w \mathbb{P}(\mathbf{w}|D)$ given $\mathbb{P}(y|\mathbf{x}, \hat{\mathbf{w}})$ ignores uncertainty in model, but optimization is typically efficient.
- **Integration based learning**/ Bayesian model averaging, i.e. $\mathbb{P}(y|\mathbf{x}) = \int \mathbb{P}(y|\mathbf{x}, \mathbf{w}) \mathbb{P}(\mathbf{w}|D)$, quantifies uncertainty in model, but integration is typically intractable.



9 Bayesian Decision Theory

Idea Given a conditional distribution over labels $\mathbb{P}(y|\mathbf{x})$ with $y \in \mathcal{Y}$, a set of actions $\mathcal{A}$, and 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

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_y(C(y, a)|\mathbf{x}) = \operatorname*{argmin}_{a \in \mathcal{A}} \sum_y \mathbb{P}(y|\mathbf{x}) C(y, a)$$

If we had access to the true distribution $\mathbb{P}(y|\mathbf{x})$ this decision would implement the Bayesian optimal decision. In practice, one can only estimate it, e.g. (logistic) regression.

Logistic Regression Estimated conditional distribution $\hat{\mathbb{P}}(y|\mathbf{x}) = \text{Ber}(y; \sigma(\hat{\mathbf{w}}^T \mathbf{x}))$, action set $\mathcal{A} = \{+1, -1\}$, and cost function $C(y, a) = [y \neq a]$. Then the action that minimizes the expected cost is the most likely class

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_y(C(y, a)|\mathbf{x}) = \operatorname*{argmax}_{y} \hat{\mathbb{P}}(y|\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x}).$$

LS Regression Estimated conditional distribution $\hat{\mathbb{P}}(y|\mathbf{x}) = \mathcal{N}(y; \hat{\mathbf{w}}^T \mathbf{x}, \sigma^2$ action set $\mathcal{A} = \mathbb{R}$, and cost function $C(y, a) = (y - a)^2$. Then the action that minimizes the expected cost is the conditional mean

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_y(C(y, a)|\mathbf{x}) = \mathbb{E}_y(y|\mathbf{x}) = \int y \hat{\mathbb{P}}(y|\mathbf{x})\, dy = \hat{\mathbf{w}}^T \mathbf{x}.$$

9.1 Asymmetric Costs

Asymmetric Costs Estimated conditional distribution $\hat{\mathbb{P}}(y|\mathbf{x}) = \text{Ber}(y; \sigma(\hat{\mathbf{w}}^T \mathbf{x}))$, action set $\mathcal{A} = \{+1, -1\}$, and cost function

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

Then the action that minimizes the expected cost is

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_y(C(y, a)|\mathbf{x}) = \begin{cases} 1 & \mathbb{P}(y = +1|\mathbf{x}) > \frac{c_{FP}}{c_{FP} + c_{FN}} \\ 0 & \text{otherwise} \end{cases}.$$

Doubtful logistic regression Estimated conditional distribution $\hat{\mathbb{P}}(y|\mathbf{x}) = \text{Ber}(y; \sigma(\hat{\mathbf{w}}^T \mathbf{x}))$, action set $\mathcal{A} = \{+1, -1, D\}$, and cost function

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

Then the action that minimizes the expected cost is

$$a^* = \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_y(C(y, a)|\mathbf{x}) = \begin{cases} y & \hat{\mathbb{P}}(y|\mathbf{x}) \geq 1 - c \\ D & \text{otherwise} \end{cases},$$

i.e. pick the most likely class only if confident enough.

Asymmetric cost for regression Estimated conditional distribution $\hat{\mathbb{P}}(y|\mathbf{x}) = \mathcal{N}(y; \hat{\mathbf{w}}^T \mathbf{x}, \sigma^2)$, action set $\mathcal{A} = \mathbb{R}$, and cost function $C(y, a) = c_1 \max(y - a, 0) + c_2 \max(a - y, 0)$. Then the action that minimizes the expected cost is

$$\begin{aligned} a^* &= \operatorname*{argmin}_{a \in \mathcal{A}} \mathbb{E}_y(C(y, a)|\mathbf{x}) = c_1 \mathbb{1}_{[y > a]} + c_2 \mathbb{1}_{[y < a]} \\ &= \hat{\mathbf{w}}^T \mathbf{x} + \sigma \Phi^{-1} \left(\frac{c_1}{c_1 + c_2} \right), \end{aligned}$$

where Φ is the Gaussian CDF.

9.2 Uncertainty Sampling

Outlook: active learning We would like to minimize the number of labels. A simple strategy is to always pick the label we are most uncertain about. Estimate $\hat{\mathbb{P}}(y|\mathbf{x})$ from seen data, compute $p_i = \mathbb{P}(y_i = +1|\mathbf{x}_i)$ for unknown data point $\mathbf{x}_i$. If $p_i \approx 1$ or $p_i \approx 0$, the model is certain, if $p_i \approx 0.5$, the model is uncertain. Define an uncertainty score $U_i = -|p_i - 0.5|$, find the most uncertain data pint $i^* = \operatorname{argmax}_i U_i$ and ask for this label. For linear regression $U_i = |\mathbf{w}^T \mathbf{x}_i|$.

Comments Active learning violates i.i.d. assumption, it can get stuck with bad models, and more advanced selection criteria are available, e.g. query point that reduces uncertainty of other points as much as possible.

Algorithm 16 Uncertainty sampling

```
1: Pool of unlabeled examples $D_x = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$
2: Also maintain an empty data set $D$, initially empty
3: for $t = 1, 2, 3, \dots$ do
4:     Estimate $\hat{\mathbb{P}}(Y_i|\mathbf{x}_i)$ given current data $D$
5:     Pick most uncertain unlabeled example
           $i_t \in \operatorname{argmin}_i |0.5 - \hat{\mathbb{P}}(Y_i|\mathbf{x}_i)|$
6:     Query label $y_{i_t}$
7:     Set $D \leftarrow D \cup \{(\mathbf{x}_i, y_{i_t})\}$
```

10 Generative Modeling

10.1 Discriminative vs. Generative Modeling

Discriminative vs. generative modeling

- **Discriminative models** aim to estimate $\mathbb{P}(y|\mathbf{x})$
- **Generative models** aim to estimate the joint distribution $\mathbb{P}(y, \mathbf{x})$

Discriminative models can be derived from generative models by

$$\mathbb{P}(y|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}, y)}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(\mathbf{x}, y)}{\sum_y \mathbb{P}(x, y)}.$$

Discriminative models do not have access to $\mathbb{P}(\mathbf{x})$ and thus will not be able to detect outliers, i.e. points for which $p(\mathbf{x}_i)$ is small.

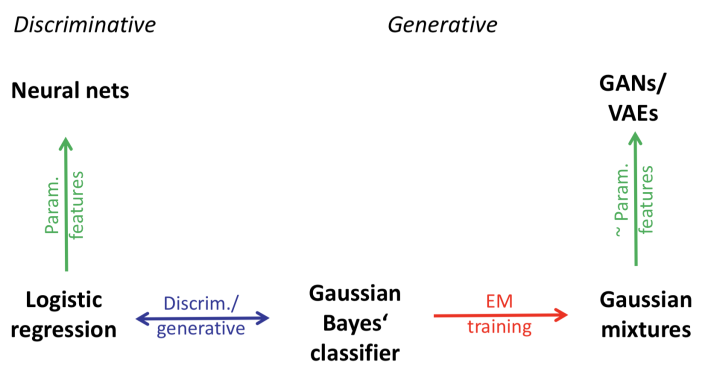
Typical approach to generative modeling

(i) Estimate prior on labels $\mathbb{P}(Y = y)$

(ii) Estimate conditional distribution $\mathbb{P}(\mathbf{X}|Y = y)$ for each class y

(iii) Obtain predictive distribution using Bayes' rule

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



10.2 Naive Bayes Model

10.2.1 Model Description

Assumptions for the naive Bayes model are

(i) Class labels can be modeled as generated from categorical variable $\mathbb{P}(Y = y) = p_y$, $y \in \mathcal{Y} = \{1, \dots, c\}$.

(ii) **Naivety**: Model features are conditionally independent given Y, i.e. given class feature, each data point is "generated" independently of the other features. This assumption is strong and mostly not true, but it still works somehow.

$$\mathbb{P}(X_1, \dots, X_d|Y) = \prod_{i=1}^{d} \mathbb{P}(X_i|Y)$$

(iii) Feature distribution, e.g. for **Gaussian Naive Bayes classifier** we have $\mathbb{P}(X = x_i|Y = y) = \mathcal{N}(x_i|\mu_{y,i}, \sigma^2_{y,i})$. Note that the parameters are class and feature dependent.

Gaussian Naive Bayes Classifier

(i) **Learning** given Data $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ using MLE

MLE for class prior $\hat{\mathbb{P}}(Y = y) = \hat{p}_y = \frac{\#[Y=y]}{n}$

MLE for feature distribution $\hat{\mathbb{P}}(x_i|y) = \mathcal{N}(x_i; \hat{\mu}_{y,i}, \hat{\sigma}^2_{y,i})$

$\hat{\mu}_{i,y} = \frac{1}{\#[Y=y]} \sum_{j:y_j=y} x_{j,i}, \quad \hat{\sigma}^2_{i,y} = \frac{1}{\#[Y=y]} \sum_{j:y_j=y} (x_{j,i} - \hat{\mu}_{y,i})^2$

Warum kein -1? Ist das nicht biased? J: wo ein -1?

(ii) **Prediction** given new data point $\mathbf{x}$

$$\hat{\mathbb{P}}(y|x) = \frac{1}{Z}\mathbb{P}(y)\mathbb{P}(\mathbf{x}|y) \quad Z = \sum_y \mathbb{P}(y)\mathbb{P}(\mathbf{x}|y)$$

$$y = \operatorname*{argmax}_{y'} \hat{\mathbb{P}}(y'|\mathbf{x}) = \operatorname*{argmax}_{y'} \hat{\mathbb{P}}(y') \prod_{i=1}^{d} \hat{\mathbb{P}}(x_i|y').$$

10.2.2 Gaussian NB vs. Logistic Regression

Assumptions are $\mathbb{P}(Y = 1) = 0.5$ (mild assumption) and independent variance, i.e. $\mathbb{P}(\mathbf{x}|y) = \prod_i \mathcal{N}(x_i; \mu_{y,i}, \sigma + i^2)$ (rather strong assumption).

Discriminant function Decision rule for binary classification $y = \operatorname{argmax}_{y'} \hat{\mathbb{P}}(y', \mathbf{x})$ is equivalent to $y = \operatorname{sign}\left(\log \frac{\mathbb{P}(Y=+1|\mathbf{x})}{\mathbb{P}(Y=-1|\mathbf{x})}\right) = \operatorname{sign} f(\mathbf{x})$, where f is called the discriminant function.

GNB recovers linear classifier With the above assumptions Gaussian Naive Bayes produces a linear classifier

$$f(\mathbf{x}) = \log \frac{\mathbb{P}(Y = +1|\mathbf{x})}{\mathbb{P}(Y = -1|\mathbf{x})} = \mathbf{w}^T\mathbf{x} + w_0 \quad \text{with}$$

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

Corresponding class distribution is of the same form as logistic regression, i.e.

$$\mathbb{P}(Y = +1|\mathbf{x}) = \frac{1}{1 + \exp(-f(\mathbf{x}))} = \sigma(\mathbf{w}^T\mathbf{x} + w_0).$$

Conclusion If model assumptions are met, GNB will make the same predictions as Logistic Regression.

10.2.3 Issues with Naive Bayes models

Overconfidence Conditional independence assumption means that features are generated independently given class label. Thus, predictions may become overconfident.

Example For duplicate data points $x_2 = x_3 = \dots = x_d = x_1$ unser certain assumptions, NBM predicts $p_1(\mathbf{x}) = \frac{1}{1+\exp(f_1(\mathbf{x}))}$ and $p_k(\mathbf{x}) = \frac{1}{1+\exp(d \cdot f_1(\mathbf{x}))}$. This gives $p_1(\varepsilon) \approx 0.5 + \varepsilon$, but $p_t(\varepsilon) \approx 01$ for large d (overconfidence).

10.2.4 Categorical Naive Bayes for discrete Features

Setting Model features by (conditionally) independent categorical random variables $\mathbb{P}(X_i = x|Y = y) = \theta^{(i)}_{x,y}$ such that $\theta^{(i)}_{x|y} \geq 0$ for all i, x, y and $\sum_{x=1}^{c} \theta^{(i)}_{x|y} = 1$ for all i, y.

MLE estimation given dataset $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$

- for class label distribution is $\hat{\mathbb{P}}(Y = y) = \hat{p}_y = \frac{\mathbb{1}[Y=y]}{n}$,
- for distribution of features is i $\hat{\mathbb{P}}(X_i = c|y) = \theta^{(i)}_{c|y} = \frac{\mathbb{1}[X_i=c, Y=c]}{\mathbb{1}[Y=y]}$.

Lifting independence assumption requires specification of probability of every possible categorical data, requiring in d exponentially many parameters, which is computationally intractable and a fantastic way to overfit.

10.2.5 Discrete and categorical Features

The (naive) Bayes classifier does not require each feature to follow the same type of conditional distribution, e.g. model some features as categorical and some as Gaussian

$X_{1:10}$ discrete $\mathbb{P}(x_i|y) = \mathit{Categorical}(x_i|y, \theta)$

$X_{11:20}$ Gaussian $\mathbb{P}(x_i|y) = \mathcal{N}(x_i; \mu_{i|y}, \sigma^2_{i|y})$

10.3 Gaussian Bayes Classifier

10.3.1 Model Description

Assumptions for the Bayes model are

(i) Class labels can be modeled as generated from categorical variable $\mathbb{P}(Y = y) = p_y, y \in \mathcal{Y} = \{1, \dots, c\}$.

(ii) Model features as generated by multivariante Gaussian $\mathbb{P}(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \mu_y, \Sigma_y^2)$.

ML for Gaussian Bayes Classifier Given Data $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y$ MLE for feature distribution $\hat{\mathbb{P}}(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \hat{\mu}_y, \hat{\Sigma}_y)$ with estimators $\hat{\mu}_y$ and $\hat{\Sigma}_y$ and MLE for class prior $\hat{\mathbb{P}}(Y = y)$ are given by

$$\hat{\mathbb{P}}(Y = y) = \hat{p}_y = \frac{\#[Y = y]}{n}$$

$$\hat{\mu}_y = \frac{1}{\#[Y = y]} \sum_{i:y_i = y} \mathbf{x}_i \quad \hat{\Sigma}_y = \frac{1}{\#[Y = y]} \sum_{i:y_i = y} (\mathbf{x}_i - \hat{\mu}_y)^T (\mathbf{x}_i - \hat{\mu}_y)$$

Discriminant function is given by

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

10.3.2 Fisher's Linear Discriminant Analysis (LDA)

Fisher's linear discriminant analysis (LDA) for binary classification ($c = 2$). Suppose $p = 0.5$ and $\hat{\Sigma}_+ = \hat{\Sigma}_- = \hat{\Sigma}$, then the discriminant function becomes again a linear function $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$ with

$$\mathbf{w} = (\hat{\mu}_+ - \hat{\mu}_-)^T \hat{\Sigma}^{-1} \quad \text{and} \quad w_0 = \frac{1}{2} \hat{\mu}_-^T \hat{\Sigma}^{-1} \hat{\mu}_- - \frac{1}{2} \hat{\mu}_+^T \hat{\Sigma}^{-1} \hat{\mu}_+.$$

Under these assumptions, we predict $y = \operatorname{sign} f(\mathbf{x})$ which is called Fisher's linear discriminant analysis.

10.3.3 LDA vs. Logistic regression

Corresponding class distribution is of the same form as logistic regression, i.e.

$$\mathbb{P}(Y = +1|\mathbf{x}) = \frac{1}{1 + \exp(-f(\mathbf{x}))} = \sigma(\mathbf{w}^T \mathbf{x} + w_0).$$

Fisher's LDA	Logistic Regression
generative model	discriminative model
+ can be used to detect outliers	- cannot detect outliers
- assumes normality of $\mathbf{x}$	+ makes no assumptions on $\mathbf{X}$
- not robust against violation of this assumption	+ more robust

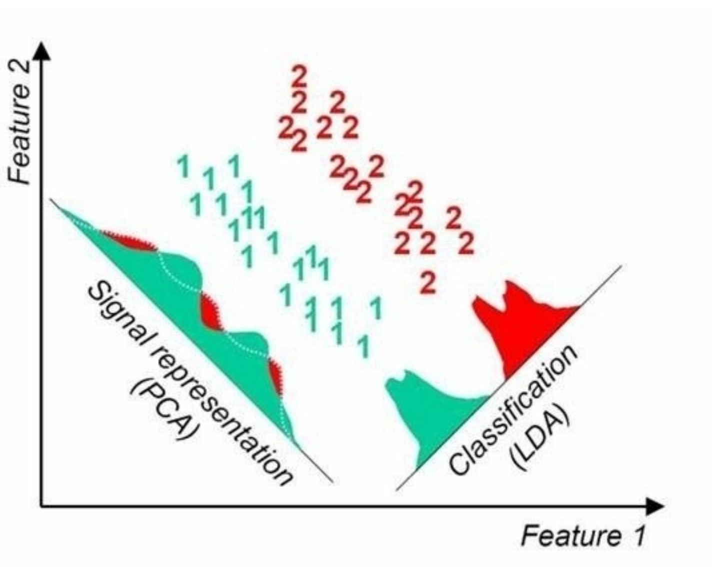
Conclusion If model assumptions are met, Fisher's LDA will make the same predictions as Logistic Regression.

10.3.4 Gaussian Naive Bayes vs. General Gaussian Bayers Classifiers

GNB models	General GB models
- conditional independence assumption may lead to overconfidence	+ captures correlations among features
+ predictions might still be useful	+ avoids overconfidence
+ # parameters = $\mathcal{O}(cd)$	- # parameters = $\mathcal{O}(cd^2)$
+ complexity (memory + inference) linear in d	complexity quadratic in d

10.3.5 LDA vs. PCA

LDA can be viewed as a projection to a 1D subspace that maximizes ration of between-class and within-class variances. In contrast, PCA ($k = 1$) maximizes the variance of the resulting 1D projection.



10.3.6 Quadratic Discriminant Analysis (LDA)

Quadratic discriminant analysis (LDA) uses the non-simplified discriminant function $f(\mathbf{x})$ and predicts using $y = \operatorname{sign} f(\mathbf{x})$.

10.4 Outlier Detection

Data point probability can be calculated as

$$\mathbb{P}(\mathbf{x}) = \sum_{y=1}^{c} \mathbb{P}(y) \mathbb{P}(\mathbf{x}|y) \overset{GBC}{=} \sum_{y=1}^{c} \hat{p}_y \mathcal{N}(\mathbf{x}|\hat{\mu}_y, \hat{\Sigma}_y).$$

Outliers are points for which $\mathbb{P}(\mathbf{x}) \leq \tau$ holds.

10.5 Avoiding Overfitting: Introducing Priors

Avoiding overfitting can be done by

- restricting the model class to reduce the number of parameters, e.g. by further assumptions on covariance structure, e.g. Gaussian Naive Bayesm
- using priors.

Problems with MLE estimation In the extreme case of $n = 1$, the estimator $\hat{\theta} = \frac{\mathbb{1}[Y=1]}{n}$ predicts $\hat{\theta} = \frac{1}{1} = 1$ for $D = \{(y_1)\}, y_1 = 1$, i.e. it is overconfident.

Introducing priors by computing the posterior distribution

$$\mathbb{P}(\theta|y_1, \dots, y_n) = \frac{1}{Z} \mathbb{P}(\theta) \mathbb{P}(y_{i:n}|\theta) \quad Z = \int \mathbb{P}(\theta) \mathbb{P}(y_{1:n}|\theta) \, d\theta$$

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

Definition 10.1 (Conjugate distributions) *A pair of prior distributions and likelihood functions is called* ***conjugate*** *if the posterior distribution remains the same family as the prior.*

Beta conjugate With prior $\operatorname{Beta}(\theta; \alpha_+, \alpha_-)$ and n_+ positive and n_- negative labels, the posterior distribution is $\operatorname{Beta}(\theta; \alpha_+ n_+, \alpha_- + n_-)$. Thus, α_+ and α_- act as pseudo-counts.

Beta MAP estimate

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

Examples of conjugate priors are listed below

Remarks conjugate priors can be used as regularizers why conjugate priors? with almost no computational cost. Choose hyperparameters by crossvalidation.

TODO check alignment adsadsad

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

11 Probabilistic Modeling of Unsupervised Learning: Latent Variable Modeling

We will focus on missing labels, ideas may be applied to missing data as well.

11.1 Gaussian Mixture Models

Assumptions $\mathbb{P}(X,Y)$ is a Gaussian-Bayes classifier: $\mathbb{P}(Y=y)=p_y$ and $\mathbb{P}(\mathbf{x}|y)=\mathcal{N}(\mathbf{x};\mu_y,\Sigma_y)$. We also require i.i.d. data.

Gaussian mixtures are convex combinations of Gaussians

$$\mathbb{P}(\mathbf{X}=\mathbf{x}|\theta)=\mathbb{P}(\mathbf{X}=\mathbf{x}|\mu,\Sigma,\mathbf{w})=\sum_i w_i\mathcal{N}(\mathbf{x};\mu_i,\Sigma_i),$$

where $w_i\geq 0$ and $\sum_i w_i=1$.

Mixture modeling models each cluster $i\in\{1,\dots,k\}$ as a probability distribution $\mathbb{P}(\mathbf{x}|\theta_j)$. Using i.i.d. assumption of data, the likelihood of data is

$$\mathbb{P}(D|\theta)=\prod_{i=1}^{n}\sum_{j=1}^{k}w_j\mathbb{P}(\mathbf{X}=\mathbf{x}_i|\theta_j).$$

Optimization problem by minimizing the negative log likelihood

$$(\mu^*,\Sigma^*,w^*)=\operatorname{argmin}-\sum_i\log\sum_{j=1}^{k}w_j\mathcal{N}(\mathbf{x}_i|\mu_j,\Sigma_j),$$

$$\text{while}\quad\sum_{j=1}^{k}w_j=1\quad\text{and}\quad\Sigma_j\text{ positive definite}$$

is non-convex and constrained. One could try to optimize it using (stochastic) gradient descent, but the constraints might be difficult to maintain.

Choosing k Same as for k-means. However, for GMMs typically cross-validation works fairly well.

GMMs for density estimation and not for clustering by for example modeling $\mathbb{P}(\mathbf{x})$ as Gaussian mixture and $\mathbb{P}(\mathbf{x}|y)$ using logistic regression, neural network, etc. Then $\mathbb{P}(\mathbf{x},y)=\mathbb{P}(\mathbf{x})\mathbb{P}(y|\mathbf{x})$ is a valid model. This combines the advantage of accurate predictions and robustness from discriminative model with the ability to detect outliers.

Anomaly detection with mixture models by comparing the estimated density of a data point against a threshold. If we do not have any examples of anomalies, this is challenging. If we do have some examples, we could try

- varying the threshold to trade false-positives and false-negatives,
- to use precision-recall curves/ ROC curves as evaluation criterion, e.g. maximizing $F1$-score.

This allows to optimize the threshold, e.g. via cross-validation.

Why are mixture models useful

- Can encode assumptions about shape of clusters, e.g. fit ellipses instead of points.
- Can be part of more complex statistical models, e.g. classifiers (or more general probabilistic models)
- Probabilistic models can output likelihood $\mathbb{P}(\mathbf{x})$ of a point $\mathbf{x}$. This can be useful for anomaly detection.
- Can be naturally used for semi-supervised learning.

11.2 Expectation-Maximization Algorithm

Latent variables are variables that are not directly observed but are rather inferred (through a mathematical model) from other variables that are observed (directly measured). Mathematical models that aim to explain observed variables in terms of latent variables are called latent variable models. Concretely, this means for each data point $\mathbf{x}_i$ we introduce a latent variables z_i denoting the class that this points is assigned.

EM algorithm is an iterative method to find maximum likelihood (ML) or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step and a maximization (M) step described below.

E-step creates a function Q for the expectation of the complete data log-likelihood

$$L(\theta;\mathbf{x})=\log\mathbb{P}(\mathbf{X}=\mathbf{x};\theta)=\log\sum_z\mathbb{P}(\mathbf{X}=\mathbf{x},Z=z;\theta)$$

evaluated using the current estimate for the parameters by computing

$$\gamma_z(\mathbf{x})=\mathbb{P}(Z=z|\mathbf{X}=\mathbf{x},\theta^{(t-1)}).$$

M-step computes parameters maximizing the expected log-likelihood found on the E step by optimizing

$$\begin{aligned}\theta^{(t+1)}&=\operatorname*{argmax}_\theta Q(\theta;\theta^{(t-1)}),\\ Q(\theta,\theta^{(t)})&=\mathbb{E}_{\mathbf{Z}|\mathbf{X}=\mathbf{x},\theta^{(t)}}\left[\log\mathbb{P}(\mathbf{X},Z|\theta)\right]\\ &=\sum_{i=1}^{n}\mathbb{E}_{\mathbf{Z}|\mathbf{X}=\mathbf{x}_i,\theta^{(t)}}\left[\log\mathbb{P}(\mathbf{X}=\mathbf{x}_i,Z|\theta)\right]\\ &=\sum_{i=1}^{n}\sum_{z=1}^{k}\mathbb{P}(Z=z|\mathbf{X}=\mathbf{x}_i,\theta^{(t)})\log\mathbb{P}(\mathbf{X}=\mathbf{x}_i,Z=z|\theta)\\ &=\sum_{i=1}^{n}\sum_{z=1}^{k}\gamma_z(\mathbf{x}_i)\log\left(\mathbb{P}(Z=z)\mathbb{P}(\mathbf{X}=\mathbf{x}_i|Z=z;\theta)\right)\end{aligned}$$

These parameter-estimates are then used to determine the distribution of the latent variables in the next E step. This procedure is equivalent to training a GBC with weighted data and admits a closed form solution.

11.2.1 Hard-EM Algorithms

Fitting a GMM = training a GBC without labels Idea is to repeatedly fill in or update the missing data and then train the resulting dataset. The algorithms assigns (only) a label to each data point which is why it called hard.

Algorithm 17 Hard Expectation Maximization (EM)

```
1: Initialize the parameters $\theta^{(0)}$
2: for $t = 1,2,3,\dots$ do
3:     E-Step: Predict most likely class for each data point
           $z_i^{(t)} = \operatorname*{argmax}_z \mathbb{P}(z|\mathbf{x}_i,\theta^{(t-1)})$
           $\quad = \operatorname*{argmax}_z \mathbb{P}(z|\theta^{(t-1)})\mathbb{P}(\mathbf{x}_i|z,\theta^{(t-1)})$
4:     and complete the data $D^{(t)} = \{(\mathbf{x}_1,z_1^{(t)}),\dots,(\mathbf{x}_n,z_n^{(t)})\}$.
5:     M-Step: compute MLE as for the Gaussian Bayes classifier
           $\theta^{(t)} = \operatorname*{argmax}_\theta \mathbb{P}(D^{(t)}|\theta)$
```

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

11.2.2 Soft-EM Algorithm

Posterior probabilities given a model $\mathbb{P}(z|\theta)$, $\mathbb{P}(\mathbf{x}|z,\theta)$, we can compute a posterior distribution over cluster membership

$$\gamma_j(\mathbf{x}) = \mathbb{P}(Z=j|\mathbf{X}=\mathbf{x},\Sigma,\mu,\mathbf{w}) = \frac{w_j \mathbb{P}(\mathbf{X}=\mathbf{x}|\Sigma_j,\mu_j)}{\sum_l w_l \mathbb{P}(\mathbf{X}=\mathbf{x}|\Sigma_l,\mu_l)}$$

MLE At MLE $(\mu^*,\Sigma^*,w^*) = \operatorname{argmin} -\sum_i \log \sum_{j=1}^k w_j \mathcal{N}(\mathbf{x}_i|\mu_j,\Sigma_j)$ it must hold that

$$\mu_j^* = \frac{\sum_{i=1}^n \gamma_j(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^n \gamma_j(\mathbf{x}_i)}$$
$$\Sigma_j^* = \frac{\sum_{i=1}^n \gamma_j(\mathbf{x}_i)(\mathbf{x}_i-\mu_j^*)^T(\mathbf{x}_i-\mu_j^*)}{\sum_{i=1}^n \gamma_j(\mathbf{x}_i)}$$
$$w_j^* = \frac{1}{n}\sum_{i=1}^n \gamma_j(\mathbf{x}_i)$$

Algorithm 18 Soft Expectation Maximization (EM)

```
1: while not converged do
2:     E-Step: calculate $\gamma_j^{(t)}(\mathbf{x}_i)$ for each $i$ and $j$ given estimates of $\mu^{(t-1)}$, $\Sigma^{(t-1)}$, $\mathbf{w}^{(t-1)}$ from previous iteration
3:     M-Step: fit clusters to weighted data points, i.e. calculate $w_j^{(t)}$, $\mu_j^{(t)}$, and $\Sigma_j^{(t)}$.
```

Constrained GMMS are special cases of Gaussian mixtures

- Spherical $\Sigma_j = \sigma_j^2 \cdot \mathbf{I}$, with $\#params = k$
- Diagonal $\Sigma_j = \operatorname{diag}(\sigma_{j,1}^2,\dots,\sigma_{j,d}^2)$, with $\#params = dk$
- Tied $\Sigma_1 = \dots = \Sigma_k$, with $\#params = \frac{d(d+1)}{2}$
- Full, with $\#params = k\frac{d(d+1)}{2}$

Discussion Soft EM will typically result in higher likelihood values, because it can deal better with "overlapping clusters".

11.2.3 Theory behind EM

Convergence of EM One can prove that the EM algorithm monotonically increases the likelihood $\log \mathbb{P}(\mathbf{x}_{1:n}|\theta^{(t)}) \geq \log \mathbb{P}(\mathbf{x}_{1:n}|\theta^{(t-1)})$. For Gaussian mixture, EM is guaranteed to converge to a local minimum, but the quality of solution highly depends on initialization. A common strategy is to rerun the algorithm multiple times and use the solution with largest likelihood.

Initialization

- for weights: typically use a uniform distribution $w_i^{(0)} = \frac{1}{k}\ \forall i$
- for means use random initialization or k-means++, i.e. pick $\mu_i^{(0)} = x_{j_i}$
- for variances for example initialize according to empirical covariance of the data (perhaps restrict to spherical) $\Sigma_1 = \dots = \Sigma_k = \frac{1}{n}\sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$.

Hard EM performs alternating optimization of the complete data likelihood

$$z_{1:n}^{(t)} = \operatorname*{argmax}_{z_{z:n}} \mathbb{P}(\mathbf{x}_{1:n}, z_{1:n}|\theta^{(t-1)}) \qquad \text{(E-step)}$$
$$\theta^{(t)} = \operatorname*{argmax}_{\theta} \mathbb{P}(\mathbf{x}_{1:n}, z_{1:n}|\theta) \qquad \text{(M-step)}$$

and converges to a local optimum of $\max_{z_{1:n},\theta} \mathbb{P}(\mathbf{x}_{1:n}, z_{1:n}|\theta)$.

EM more generally EM algorithm is more widely applicable. It can be used whenever the E and M steps are traceable, i.e. we must be able to compute and maximize the complete data likelihood. This can be used for example for imputing (some) missing features and handling likelihoods beyond Gaussian (e.g. categorical).

11.2.4 EM vs. k-means

Assumptions are uniform weights over mixture components and identical spherical covariance matrices.

Hard EM recovers k-means under the above assumptions. The steps in the hard EM algorithm become the same decisions as Lloyd's heuristic

$$z_i^{(t)} = \operatorname*{argmax}_z \mathbb{P}(z|\theta^{(t-1)}) = \operatorname*{argmin}_z \|x_i - \mu_z^{(t-1)}\|, \qquad \text{(E-step)}$$
$$\mu_j^{(t)} = \frac{1}{n_j}\sum_{i:z_i^{(t)}=j} x_i. \qquad \text{(M-step)}$$

Soft EM recovers k-means under the above assumptions and additionally variances tending to 0, because for $\sigma^2 \to 0$ it holds that $\gamma_i(x) \to \begin{cases} 1 & \mu_i \text{ is closest to } x \\ 0 & \text{otherwise} \end{cases}$.

11.3 Avoiding Overfitting with GMMs

Degeneracy For only one data point, the optimal log-likelihood is $-log\mathbb{P}(x|\mu,\sigma) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(x-\mu)^2 \to -\infty$ for $\mu = x$ and $\sigma^2 \to 0$, i.e. the minimization problem is not bounded from below. Thus, the "optimal" GMM chooses $k = n$ and puts one Gaussian around each data point with variance tending to 0. This is overfitting.

Adding a Wishart prior to the covariance matrix and computing the MAP instead of MLE can regularize the problem and thus avoid degeneracy (variances $\to$ 0). The corresponding update rule reads

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

11.4 Gaussian-Mixture Bayes classifier

Given labeled data set $D = \{(\mathbf{x}_1,y_1),\dots,(\mathbf{x}_N,y_N)\}$ with labels $y_i \in \{1,\dots,m\}$, estimate class prior $\mathbb{P}(y)$ and conditional distribution for each class as Gaussian mixture model

$$\mathbb{P}(\mathbf{x}|y) = \sum_{j=1}^{k_y} w_j^{(y)} \mathcal{N}(\mathbf{x};\mu_j^{(y)},\Sigma_j^{(y)}).$$

Classification is done by Bayes rule

$$\mathbb{P}(y|\mathbf{x}) = \frac{1}{Z}\mathbb{P}(y)\sum_{j=1}^{k_j} w_j^{(y)} \mathcal{N}(\mathbf{x};\mu_j^{(y)},\Sigma_j^{(y)}).$$

11.5 Semi-supervised Learning with GMMs

We would like to combine unlabeled and labeled data.

Semi-supervised learning is learning from large amounts of unlabeled data and small amounts of labeled data.

Modification to EM algorithm The computation of γ also takes into account the labeled data points $\mathbf{x}_i$

$$\gamma_j(\mathbf{x}_i) = \begin{cases} \mathbb{P}(Z=j|\mathbf{x}_i,\Sigma,\mu,\mathbf{w}) & \mathbf{x}_i \text{ is unlabeled} \\ [j = y_i] & \mathbf{x}_i \text{ is labeled with label } y_i \end{cases}$$

The computation of w_j, μ_j and Σ_j does not change.

11.6 Outlook: Implicit generative Models

Given sample of (unlabeled) points $\mathbf{x}_1,\dots,\mathbf{x}_n$, the goal is to learn a model $\mathbf{X} = f(\mathbf{Z};\mathbf{w})$. The approach is to optimize parameters $\mathbf{w}$ to make samples from model hard to distinguish from data sample.

A Convex functions

Theorem A.1 (Jensen's inequality) *Let f be convex and $x_1,\dots,x_n \in \mathbb{R}^d$, $\lambda_1,\dots,\lambda_n \in [0,1]$, such that $\sum_{i=1}^n \lambda_i = 1$. Then*

$$f(\lambda_1 x_1 + \dots + \lambda_n x_n) \leq \lambda_1 f(x_1) + \dots + \lambda_n f(x_n)$$

Also, if $\boldsymbol{x} \in \mathbb{R}^d$ is a random variable, then

$$f(\mathbb{E}[\boldsymbol{x}]) \leq \mathbb{E}[f(\boldsymbol{x})].$$

Theorem A.2 (Gradient inequality) *Let f be convex, then $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$*

$$f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \nabla f(\boldsymbol{x})^T (\boldsymbol{y} - \boldsymbol{x}).$$

Theorem A.3 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be convex and $\mathbf{A}, \mathbf{b}$ such that $\forall \boldsymbol{z} \in \mathbb{R}^d$ $\boldsymbol{A}\boldsymbol{z} + \boldsymbol{b} \in \mathbb{R}^d$. Then $g(\boldsymbol{z}) = f(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{b})$ is convex in $z \in \mathbb{R}^n$.*

Theorem A.4 *If f is convex and $x^* \in \mathbb{R}^d$ sucht that $\nabla f(x^*) = 0$, then x^* is a global minimizer, i.e. $f(x^*) \leq f(x)$ $\forall x \in \mathbb{R}^d$.*

Bibliography

[1] StackExchange. How to choose the number of hidden layers and nodes in a feedforward neural network?, 2010.