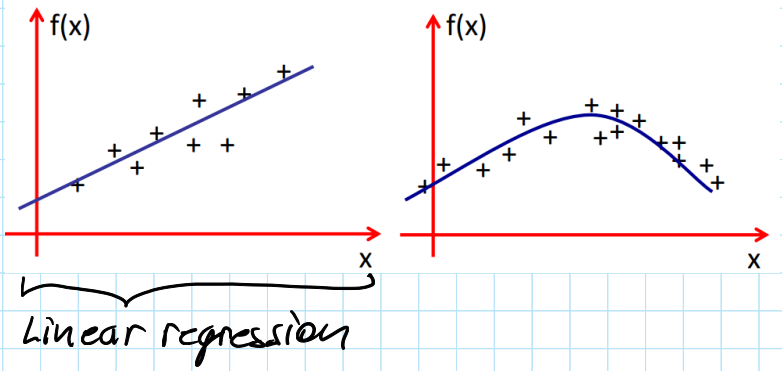
# Linear Regression

- Input: X: Age, Sex, BMI
- Output: y: Measure of disease progression
- Goal: Learn real valued mapping




Linear regression

- Linear regression: $y = f(x)$

$$f(x) = w^T x + b \quad (\text{e.g. } ax + b)$$

$$\hat{w}^T \hat{x}$$

- $\hat{x} = [x_1, x_2, \ldots, x_d, 1]$
- $\hat{w} = [w_1, w_2, \ldots, w_d, b]$
- How to find optimal weight vector:

$$\hookrightarrow w^* = \text{argmin} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

↳ Least squares error
↳ Minimize error

- Residual: $r_i = y_i - f(x_i)$
  (↳ "Restwert" $= y_i - w^T x_i$)

$r_i \in \mathbb{R}$

- Squared Residual: $R(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$
- Closed form solution: $w^* = (X^T X)^{-1} \cdot X^T y$

↳ $X$ = Data point inputs
↳ $y$ = outputs corresponding to inputs

- computational complexity: $O(d^3)$
- Input data is just reference data
  ↳ No need for exact solution

- Optimization: · Minimize $\hat{R}(w) = \sum_i (y_i - w^T x_i)^2$ with gradient descend
· $\hat{R}(w)$ is convex: local minima $\hat{=}$ global minima
1. Start at an arbitrary point $w_0 \in \mathbb{R}^d$
2. $w_{t+1} = w_t - \eta \nabla \hat{R}(w_t)$
· $\eta$: - Learning rate / step size
- If we choose a poor stepsize
↳ oscillation 

- Adaptive step size:
· Line search: - $\eta_t = \text{argmin}\ \hat{R}[w_t - \eta \cdot \nabla \hat{R}(w_t)]$
- choose stepsize that gives biggest improvement towards 0
· Bold driver: - If function decreases, increase step-size
↳ $\hat{R}(w_{t+1}) < \hat{R}(w_t) \Rightarrow \eta_{t+1} = \eta_t \cdot 1{,}1$
- If function increases, decrease step-size
↳ $\hat{R}(w_{t+1}) > \hat{R}(w_t) \Rightarrow \eta_{t+1} = \eta_t \cdot 0{,}5$

---

Loss functions

- Least squared loss: $L(r) = r^2$
↳ If I under/overpredict, it's the same, because square gets rid of sign
- Absolute loss: $L(r) = |r|$
↳ Overvalue small errors / undervalue big errors
↳ More robust to ausschlägen

## Over- / Underfitting

- Overfitting: Too high degree polynomial
- Underfitting: Too low degree polynomial
- The higher the degree, the better we fit the data

---

## Prediction Error

- Wenn wir zufällig ein x einfügen, was ist dann der expected error
- We assume, our testdata is drawn from the distribution P: $(x_i, y_i) \sim P(x, y)$
- True risk: · Expected error

  probability that (x,y) pair occurs

  Error on that pair

  $$\cdot R(w) = \int P(x,y) \cdot (y - w^T x)^2 \cdot dx \cdot dy$$

  $$= E_{x,y}[(y - w^T x)^2]$$

  · Cannot be computed since we don't know the real distribution of P

- Estimating true risk

  · Empirical risk ≈ true risk

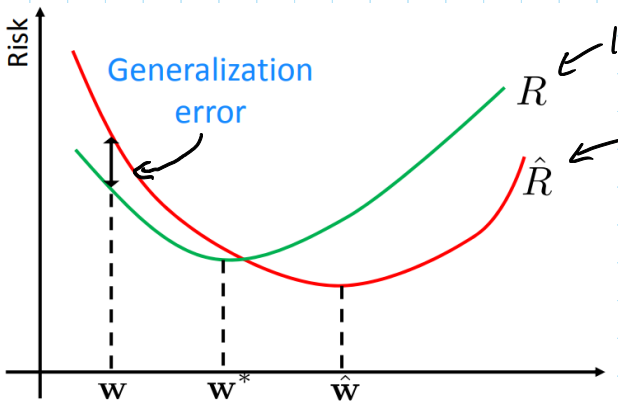
  ↳ If we have enough data points

  ↳ Law of large numbers

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

  Total error

  ↳ Was ist der expected error, wenn wir y ausgeben




← If we know probability distribution

← Training set

- To be successful: $|R(w) - \hat{R}_D(w)| \to 0$ as $D \to \infty$

- $\underbrace{E[R_{TrainData}(\hat{w})]}_{\text{only training data}} << \underbrace{E[R(\hat{w})]}_{\text{whole distribution}}$ (Da wir viel mehr Punkte haben)
  → Underestimating prediction error

---

## Avoid underestimating prediction error

- Split data into training and test set
1. Optimize $\hat{w}$ using training data
   - $\hat{w} = \text{argmin}\ \hat{R}_{trainData}(w)$
2. Evaluate prediction error on test data
   - $\hat{R}_{testData}(\hat{w}) = \frac{1}{|D_{Testdata}|} \cdot \sum_{(x,y) \in \text{Test data}} (y - \hat{w}^T x)^2$

---

## Choosing polynomial degree

1. Do for each degree m the following for $i = 1, \dots, k$
2. Split data into training/test-data: $D = D_{Train}^{(i)} \dot{\cup} D_{Test}^{(i)}$
3. Train model: $\hat{w}_{i,m} = \text{argmin}\ \hat{R}_{Train}^{(i)}(w)$
4. Estimate error: $\hat{R}_m^{(i)} = \hat{R}_{test}^{(i)}(\hat{w}_{i,m})$
5. select polynomial $\hat{m} = \underset{m}{\text{argmin}}\ \frac{1}{k} \sum_{i=1}^{k} \hat{R}_m^{(i)}$

Durchschnittlicher Error für Polynom mit grad m

---

## Cross validation

- Monte-Carlo cross validation: Randomly
- K-fold cross validation: 1. Partition data into k-folds
  2. Train on (k-1) folds
  3. Test on 1 fold

- How to choose # folds?
  - Too small: · Overfitting to test data
    · Underfitting to training data

---

Regularization (= Encourage small weights with penalty)

- If we only optimize loss functions, we'll get large weights

---

Ridge Regression (= L2-regularized Regression)

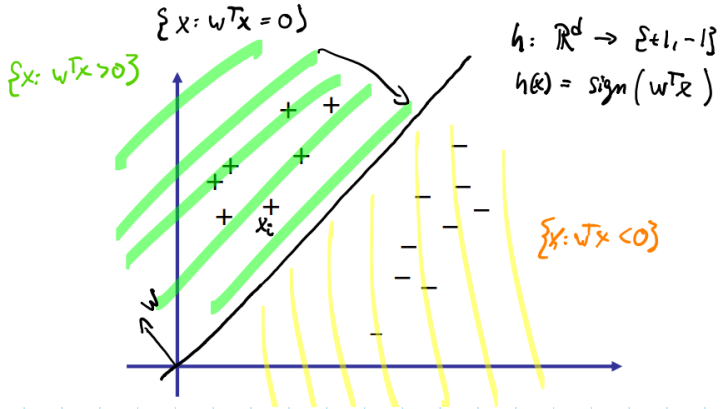
- $\hat{w} = \min \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \underbrace{\|w\|_2^2}_{\sum w_i^2}$
- Closed solution: $\hat{w} = (X^T X + \lambda I)^{-1} \cdot X^T y$
- How to choose $\lambda$: cross validation
- The greater $\lambda$, the lower the weights

---

Classification

- y is discrete (e.g. spam / not spam)

- Data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

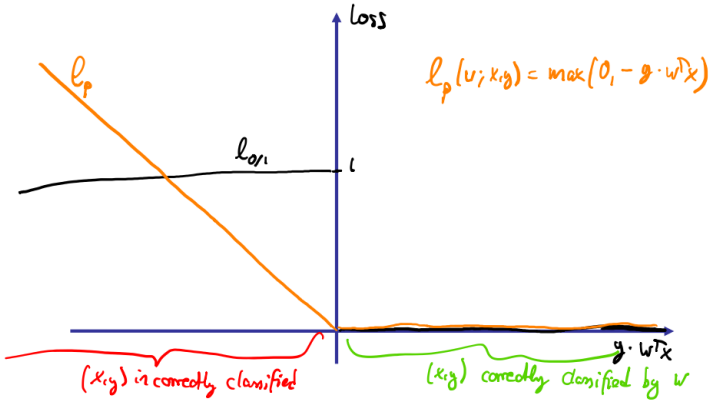



- $w^T x$ either $> 0$ / $< 0$
- $h(x) = \text{sign}(w^T x)$

- We have to find a linear separator
- 0/1 loss is not convex => Surrogate loss

## Surrogate Loss

- Solve $\hat{w} = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} l_p(w, x_{x_i}, y_i)$
- Perceptron loss: $l_p(w, x_i, y_i) = \max(0, -y_i \cdot w^T x_i)$
- $-y_i w^T x_i > 1$ if $\cdot\ y = +1$ and $w^T x_i = -1$ } Error
  $\cdot\ y = -1$ and $w^T x_i = +1$ }

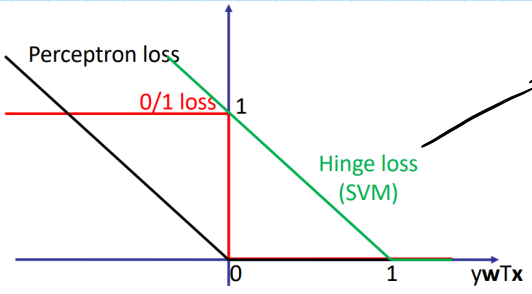



## Stochastic gradient descend

- Computing the gradient requires summing over all data
- For large data sets, this is inefficient
- Use a part of the set or even only one point
- Mini batches: · Averaging over the gradients of randomly selected points

## Support Vector Machines (= SVMs)

- Want to maximize the margin/boundary between separating line and first data points
- Hinge Loss: $l_H(w, x, y) = \max(0, 1 - y \cdot w^T x)$




→ Encourages maximizing the margin

- $\hat{w} = \arg\min_w \frac{1}{n} \cdot \sum_{i=1}^{n} \max(0, 1 - y_i w^T x_i) + \underbrace{\lambda \|w\|_2^2}_{\text{Regularization}}$

---

## Feature selection

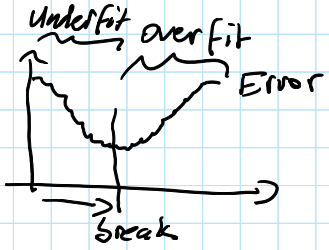
### - Greedy feature selection

- Greedily add/remove features to maximize
- Set of all features $X = \{1, \dots, d\}$

1. Only train with subset $S$
2. $\hat{L}(S)$ := cross validation error using only features in $S$

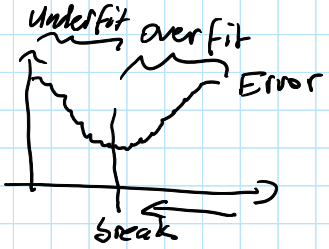
### - Greedy forward selection

- for $i = 1, \dots, d$
  - find best feature to add: $s_i = \arg\min_{j \in V \setminus S} \hat{L}(S \cup \{j\})$
  - Compute error with $S \cup s_i$
  - If $E_i > E_{i-1}$ => break
  - Else $S \leftarrow S \cup \{s_i\}$
- Faster than backward




### - Greedy backward selection

- for $i = d-1, \dots, 1$
  - find best feature to remove: $s_i = \arg\min_{j \in S} \hat{L}(S \setminus \{j\})$
  - Compute error without $s_i$
  - If $E_i > E_{i+1}$ break
  - else $S \leftarrow S \setminus \{s_i\}$



## Sparse Regression: Lasso (= L1-regularized regression)

- We only want the most important features, therefore constraining the sparsity of $w$ induces how many features are used
- Replace $\|w\|_2^2$ by $\|w\|_1$
- $\min_w \frac{1}{n} \sum (y_i - w^T x_i) + \lambda \cdot \|w\|_1$
- This encourages coefficients to be exactly 0 which induces automatic feature selection
- Can be used in SVM, too!

---

## Kernels

- used for non-linear classification
- The weights are just: $w \overset{!}{=} \sum y_i \cdot \alpha_i \cdot x_i$
- We only need the inner/dot product
  ↳ $x^T x \to \phi(x)^T \phi(x') =: k(x, x')$
- Polynomial kernel: $k(x, x') = (1 + x^T x')^m$
- Kernel Trick: 1. Express problem such it only depends on inner product $x^T \cdot x$
  2. Replace inner product by kernel
- Reformulating perceptron:
  - $\hat{w} = \text{argmin} \; 1/n \sum \max(0, -y_i \cdot w^T x_i)$ /!
- Kernel requirements:
  - symmetric $k(x', x) = k(x, x')$
  - positive semi-definite

$$-y_i \left(\sum y_j \cdot \alpha_j \cdot x_j\right)^T x_i$$

$$-y_i \sum y_j \cdot \alpha_j \cdot \underbrace{(x_j^T \cdot x_i)}_{\text{inner product}}$$

- Kernel engineering
  - $c \cdot k_1(x, x') + [k_2(x, x') \cdot k_3(x, x')] + f(k_4(x, x'))$
    ↳ Composition of kernels are valid kernels

- Kernels as similarity function
  - Given two D dimensional vectors $x_i$ and $x_j$
  - The Gaussian kernel is defined as $K(x_i, x_j) = \exp\left(\frac{-\|x_i - x_j\|^2}{\sigma^2}\right)$
  - $\|(x_i - x_j)\|^2$ is the euclidean distance between $x_i$ and $x_j$
  - So the Gaussian Kernel is a similarity measure
  - If it evaluates to 1 $\Leftrightarrow$ $x_i = x_j$
  - And the more it approaches 0, the further apart they are

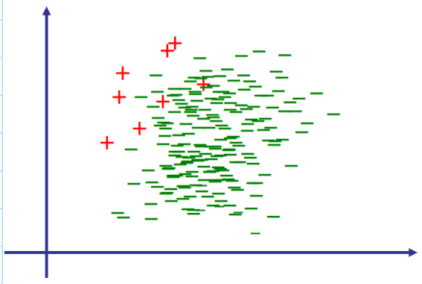
---

Parametric vs. non-Parametric learning

- Parametric models: · finite set of date according to dimension
  · Even if we add new (x,y) pairs
  · e.g linear regression
- Nonparametric model: · Model grow in complexity with the size of data
  · Kernalized perceptron
- Kernels convert parametric to nonparametric
  - For each new (x,y) pair, we need new $\alpha_j$ variable

---

Class Imbalance



- Sources of imbalanced data
  · Fraud detection
  - Spam filtering
- Subsampling: Remove data from majority class randomly
- Upsampling: Repeat data from minority class with small perturbations

- Cost sensitive Classification
  - Modify Perceptron / SVM to take class balance into consideration
  - Add variable c to loss function which has different values for +/- controlling the tradeoff

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

with parameters $c_+, c_- > 0$ controlling tradeoff

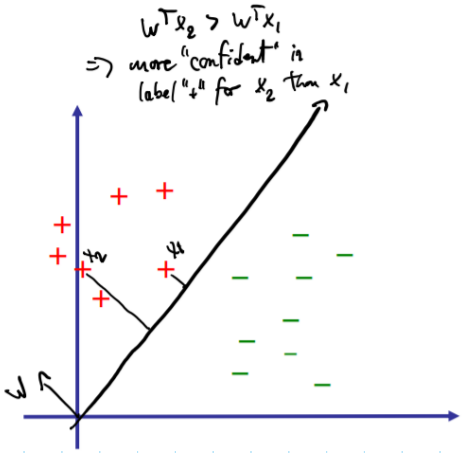
**True label**

| **Predicted label** | Positive | Negative | |
|---|---|---|---|
| Positive | TP | FP | $\Sigma = p_+$ |
| Negative | FN | TN | $\Sigma = p_-$ |
| | $\Sigma = n_+$ | $\Sigma = n_-$ | $p_+ + p_- = n_+ + n_- = n$ |

---

Dealing with multiple classes

- One vs. all: Solve C binary classifiers, one for each class
  - Positive: all points from class C
  - Negative: all points from other classes
  - Classify x by looking for classifier with largest confidence




Je weiter weg der Punkt von der Linie ist, desto höher ist der Wert von $w^Tx$

- One vs. one: · Use a classifier for each pair of classes $(i,j)$
  · Class with highest number of positive predictions, win
- Multi-class-hinge loss: · Have a weight-vector for each of the $c$ classes
  · $w^{(1)}, \dots, w^{(c)}$
  · Given a data-point, we want to train $\underbrace{w^{(r)T}}_{\text{confidence in correct label}} x \geq \underbrace{w^{(i)T}}_{\text{confidence in other labels}} x + 1$

---

Artificial Neural Networks (ANNs / Multi layer perceptron)

- Each neuron holds a value: · Between 0-1
  · $v_i^{(j)} = \varphi(w_i^{(j)T} \cdot x)$
- Activation function: · $\varphi(x) \to (0,1)$
  · e.g. sigmoid function
- Training: · We use backward propagation
  · Randomly set all weight values
  · Apply loss function to output
  · When predicting multiple outputs at the same time, define loss as sum of per output losses
  · Optimize the weights to minimize loss
  ↳ $w^* = \arg\min \sum \ell(W; y_i, x_i)$
  · Use stochastic gradient descend

- Prediction: Using forward propagation
  - Predict $y_i = f_i$ for regression
  - Predict $y_i = \text{sign}(f_i)$ for classification
  - Use maximum value of all outputs

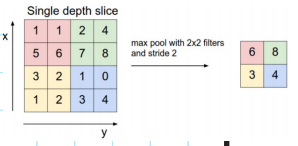
  } 1 output

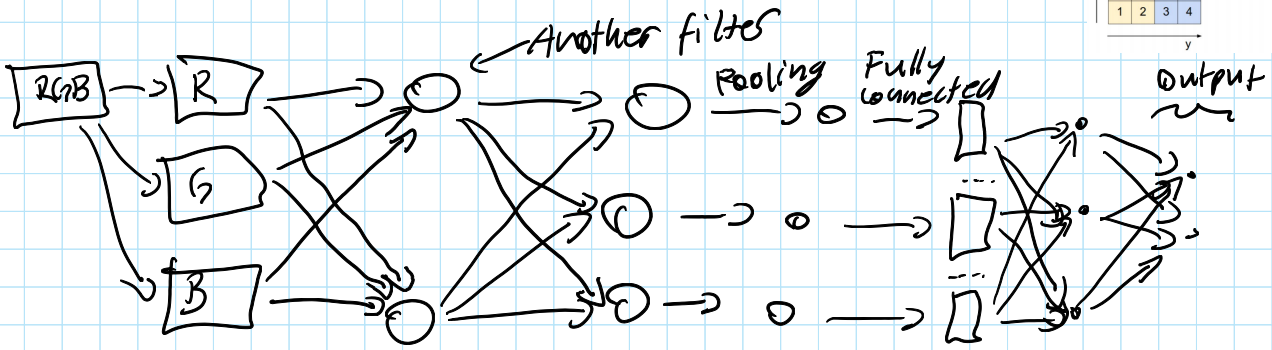
- Overfitting:
  - Don't run SGD until convergence
  - Add regularization to keep weight small

---

## Convolutional Neural Network (=CNN)

- Input → Convolution → Pooling → Fully connected → Output
- Filter:
  - Padding on borders to let each pixel be at the center
  - Stride: move (x, x) at a time (normally (1, 1))
    ↳ helps reducing the size of output
  - Filters are initialized randomly
- Pooling: Aggregate several units to decrease width of the network by averaging them




All 3 images run through the same filter generates a tensor

---

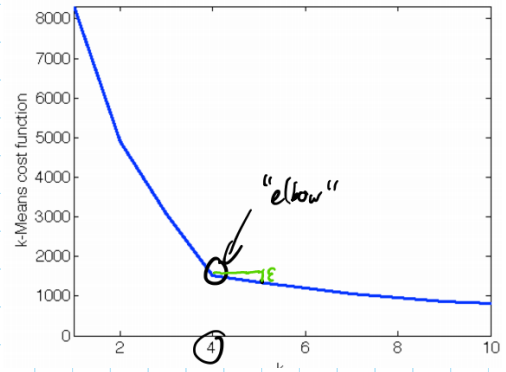
## Learning with momentum

- Can help to escape a local minima
- Move not only in the direction of the gradient, but also in the direction of the last weight update
- $a = m \cdot a + \eta \cdot \nabla \ell(w, y, x)$

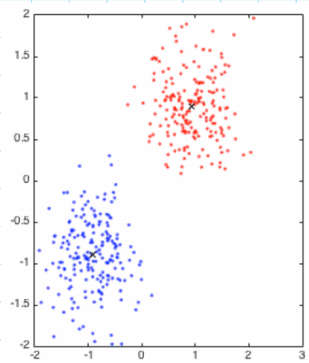
  momentum $< 1$

# Clustering

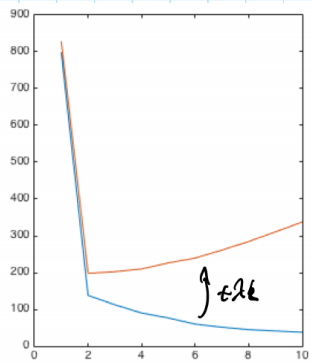
- Given data points, group into clusters
- K-means:
  - Represent each cluster by single center point
  - Assign points to closest center-point
  1. Initialize cluster centers $\mu^{(0)} = [\mu_1^{(0)}, \ldots]$ randomly
  2. Assign each point to closest center
  3. Update center as mean of all points assigned to that center
  - How to choose # of classes = # centers:




- Can't use cross validation because loss keeps decreasing
  ↳ Look for elbow
- you can add regularization to add penalty for each additional class



$\min_{k,\mu_{1:k}} \hat{R}(\mu_{1:k}) + \lambda \cdot k$




Equivalent to "elbow method" for $\lambda = 1$

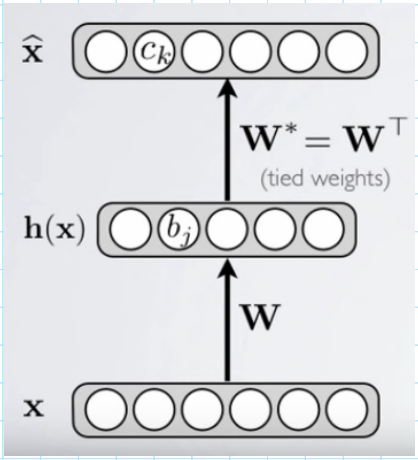
- K-means++ : · start with one center randomly
  · Add centers 2, ..., k randomly, proportionally to squared distance to closest selected center

---

## Dimension reduction

- We want to reduce dimension from d to k
- We have to translate features $x_{(d,1)}$ to $z_{(k,1)}$
- $W \cdot z_i = x_i$ (We have to find W with dimensions $(d \times k)$
- $z_i = W^T x_i$
- To get unique results, we have to center data to covariance-matrix
  ↳ $\Sigma = \frac{1}{d} \sum_{i=1}^{d} x_i \cdot x_i^T = \frac{1}{n} \cdot X^T \cdot X$
- PCA : · $[W, z_1, \ldots, z_n] = \text{argmin} \sum_{i=1}^{n} \|W z_i - x_i\|_2^2$
  · W is either given bei Eigendecomposition or SVD
  · $W = [v_1 | \ldots | v_k]$ $\quad \Sigma = \sum_{i=1}^{d} \lambda_i v_i \cdot v_i^T$ $\quad \Sigma = USV^T$
- Kernel PCA : · For non-linear dimension reduction
  · First increase dimension to get linear mapped data and then reduce dimension to k
  · $K = k^T \cdot k$
  · $\alpha^* = \arg\max \alpha^T \cdot K^T \cdot K \cdot \alpha$
  · $\alpha^{(i)} = 1/\sqrt{\lambda_i} \cdot v_i$
  · A new point is projected:
  ↳ $z_i = w^T x = \underbrace{\left(\sum \alpha_j \cdot x_j\right)^T}_{w} \cdot x = \sum \alpha_j \cdot (x_j^T x)$
  $= \sum \alpha_j \cdot k(x_j, x)$

- Autoencoder:




- $x \approx \hat{x}$
- We use W as weights for dimension reduction

---

## Probability Model

- What is the best prediction if we would know the probability and could use any function $h(x)$
- Least squares:
  - $\min_h E_{x,y}[(y - h(x))^2]$

    $= E_x[\min_{h(x)} E_y[(y - h(x))^2 | X = x]]$

    ↑ For each x we could use another function
  - $\hat{y} = E[Y | X = y]$

    ↳ Conditional mean is the best predictor for y, given x

    ↳ Baye's optimal predictor
  - But we never have true distribution, therefore estimate conditional distributions

    ↳ use MLE to search for parameters for distribution with using the train-data
  - When solving linear regression, we do maximum likelihood on the data

## Logistic Regression (= Discriminitative)

- Predicts wheter something is true/false instead of a continous output
- Returns a probability between (0, 1)
- We use Maximum Likelihood for positioning the S-curve to the data
- Use S-curve to get probability

---

## Bayesian decision theory

- Recommendation for picking the action → Labeling features, eg. $\xi_{1,3}$ that minimizes the expected cost
- $a^* = \underset{a \in A}{\operatorname{argmin}} \; E_y[C(y, a) | x)$
- The action that minimizes the cost is the most likely function
- Assymetric cost: · If we have more costs for e.g. FP than FN
- Doubtful: pick most likely class only if confident enough
  ↳ $a^* = \begin{cases} y \text{ if } P(y|x) \geq 1-c \\ 0 \text{ otherwise } (c = \text{Don't know}) \end{cases}$

## Uncertainty sampling

- Labels (test/train-data) is expensive
- Always pick the example that we are most uncertain about

1. Estimate $\hat{P}(y|x)$ for all labels
2. Pick most uncertain label: $\underset{i}{\operatorname{argmin}} |0,5 - \hat{P}(y_i|x_i)|$

---

## Generalization error

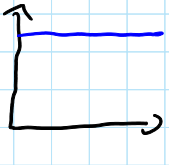
- Difference between the expected and empirical loss
- **Expected loss:** Expected error over all possible values of $(x,y)$ when we know the true distribution
- **Empirical loss:** Empirical error on sample data
  ↳ $\frac{1}{n} \sum R(w x, y)$

---

## ROC and Precision-Recall-curves

- **ROC:**
  - x-axis: false positives
  - y-axis: true positives
  - Used for balanced classes

- **PR:**
  - x: axis: recall
  - y: axis: precission
  - Used when there are class imbalances



## Metrics for classification

- True positive rate: $\frac{TP}{TP+FN} = \frac{TP}{n_+}$ (= Recall)
- False positive rate: $\frac{FP}{TN+FP} = \frac{FP}{n_-}$
- Precision: · How many selected items are relevant
  · $\frac{TP}{TP+FP}$
- Recall: · How many relevant items are selected
  · $\frac{TP}{TP+FN}$

---

## Generative vs. Discrimitative

- Used for classification
- Discrimitative: · Focus on the decision boundary
  · $P(y|x)$
  · Sagt nicht über outliers aus
- Generative: · What kind of positives/negatives are you likely to see
  · $P(x,y) = p(y|x)\,f(x)$
  ↳ How probable is it that x appears (= Prior)

## Maximum Likelihood Estimation

- Point estimate, no representation of uncertainty $\Rightarrow$ MAP
- $\theta_{MLE} = \text{argmax}\ P(D|\theta)$
- Least squares regression

---

## Maximum a Posteriori

- Assumes a joint distribution $p(D, \theta)$
- $p(x_1, \ldots, x_n | \theta)$
- $\theta_{MAP} = \text{argmax}\ P(\theta|D) = \text{argmax}\ P(D|\theta) \cdot p(\theta)$

  ($p(\theta)$: prior distribution)
- If prior distribution is uniform: MAP = MLE
- Allows to model prior belief about parameters
- Ridge Regression

---

## Gaussian Naiv Bayes

- Classification
- $P(x, y) = p(x|y) \cdot P(y) \overset{\text{we assume, that all components are independant}}{=} p(x_1|y) \cdots p(x_d|y) \cdot P(y)$
- For new $x$, predict its $y$

1. Compute probabilities
2. Compute $y = \text{argmax}\ p(y|x) = p(y|\underbrace{x_1, \ldots, x_n}_{\text{components of } x})$

$$= P(y) \prod P(x_i|y)$$

$$P\left(\frac{\text{Banana}}{\text{Long, Sweet, Yellow}}\right) = \frac{P\left(\frac{\text{Long}}{\text{Banana}}\right) \times P\left(\frac{\text{Sweet}}{\text{Banana}}\right) \times P\left(\frac{\text{Yellow}}{\text{Banana}}\right) \times P(\text{Banana})}{P(\text{Long})\, P(\text{Sweet})\, P(\text{Yellow})}$$

} kann weggelassen werden, da es für alle $y$ den gleichen Wert hat.

- If model assumptions are met, GNB will make same predictions as Logistic Regression

## Gaussian Bayes

- we don't assume conditional independance

---

## Fisher's linear discriminant analysis (=Generative)

- Models distribution of predictors separately in each of the response classes
- LDA is like PCA, but it focuses on maximizing the seperability among known categories
- LDA creates a new axes where it can projects the data onto this axis in a way to maximize the separation of the two categories
- Maximize distance between the means of the categories
- Minimize scatter (=variation) of categories

Gaussian Mixture Model (GMM) (Unlabeled clustering)

- K-means: Only mean (= circels)
- Gaussian Bayes classifier: Trained with labels
- GMM: · each cluster is modeled as gaussian distribution with mean and variance (= oval)
  · We can say how probable a data is
  · Each point comes from a cluster with gaussian distribution
- We want to discover the variance/mean of each cluster with EM
  ↳ Do MLE for each cluster to get parameters
- But we don't know which data-point goes where
- Hard EM: · Clusters do not overlap
- Soft EM: · Clusters may overlap

1. Place Gaussians randomly in space
2. For each point figure out to which cluster it is most probable
3. Assign it to all cluster with probabilities
   ↳ Don't do hard assignment like k-means
4. Reestimate clusters

- For partially labeled data:
  - For labeled data: assign directly to corresponding cluster
  - Unlabeled data: Assign to most likely cluster

# Generative Adversarial Networks

- Unsupervised neural network learning
- Two neural nets run against each other
- Discriminator:
  - Has to decide wheter the input data is from the real data set or generated from the generator
- Generator:
  - Generates data from random input noise
  - Learns the distribution of the real data-set
- Like a game between D and G