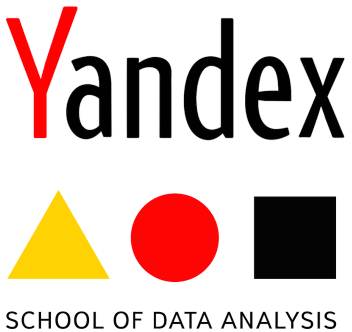
Machine Learning in Science and Industry

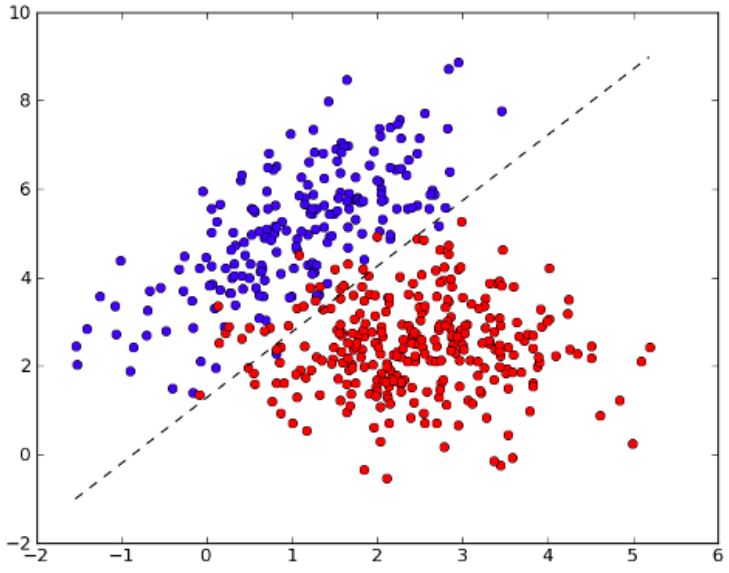
Day 3, lectures 5 & 6

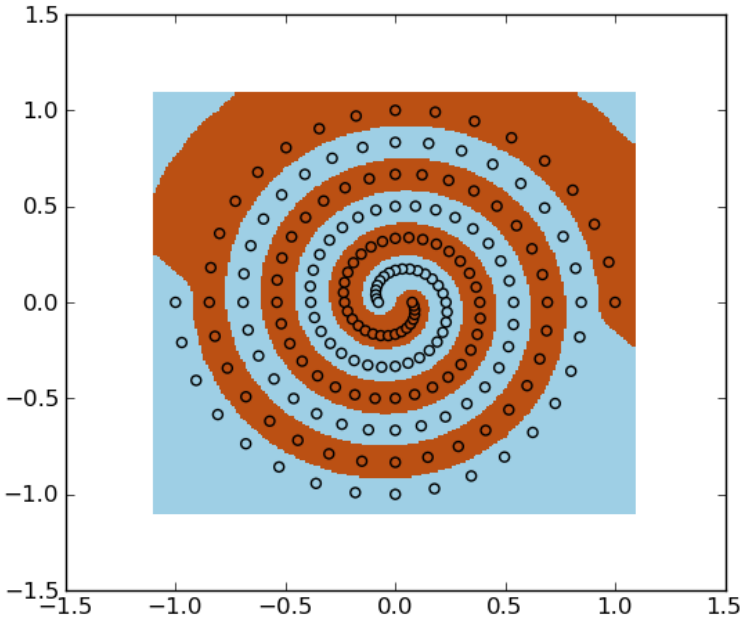
Alex Rogozhnikov, Tatiana Likhomanenko

Heidelberg, GradDays 2017



Today

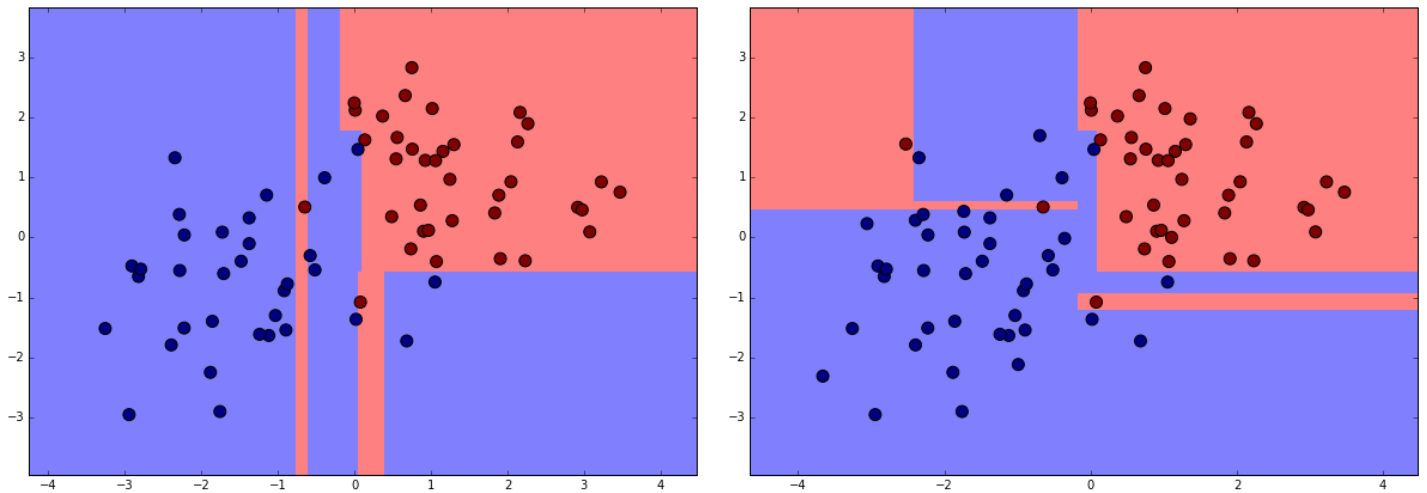




- it is linear boundary on the pictures
- how recommendations are working
- working with texts and detecting spam

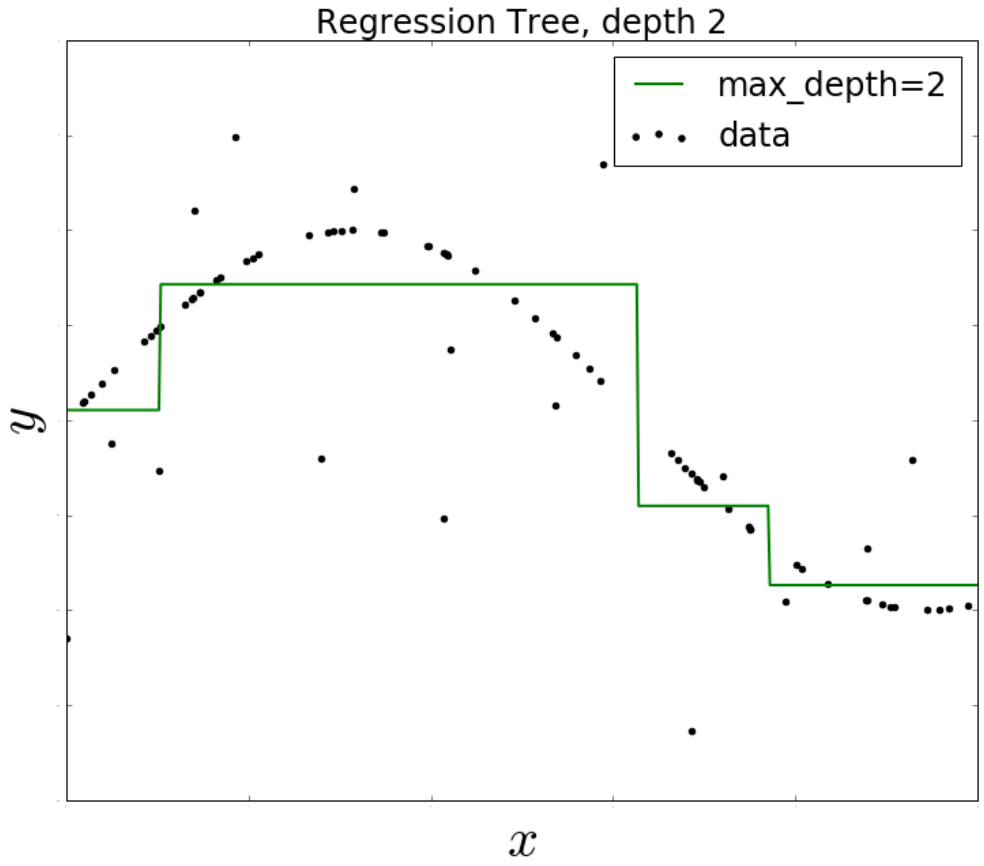
Recapitulation: decision trees for classification

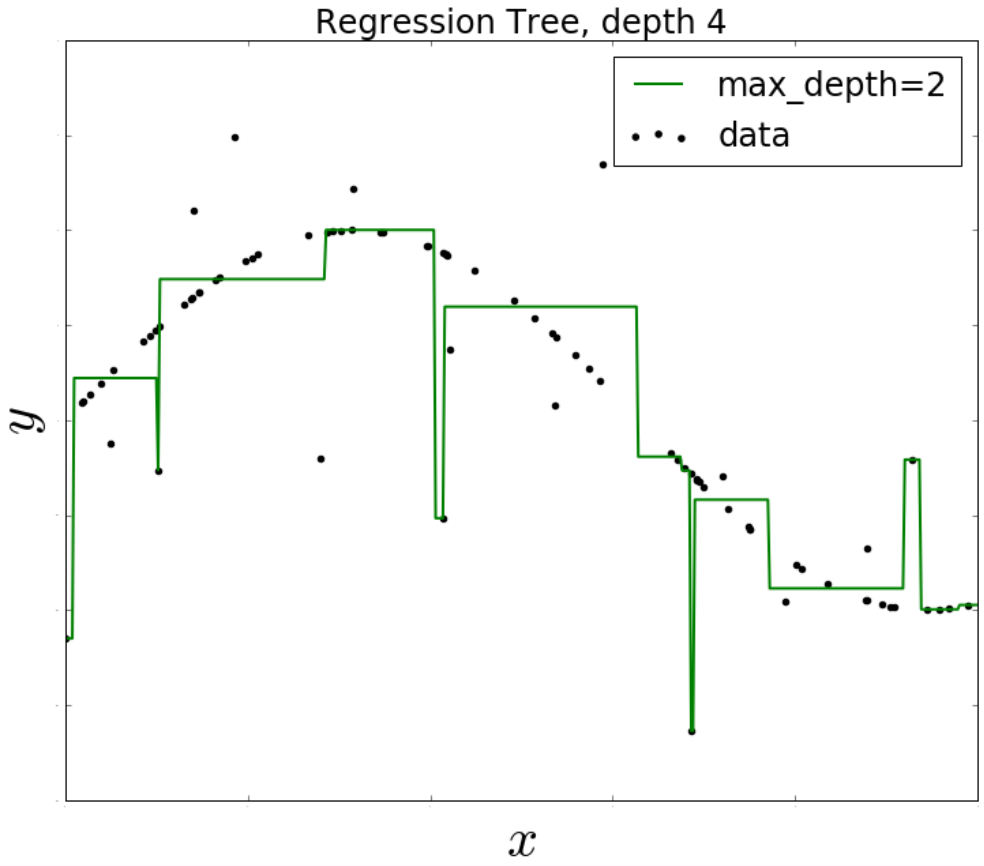
- built greedily starting from the root
- some criterion is used to select best splitting
- unstable to modifications in data
- use pre-stopping to prevent overfitting
- works with data of different nature, scale-invariant



Recapitulation: decision trees for regression

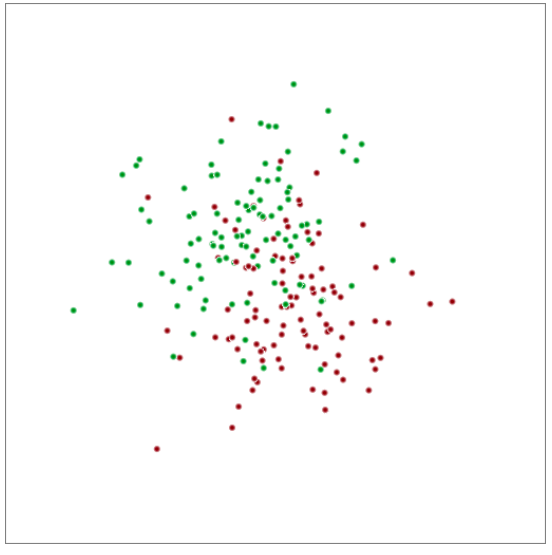
- the only difference is that it minimizes different criterion: MSE

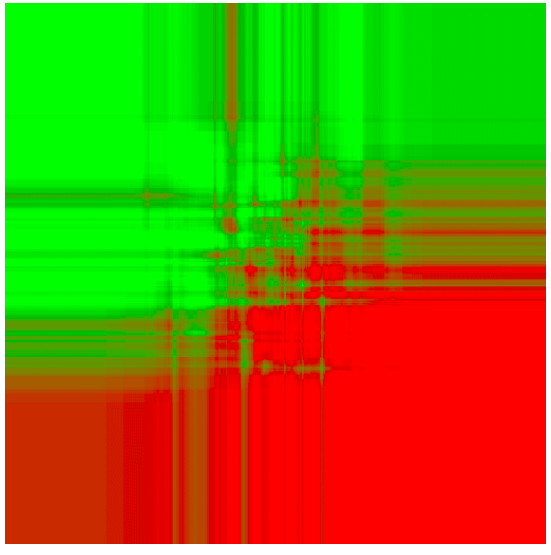




Recapitulation: Random Forest

- build many independent trees
- each tree is overfitted
- to make trees different, each tree uses part of the data and part of the features



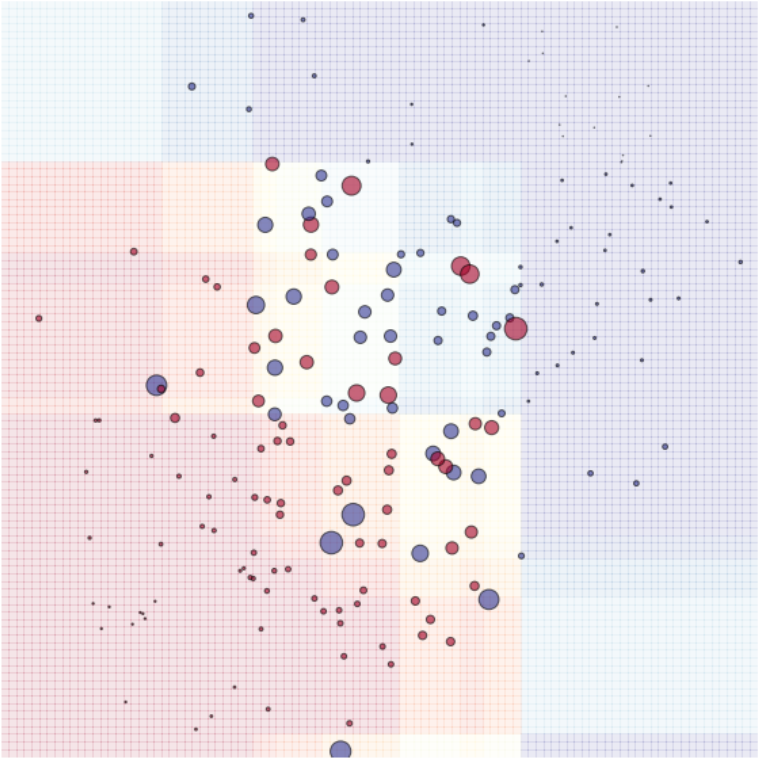


- the ensemble never overfits
  - holdout quality doesn't go down when increasing number of trees
- previous mistakes not taken into account
- considered as "i-don't-want-to-tune" classifier

Recapitulation: AdaBoost

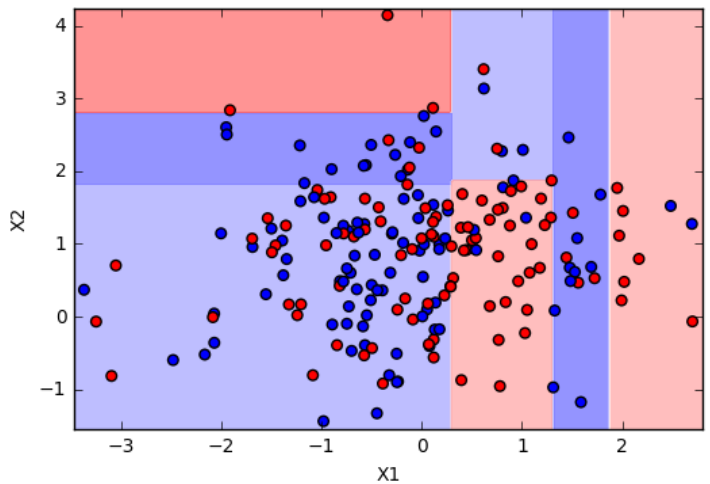
- observations are given sample weights w_i
- trains a sequence of estimators (i.e. trees)
- if event was misclassified, it's weight is increased
  - otherwise weight is decreased

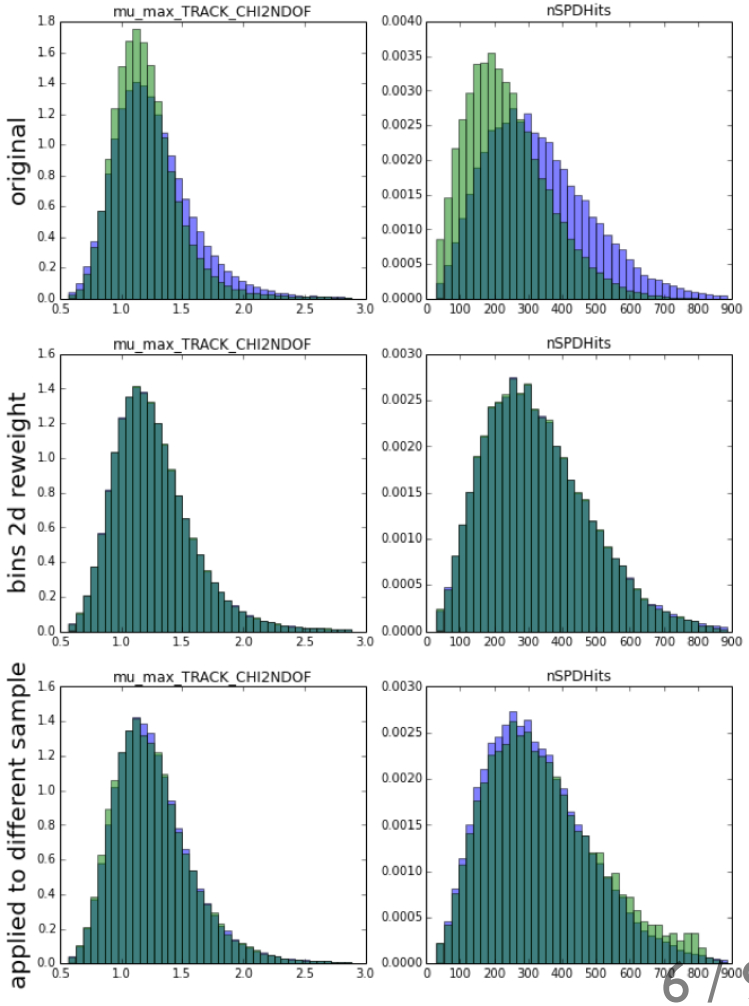
- greedy optimization of exponential loss in the core
- efficiently combines many weak learners



Recapitulation: reweighting with boosting

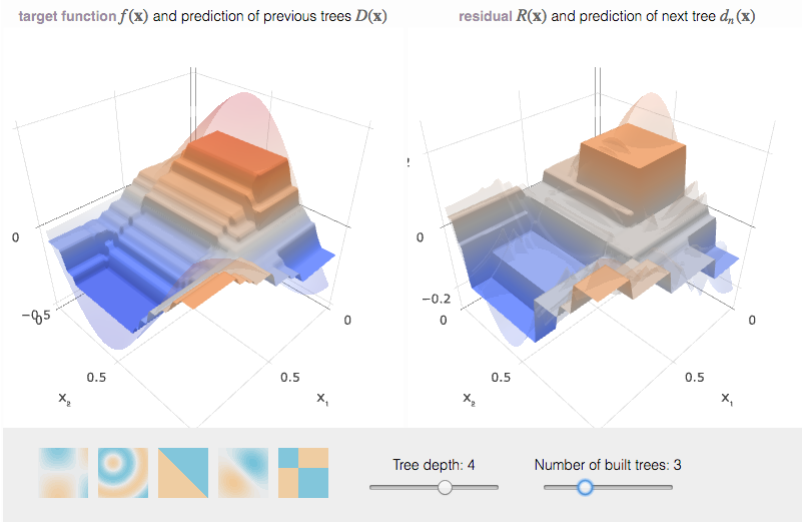
- use binning of the space provided by a tree
- tree searches for splitting that provides regions with high misbalance between two distributions
- if in the bin amount of MC is less then amount of data, increase weights of MC
  - decrease otherwise





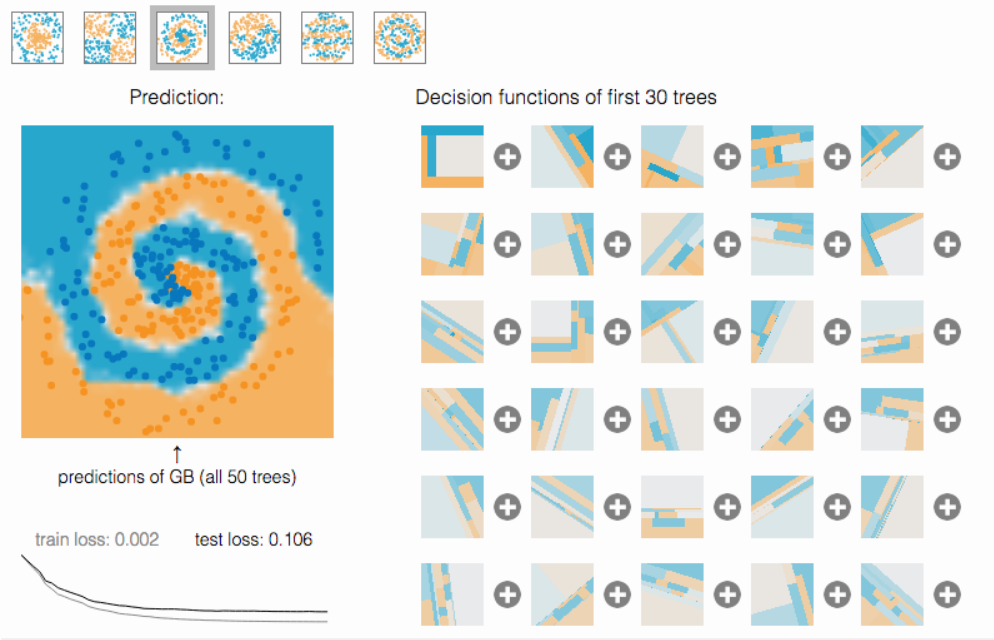
Recapitulation: Gradient Boosting

- sequentially builds trees to minimize some loss
- each new tree approximately follows the gradient of loss
- loss can be selected accordingly to different problems
  - classification
  - regression
  - ranking
  - uniform classification
- efficiently works with tabular data of different nature
- widely used and requires tuning



Recapitulation: GB tricks

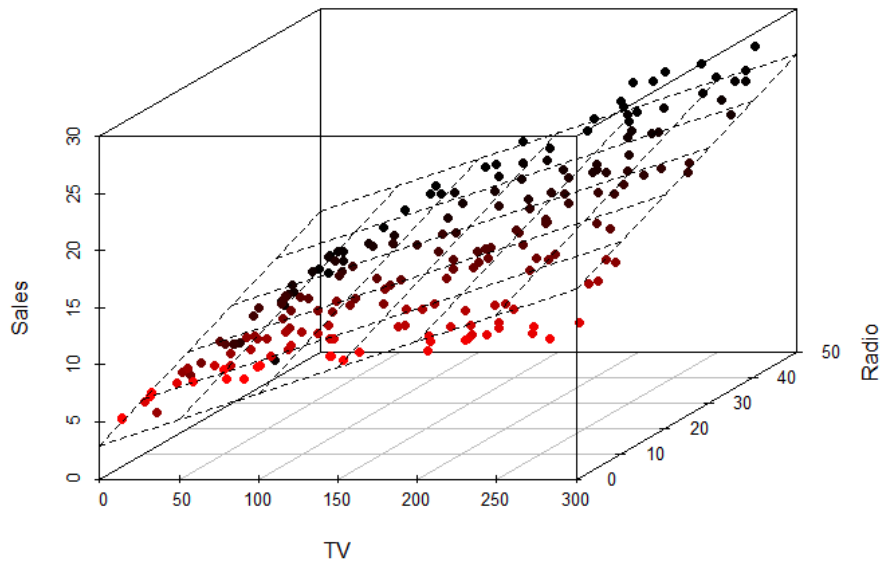
- randomization of trees to avoid duplicates
  - subsampling of features
  - subsampling of training data
- smaller step (learning rate) to make optimization more stable and get less noisy residual
- can overfit
  - monitoring of quality vs number of trees helps

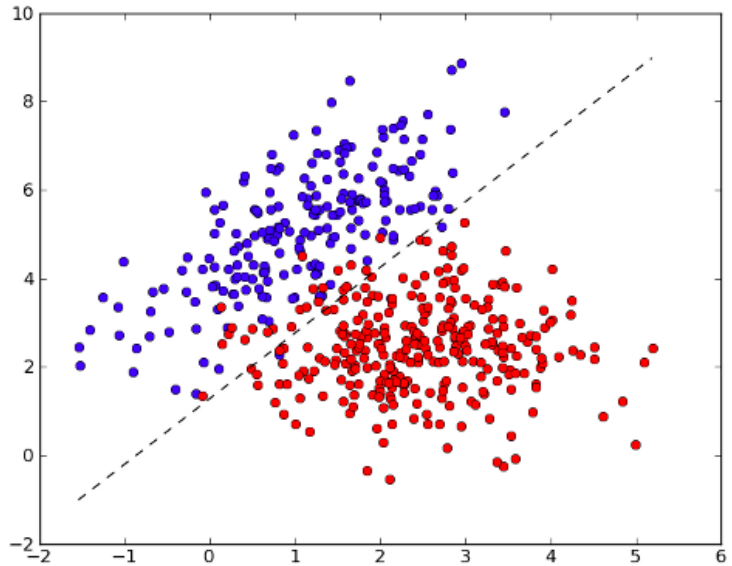


Back to linear models

Linear decision rule: $d(x) =< w, x >= \sum_j w_j x_j$

- used for regression with minimizing MSE
- used for classification with minimizing LogLoss





Polynomial decision rule

$$d(x) = w_0 + \sum_j w_j x_j + \sum_{j,j'} w_{jj'}\, x_j x_{j'}$$

Polynomial decision rule

$$d(x) = w_0 + \sum_j w_j x_j + \sum_{j,j'} w_{jj'}\, x_j x_{j'}$$

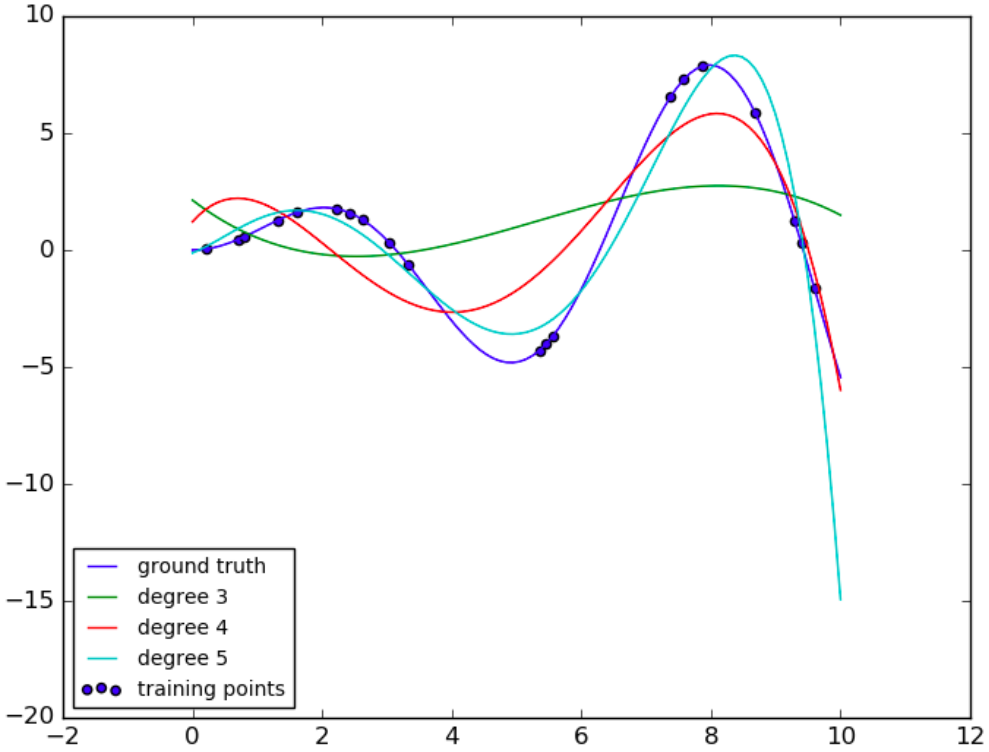
is again a linear model, introduce an extended set of features:

$$z = \{1\} \cup \{x_j\}_j \cup \{x_j x_{j'}\}_{j,j'}$$

$$d(x) = \sum_k w_k z_k =< w, z >$$

and reuse logistic regression.

Polynomial regression



is done in the same way.

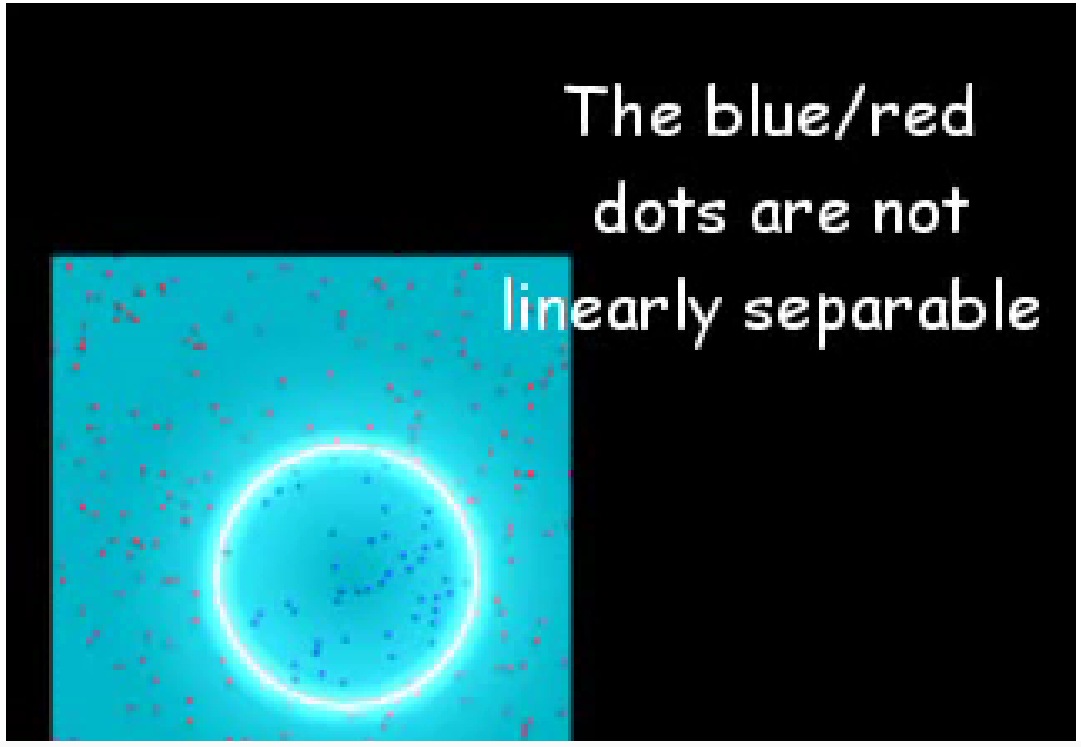
E.g. to fit the polynomial of one variable, we construct for each observation a vector of

$$z = (1, x, x^2, x^3, \dots x^d)$$

and train a linear regression

$$d(x) =< w, z >= w_0 + w_1 x + w_2 x^2 + \dots w_d x^d$$

Projecting into the space of higher dimension



0:04 / 0:42

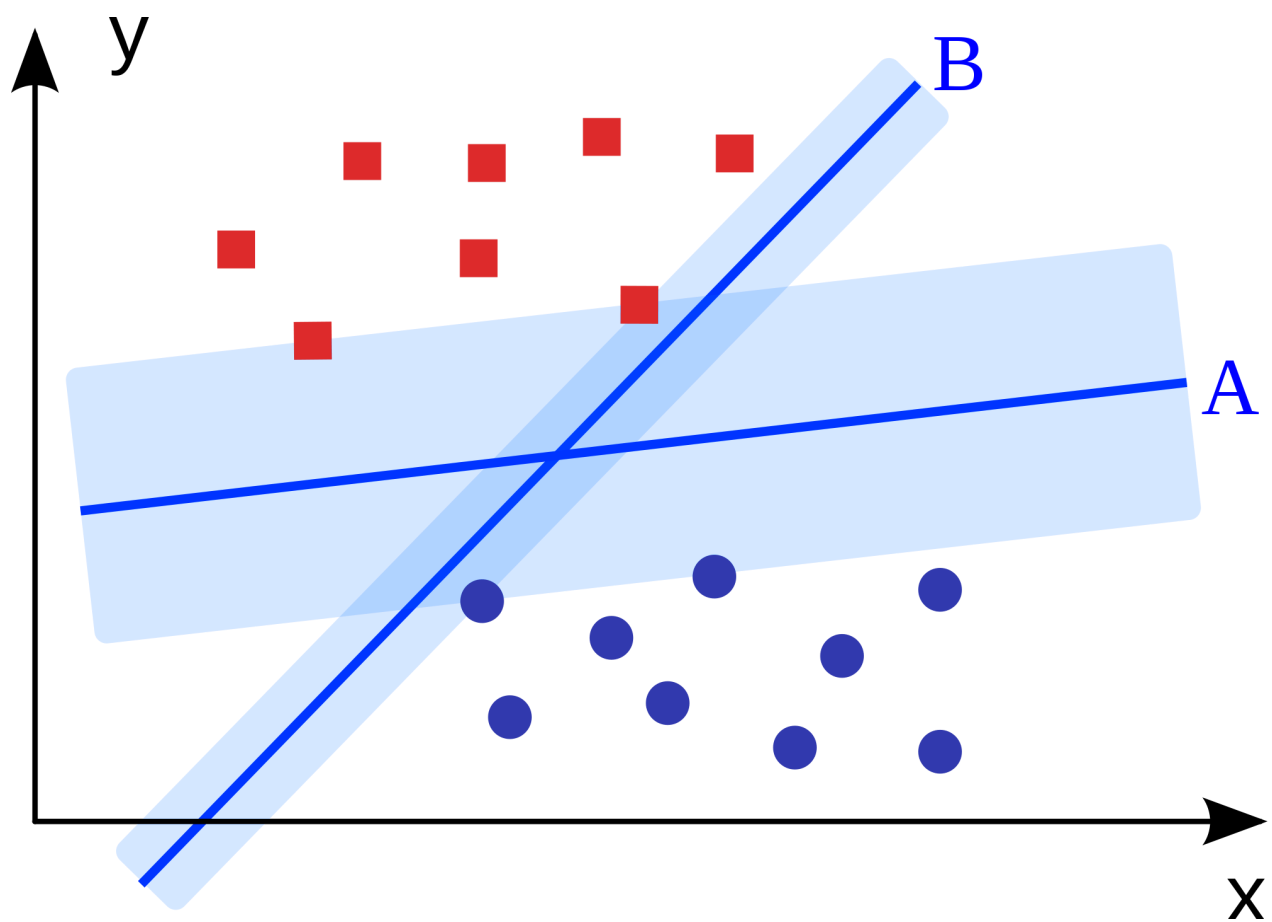
After adding new features, classes may become separable.

Logistic regression overview

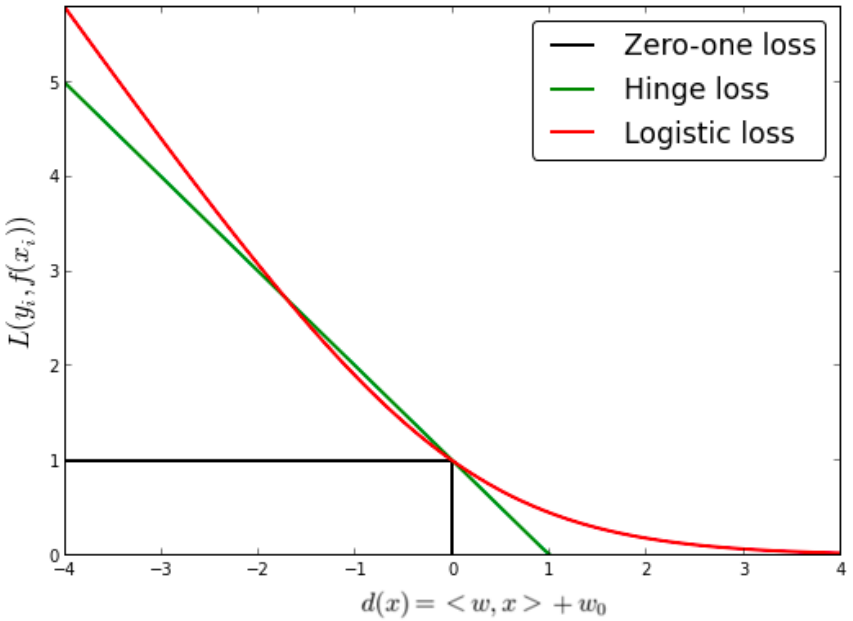
- classifier based on linear decision rule
- training is reduced to convex optimization
- can handle > 1000 features, but requires regularization
- no interaction between features presents in the model
  - other decision rules are achieved by adding new features

Support Vector Machine [Vapnik, Chervonenkis, 1963]

SVM selects a decision rule with maximal possible margin (rule A).



Hinge loss function



SVM uses different loss function:

$$L_{\text{hinge}}(x_i, y_i) = \max(0, 1 - y_i d(x_i))$$

$$L_{\text{logloss}}(x_i, y_i) = \log(1 + e^{-y_i d(x_i)})$$

Margin $y_i d(x_i) > 1 \rightarrow$ no penalty.
(corresponds to being on the right side of the margin)

Only losses for positive class compared on the plot.

Kernel trick

P is a projection operator (which "adds new features").

$$d(x) = < w, x > \quad \rightarrow \quad d(x) = < w, P(x) >_{\text{new}}$$

Assume that optimal $w = \sum_i \alpha_i P(x_i)$ (combination of support vectors) and look for α_i

$$d(x) = \sum_i \alpha_i < P(x_i), P(x) >_{\text{new}} = \sum_i \alpha_i K(x_i, x)$$

We need only *kernel*, not a projection operator:

$$K(x, \tilde{x}) = < P(x), P(\tilde{x}) >_{\text{new}}$$

Kernel trick

Polynomial kernel:

$$K(x, \tilde{x}) = (1 + \tilde{x}^T x)^m$$

projection contains all monomials up to degree m.

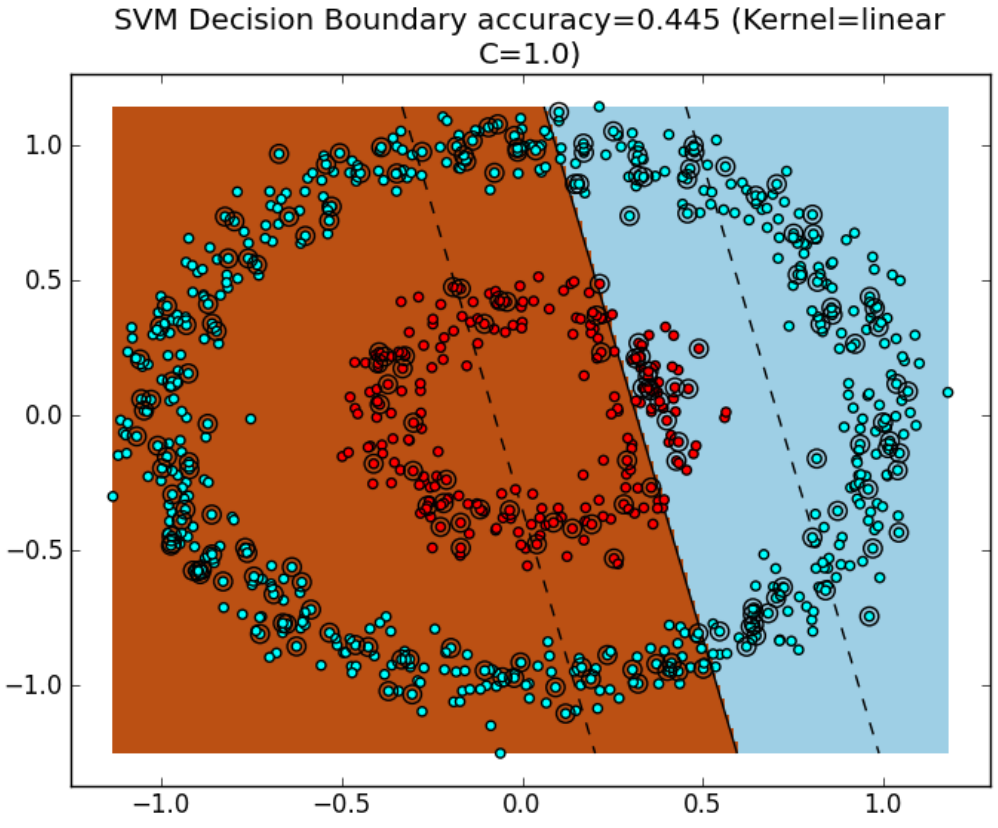
Popular kernel is a gaussian Radial Basis Function (RBF):

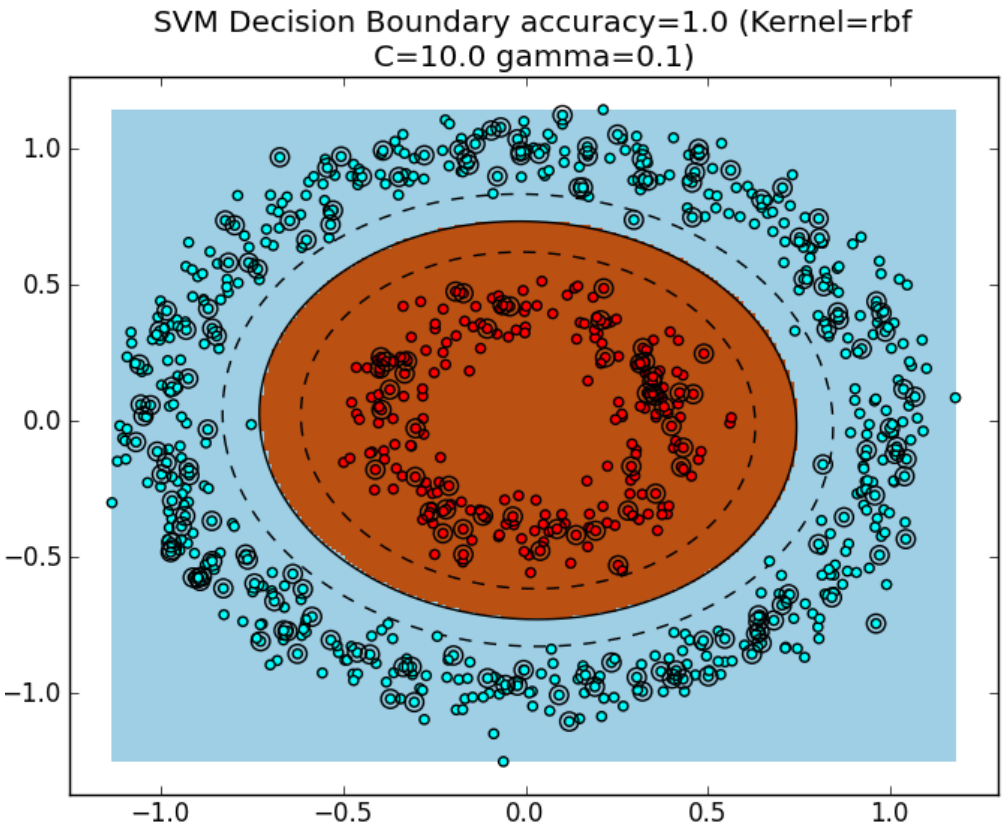
$$K(x, \tilde{x}) = e^{-\gamma ||x - \tilde{x}||^2}$$

Corresponds to projection to the Hilbert space.

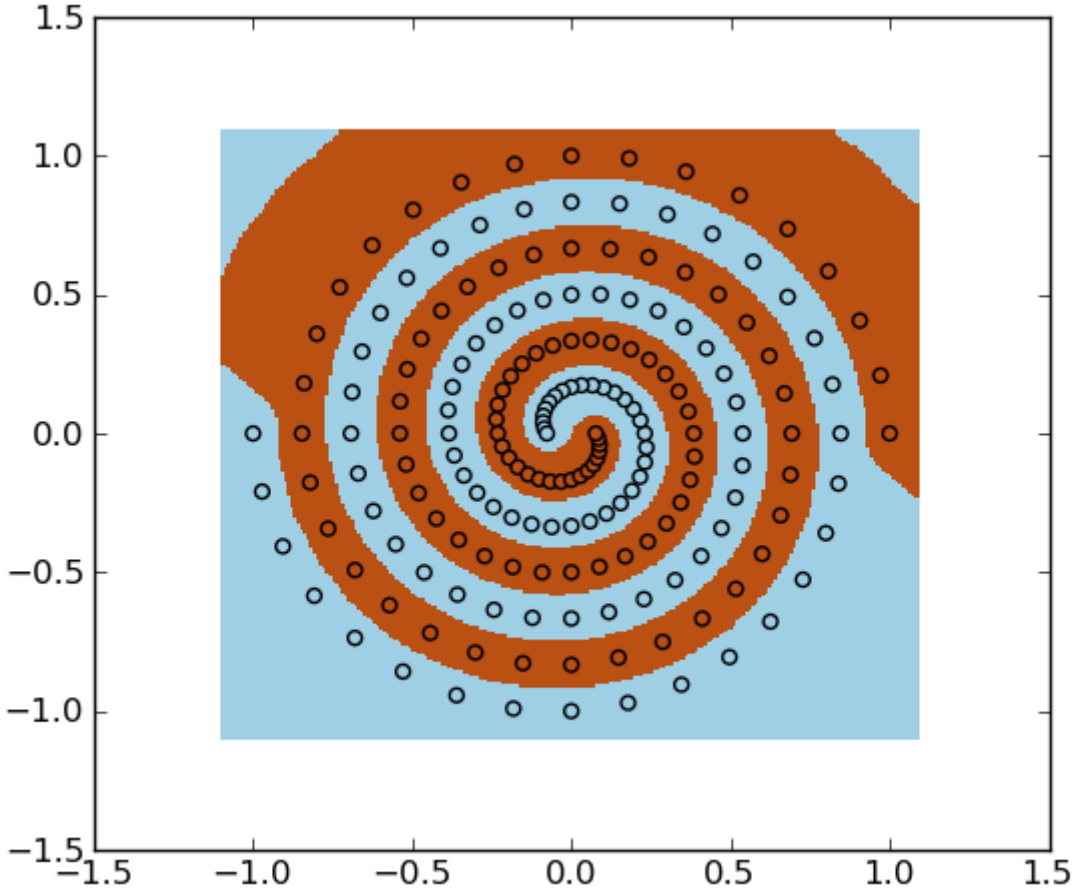
Exercise: find a corresponding projection for RBF.

SVM + RBF kernel

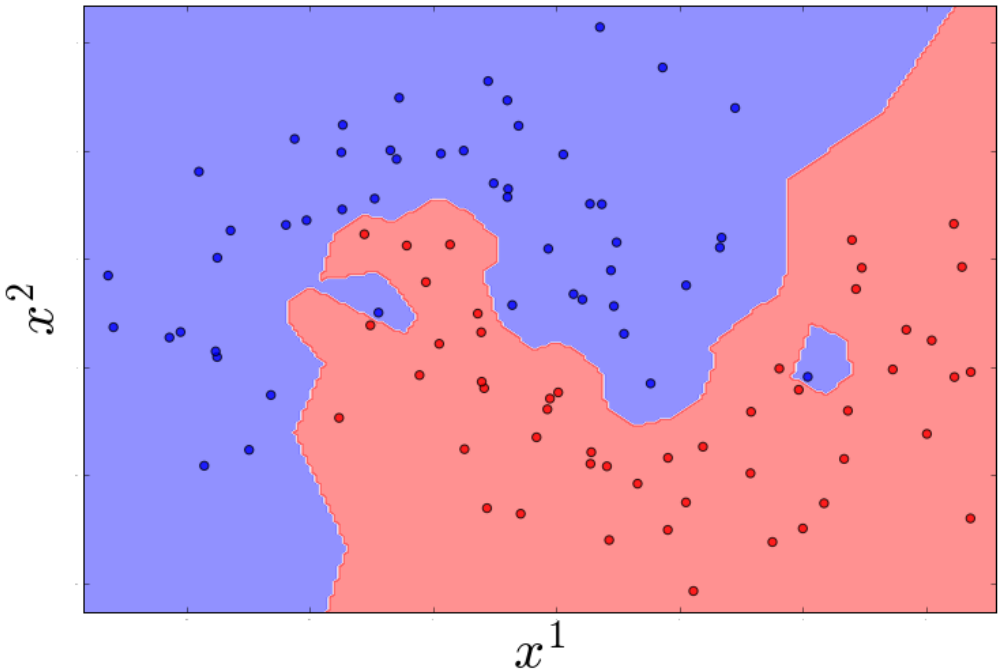


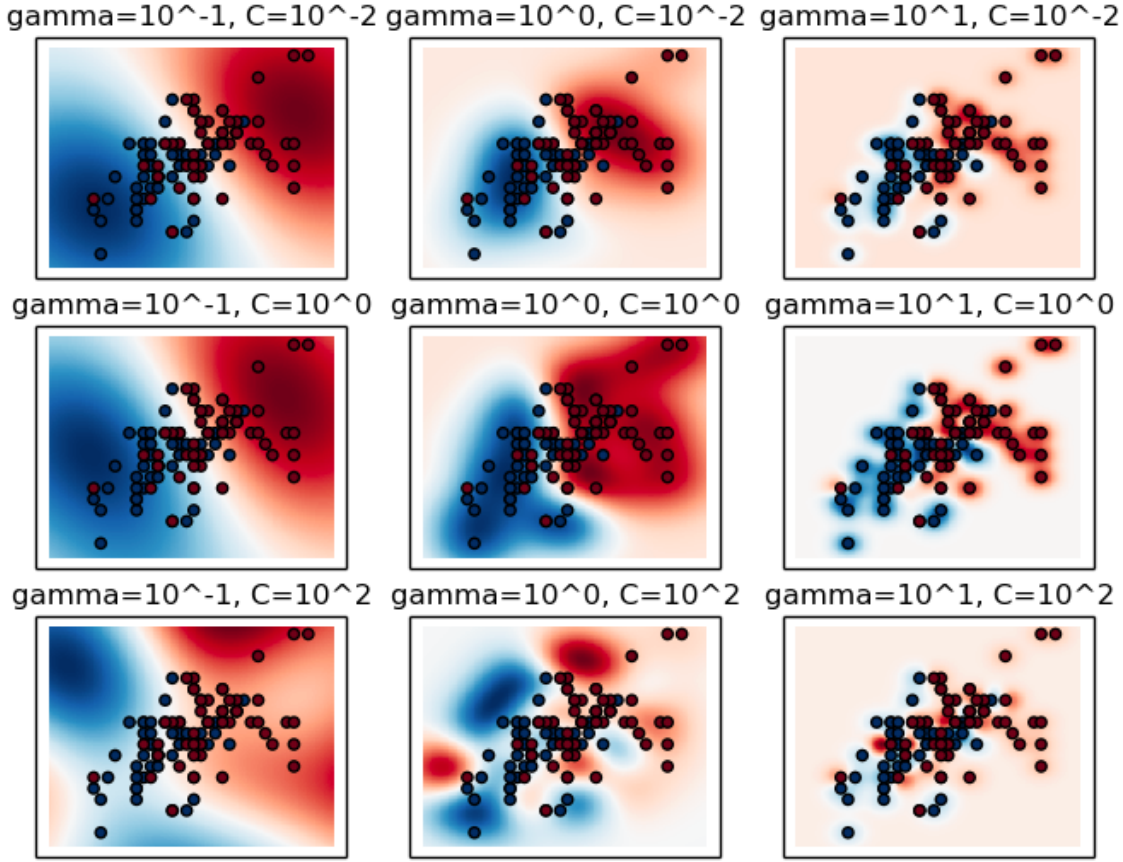


SVM + RBF kernel



Overfitting

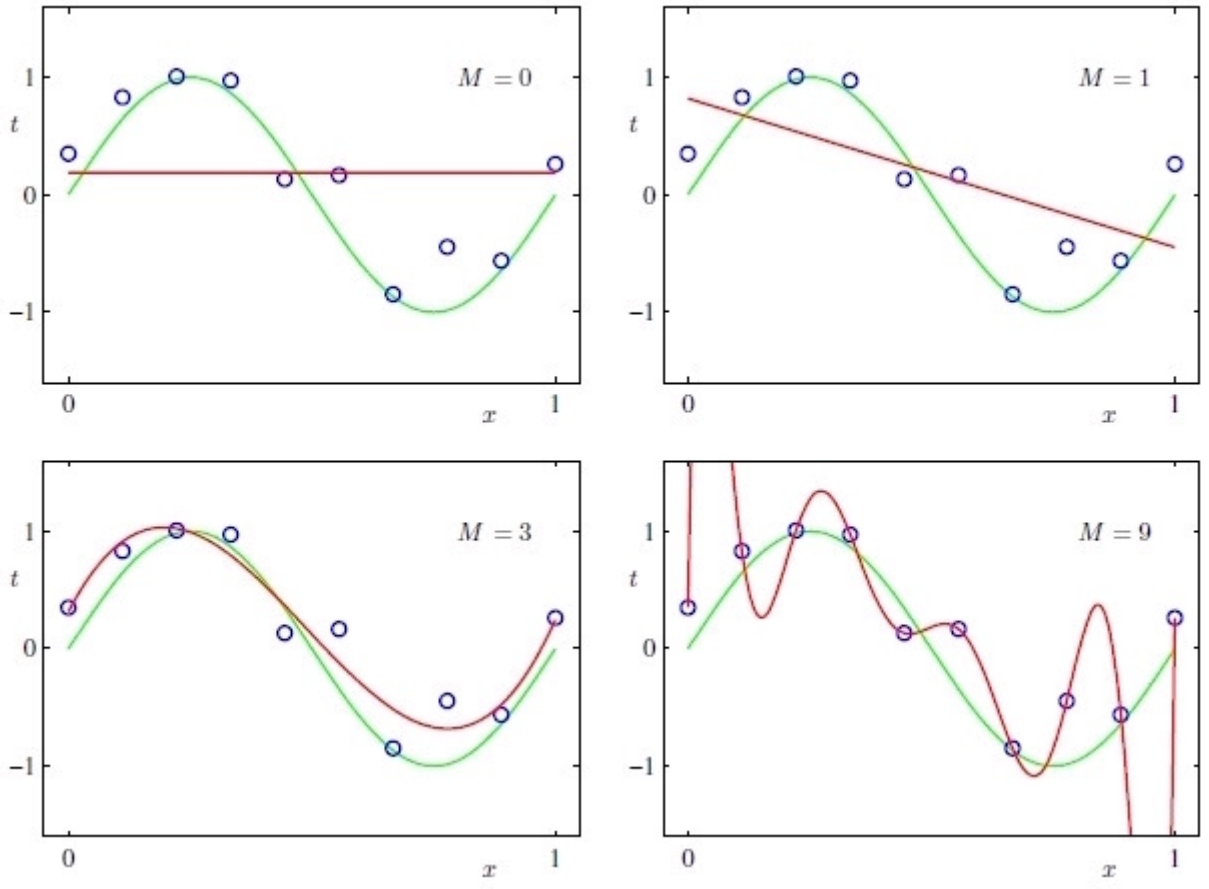




$k$nn with k=1 gives ideal classification of training data.

SVM with small radius of RBF kernel has the same property.

Overfitting



Same issues for regression.

Provided high enough degree, the polynomial can go through any set of points and get zero error this way.

Reminder: model selection

select model which gives better results on new data!

SVM and regularization

Width of margin is $\frac{1}{||w||}$, so SVM loss is actually:

$$\mathcal{L} = \frac{1}{2C}||w||^2 + \sum_i L_{\text{hinge}}(x_i, y_i)$$

- first term is maximizing a margin
- second term penalizes samples that are not on the correct side of the margin
- C is controlling the trade-off

Linear models summary

- linear decision function in the core
- reduced to nice optimization problems
- losses are additive

$$\mathcal{L} = \sum_i L(x_i, y_i)$$

- can support nonlinear decisions w.r.t. to original features by using kernels
- apply regularizations to avoid bad situations and overfitting (poor results on new data)

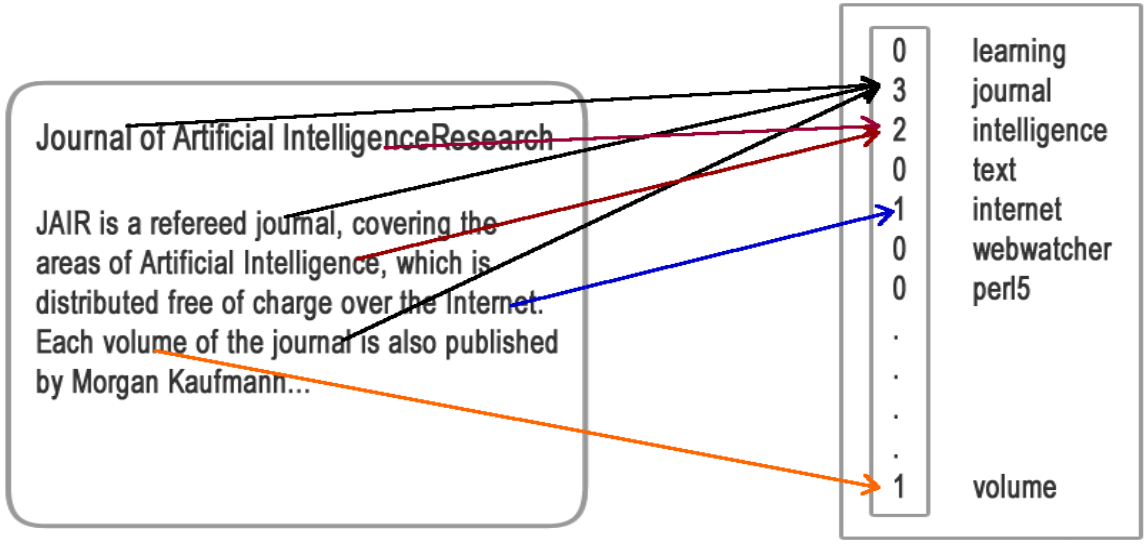
Linear models applications

Provide a very good quality in problems with many features and/or very few labeled observations.

- spam classification.
  - Bag-of-words / n-grams + SVM were a standard for a long time
  - still good for SMS spam
- opinion mining
  - estimate how people liked / disliked new cinema / show / book / product / etc.
- basic advertisement systems
- earlier linear models were very popular in computer vision, in particular SVM+RBF

Bag of words

- simple way of feature extraction for texts
- each column in data = number of occurrences of some word
- ignores the order of words



This method produces lots of features, and linear models are used (with regularizations) to 'select' appropriate words.

Dear friend,
I am Mrs. Sese-seko widow of late President Mobutu Sese-seko of Zaire, now known as Democratic Republic of Congo (DRC)...

spammish words like 'Congo', 'president', 'widow' are detected quite trivially.

Question: how to fool such anti-spam system?

Personalized search engine

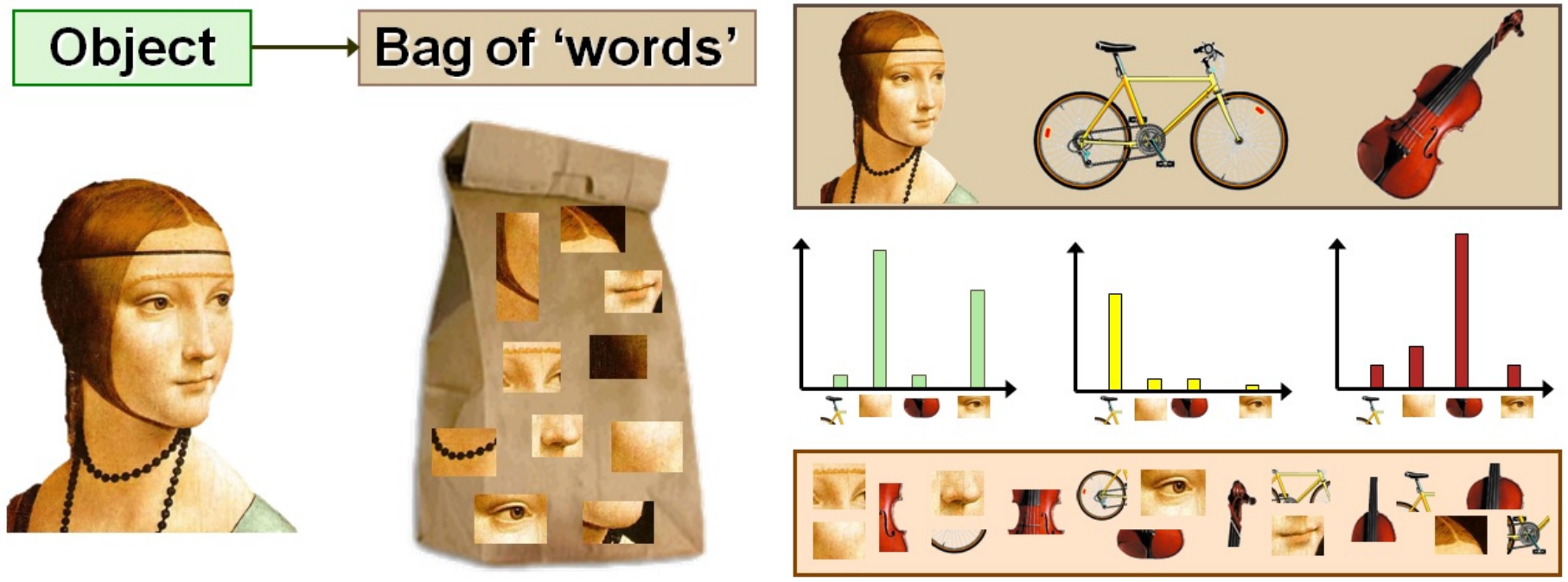
Linear regression provides very strong baseline in a system for customers searches, specially coupled with text preprocessing and linguistic methods.

Finds very specific pages with very high *recall* (= found relevant / all relevant documents):

- collect all scientific papers about antibiotic resistance
- collect web-pages of unlicensed money lenders

Scale: 1000-10000 results from billions of pages.

- an expert is suggested to start from pointing several search queries and marking pages as relevant or not
- system dynamically brings new pages for judgement
- assessing several hundreds of document is enough



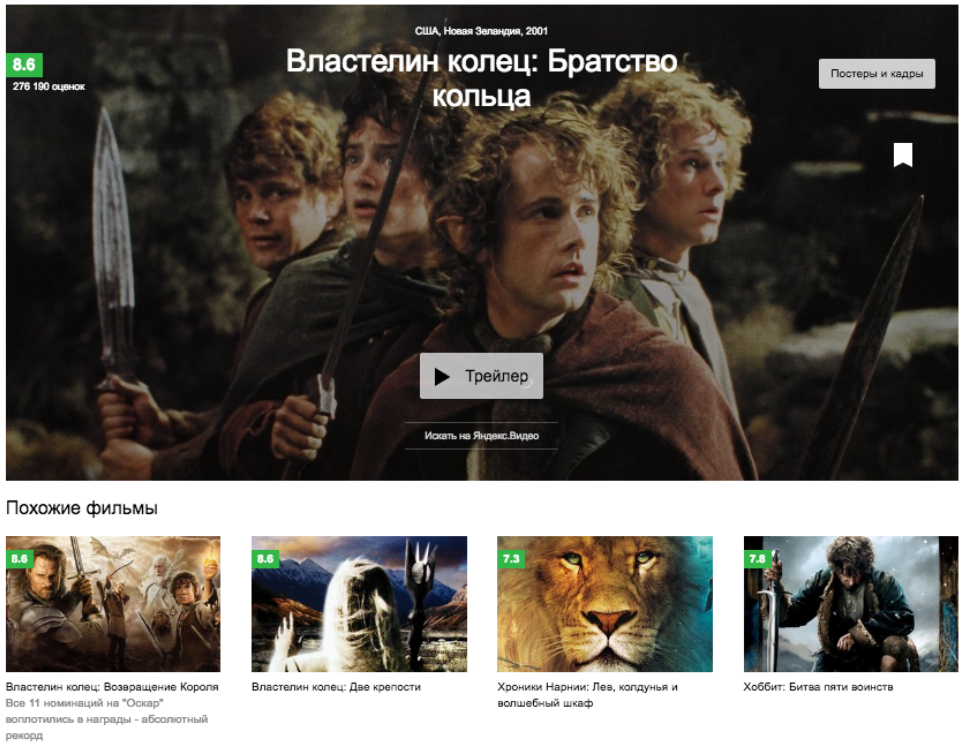
Quite similar approach to bag-of-words used to find very similar images.
Also it was used for a long time for image recognition.

Small image patches are considered as 'words', and represented as vectors (e.g. with SIFT descriptors), the space of patches was clustered (e.g. with k-means).

Image is represented as a histogram: how many patches were there in each cluster.

Factorization models

Motivation: recommending



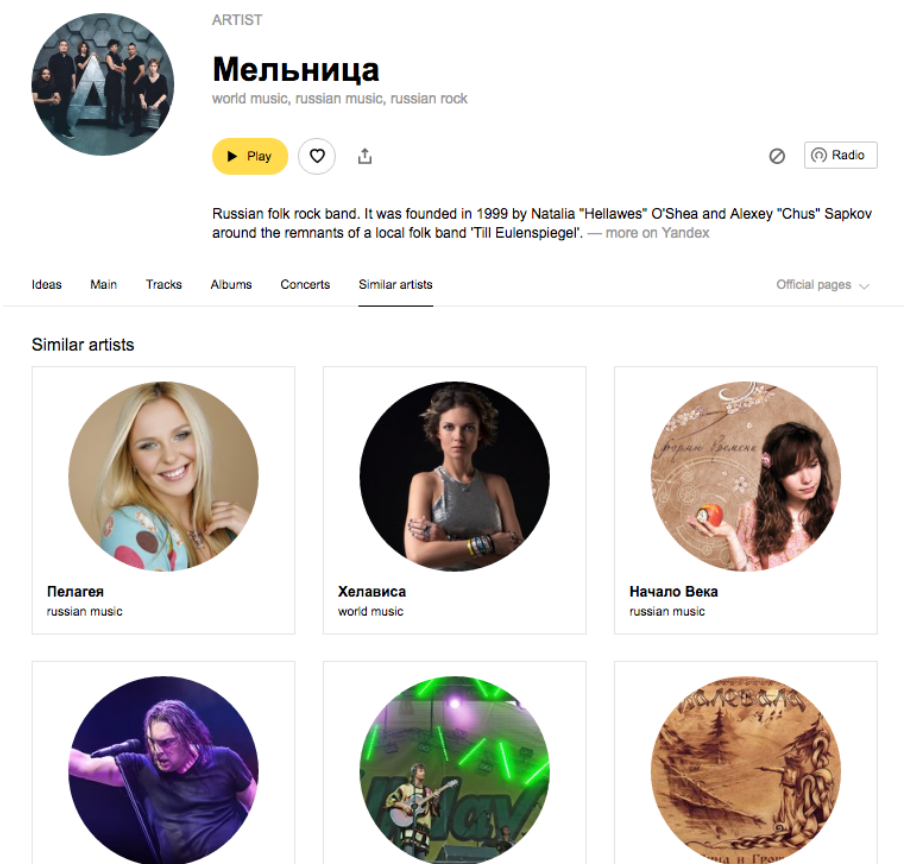
Images from [kinopoisk]

Problems typically arising in systems with lots of content:

- group items by similarity
- show similar artists, actors, music tracks, movies, books, blogs, goods etc.
- personalized recommendations, which encounter previous history

This should be done efficiently without irritating user with lots of requests.

Problem of finding an appropriate feature space



Images from Yandex.Music

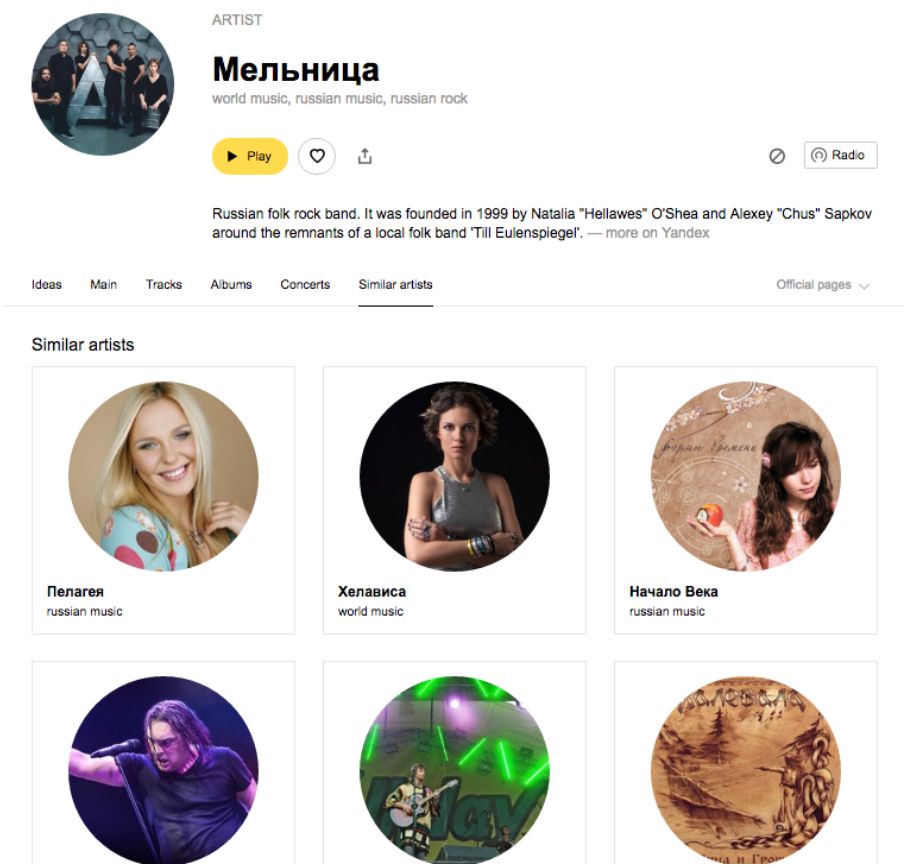
Linear models provide a good basis for a personalized recommender system.

Regularization make it possible to work with very little feedback (even < 10 items).

But there is no appropriate feature space. Example for songs:

- presence of word in title / lyrics?

Problem of finding an appropriate feature space



Images from Yandex.Music

Linear models provide a good basis for a personalized recommender system.

Regularization make it possible to work with very little feedback (even < 10 items).

But there is no appropriate feature space. Example for songs:

- presence of word in title / lyrics?
- one-hot of artist?

Factorization models

[image source]

Users' feedback can be considered as a huge matrix of size $N_{\text{users}} \times N_{\text{items}}$ with missing items.
Available numbers are sparsely distributed.

Our goal is to be able to predict any missing element.

Factorization models

[image source]

Users' feedback can be considered as a huge matrix of size $N_{\text{users}} \times N_{\text{items}}$ with missing items.
Available numbers are sparsely distributed.

Our goal is to be able to predict any missing element.

- Assumption: this matrix *is low-rank*
- thus, it can be decomposed (factorized):

$$\hat{R} = UV$$

$$\hat{R}_{u,i} = \sum_{f=1}^{k} U_{u,f} V_{f,i}$$

Matrix factorization

U of size $N_{\text{users}} \times k$, V of size $k \times N_{\text{items}}$.

Parameters (elements of U and V) are obtained by minimizing

$$\sum_{\text{known } R_{u,i}} \left[\hat{R}_{u,i} - R_{u,i}\right]^2 \to \min \qquad \hat{R}_{u,i} = \sum_{f=1}^{k} U_{u,f} V_{f,i} =< U_{u,}.V_{.,i} >$$

Number of parameters is quite reasonable:

- for each user a vector $U_{u,.}$ of size k is found
- for each item a vector $V_{.,i}$ of size k is found

Note: for the user model is linear, for the item too!
Factorization infers appropriate feature space *automatically*.

Interpretation of factorization

Factors (vector elements at position f) may become interpretable:

- dub-step that is popular in India / folk music
- movies with Leonardo DiCaprio / horror film
- science fiction book

$$U_{user,\cdot} = \begin{bmatrix} 0.7 \\ 0.4 \\ -0.6 \end{bmatrix} \begin{matrix} \text{user loves} \\ \leftarrow \text{fiction?} \\ \leftarrow \text{romance?} \\ \leftarrow \text{drama?} \end{matrix} \qquad V_{\cdot,item} = \begin{bmatrix} 0.6 \\ -0.5 \\ 0.6 \end{bmatrix} \begin{matrix} \text{this book is like} \\ \leftarrow \text{fiction?} \\ \leftarrow \text{romance?} \\ \leftarrow \text{drama?} \end{matrix}$$

In this example for books $\hat{R}_{user,item} = 0.7 \times 0.6 - 0.4 \times 0.5 - 0.6 \times 0.6 = -0.14$

Similar books get similar vectors!

Better recommendations

Regularizations make it possible to work with little feedback and get sensible recommendations.

Much more information can be incorporated in the model:

- implicit feedback
    - click history (internet shops use this frequently)
    - skipping of tracks in music

Better recommendations

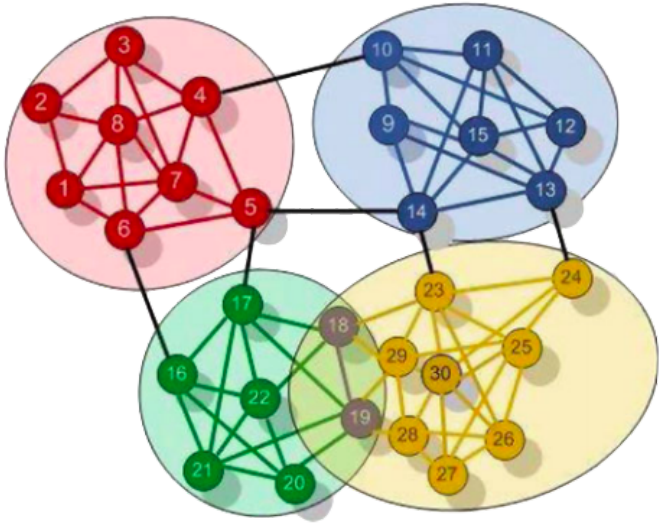
Regularizations make it possible to work with little feedback and get sensible recommendations.

Much more information can be incorporated in the model:

- implicit feedback
  - click history (internet shops use this frequently)
  - skipping of tracks in music
- incorporating basic features like age / country / etc.
  - that's tensor factorizations methods implemented in Factorization Machines, Vowpal Wabbit
- friends' likes (in social networks)
  - e.g. by saying that vectors $U_{user1,.}$ and $U_{user2,.}$ should be closer

Other applications of factorization models

Factorization models reveal internal properties of interacting entities



- a matrix of mutual friendship / likes / messages can be factorized to discover communities
- Spam detection in social network
- Drug-target interaction in medicine
- Source separation in sound
  - for instance, for noise speech removal
- Document clustering

Factorization Machine (Rendle, 2010)

FM is a generalization of this idea. Starting from a quadratic model:

$$y = w_0 + \sum_j w_j x_j + \sum_{j>j'} w_{j,j'} x_j x'_j$$

assume that interaction of features have low rank $w_{j',j} =< v_j, v_{j'} >$:

$$y = w_0 + \sum_j w_j x_j + \sum_{j'>j} < v_j, v_{j'} > x_j x'_j$$

Again, we use the same approaches for setting a machine learning problem.

This model is useful when there are many features and reconstruction of a matrix $w_{j',j}$ is not feasible.

n-minutes break

Recapitulation

- linear models
  - adding new features
  - kernels
  - SVM
  - bag-of words + linear models to process texts

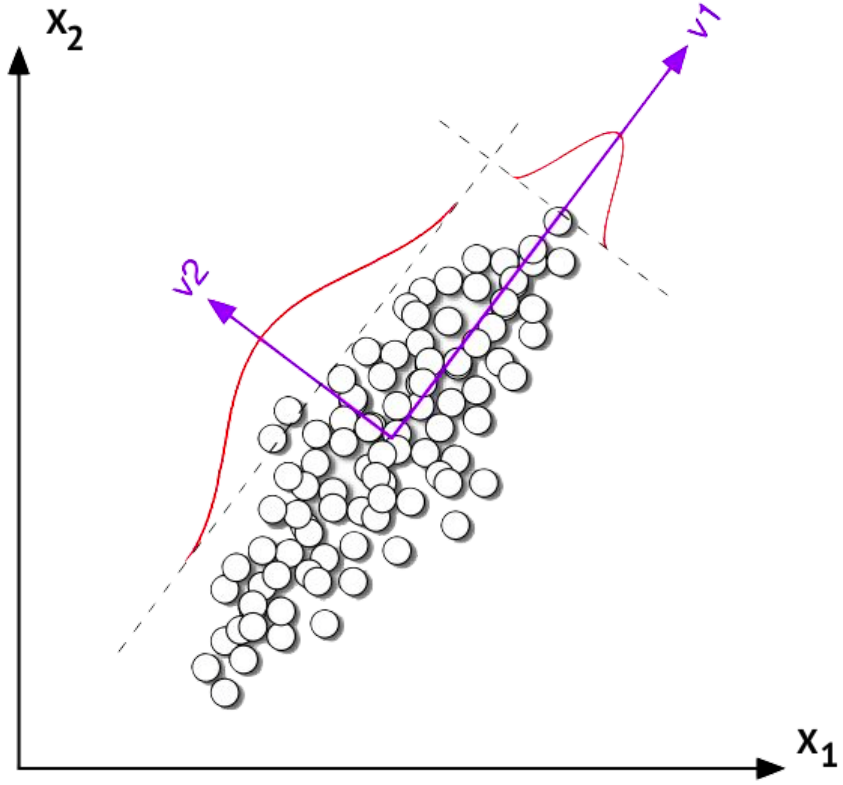
- factorization models
  - recommendations
  - 'understanding things though interaction'

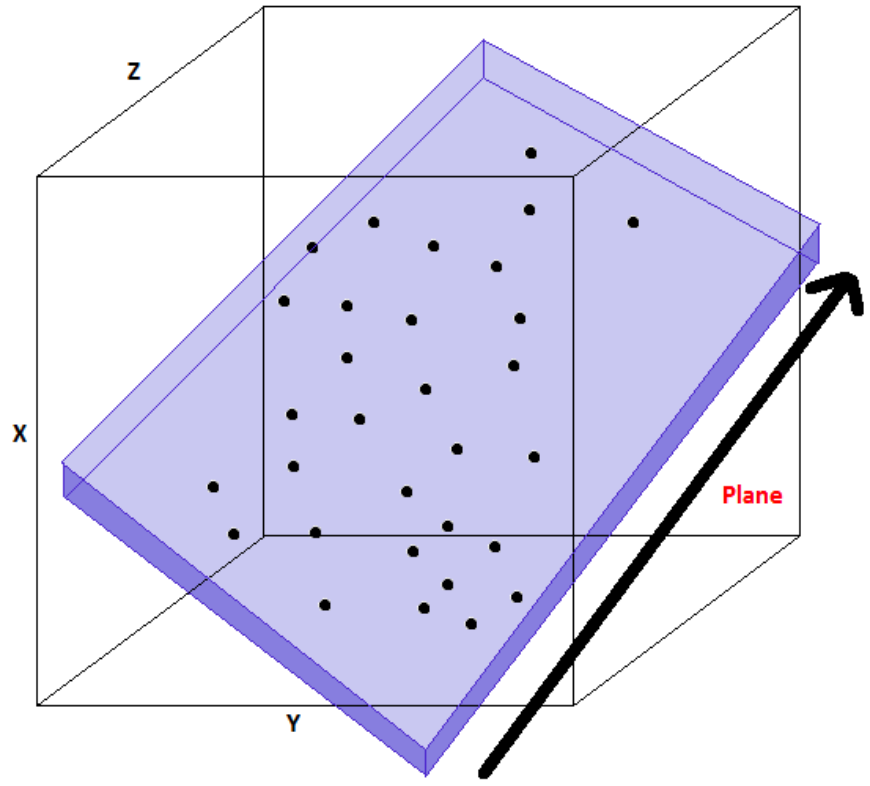
Unsupervised dimensionality reduction

a way to extract less features, which are *probably* more useful and easier to operate with

Principal component analysis [Pearson, 1901]

PCA is finding axes along which variance is maximal





PCA description

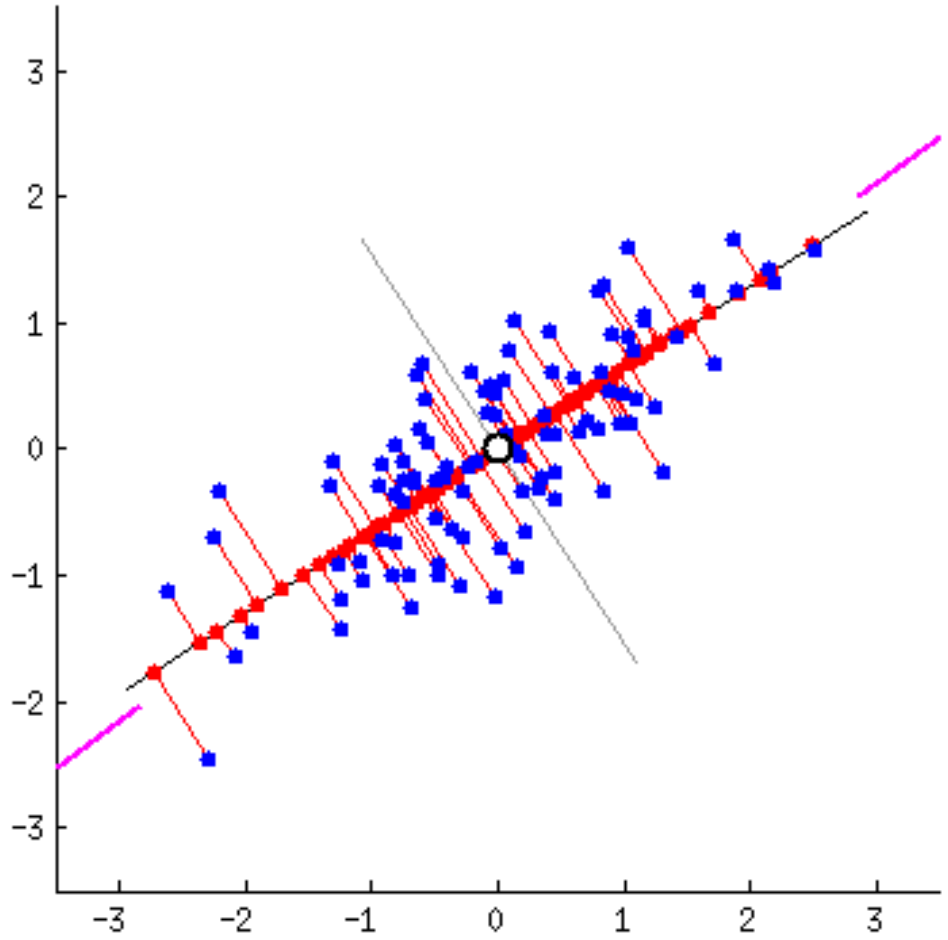
PCA is based on the principal axis theorem

$$Q = U^T \Lambda U$$

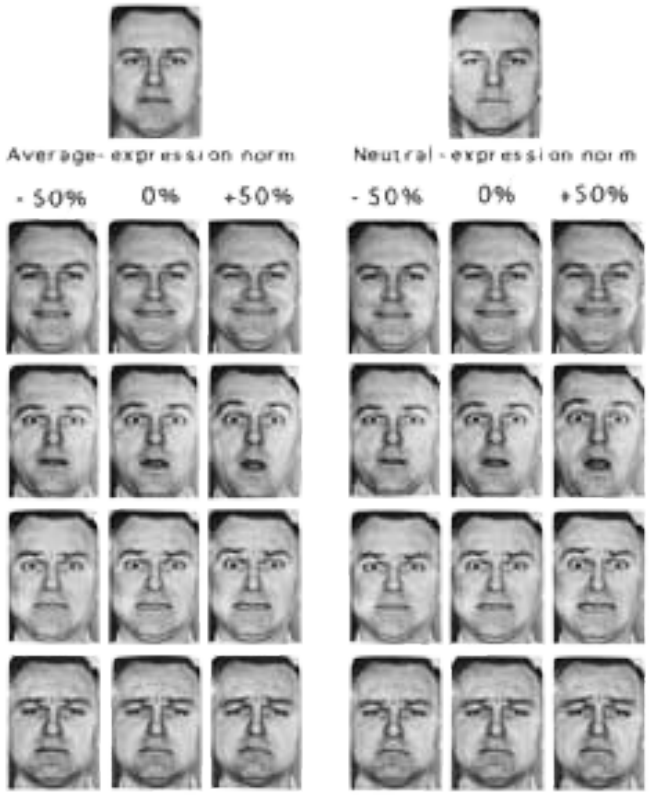
Q is covariance matrix of the dataset, U is orthogonal matrix, Λ is diagonal matrix.

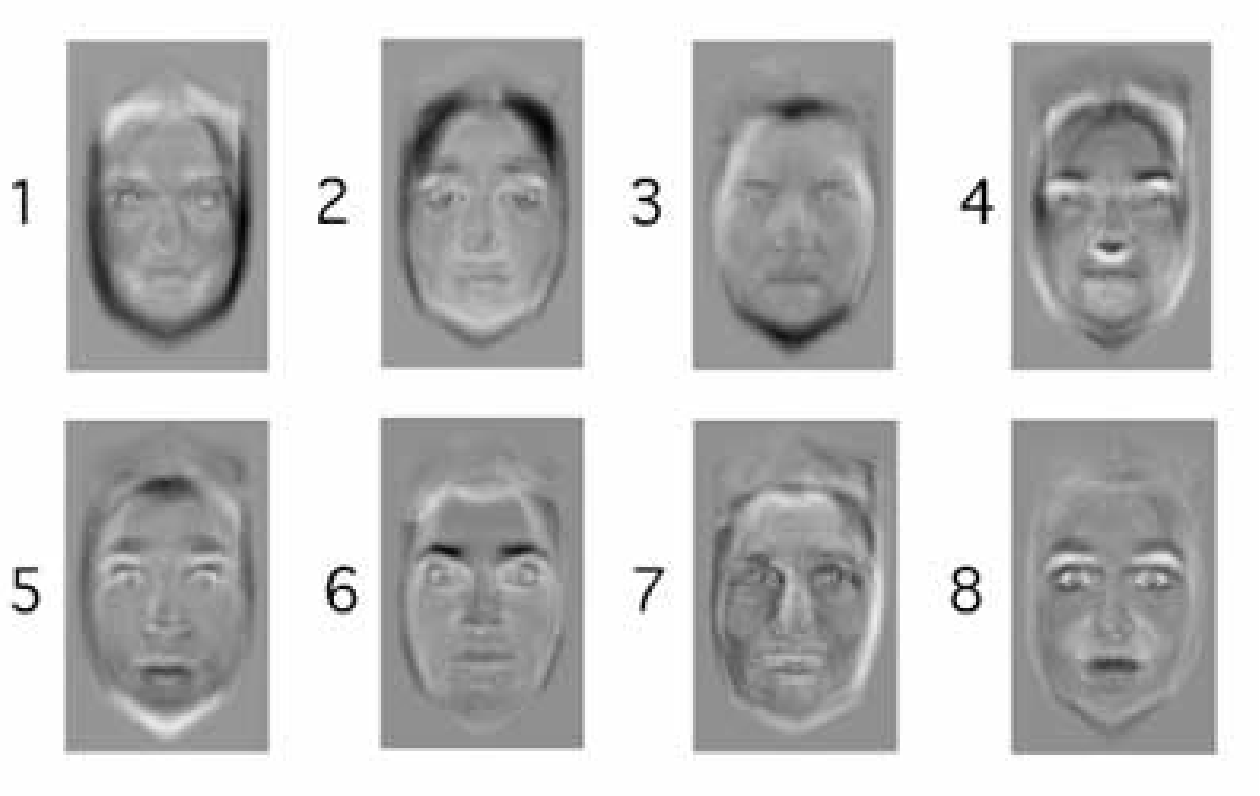
$$\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_n), \qquad \lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$$

PCA optimization visualized (animation source)



PCA: eigenfaces





$$\text{Face} = \alpha[\text{scared}] + \beta[\text{laughs}] + \gamma[\text{angry}] + \ldots$$

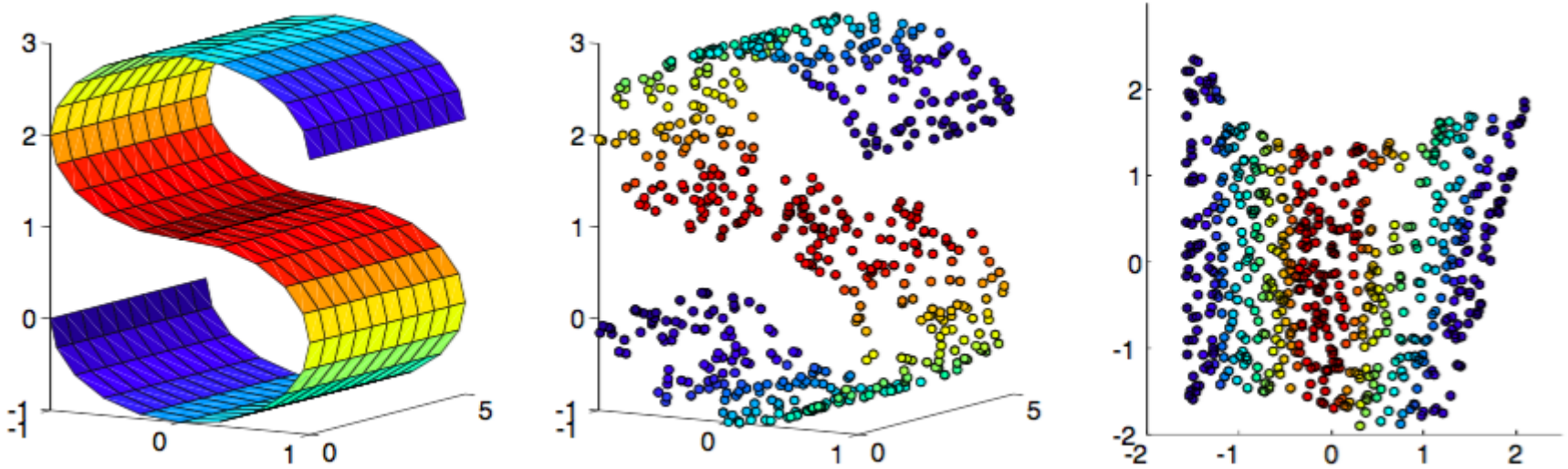
PCA

- simple way to describe jets shapes or shapes of object in telescope data
- Typically applied to high-dimensional arrays with measurements of the same nature (pictures, sounds, temperature maps, gene expressions)
  Copy-paste from wikipedia:

> Depending on the field of application, it is also named the discrete Kosambi-Karhunen–Loève transform (KLT) in signal processing, the Hotelling transform in multivariate quality control, proper orthogonal decomposition (POD) in mechanical engineering, singular value decomposition (SVD) of X (Golub and Van Loan, 1983), eigenvalue decomposition (EVD) of XTX in linear algebra, factor analysis (for a discussion of the differences between PCA and factor analysis see Ch. 7 of [3]), Eckart–Young theorem (Harman, 1960), or Schmidt–Mirsky theorem in psychometrics, empirical orthogonal functions (EOF) in meteorological science, empirical eigenfunction decomposition (Sirovich, 1987), empirical component analysis (Lorenz, 1956), quasiharmonic modes (Brooks et al., 1988), spectral decomposition in noise and vibration, and empirical modal analysis in structural dynamics.

Locally linear embedding

handles the case of non-linear dimensionality reduction



Express each sample as a *convex combination* of neighbours $\sum_i |x_i - \sum_{\tilde{i}} w_{i\tilde{i}} x_{\tilde{i}}| \to \min_w$

Locally linear embedding

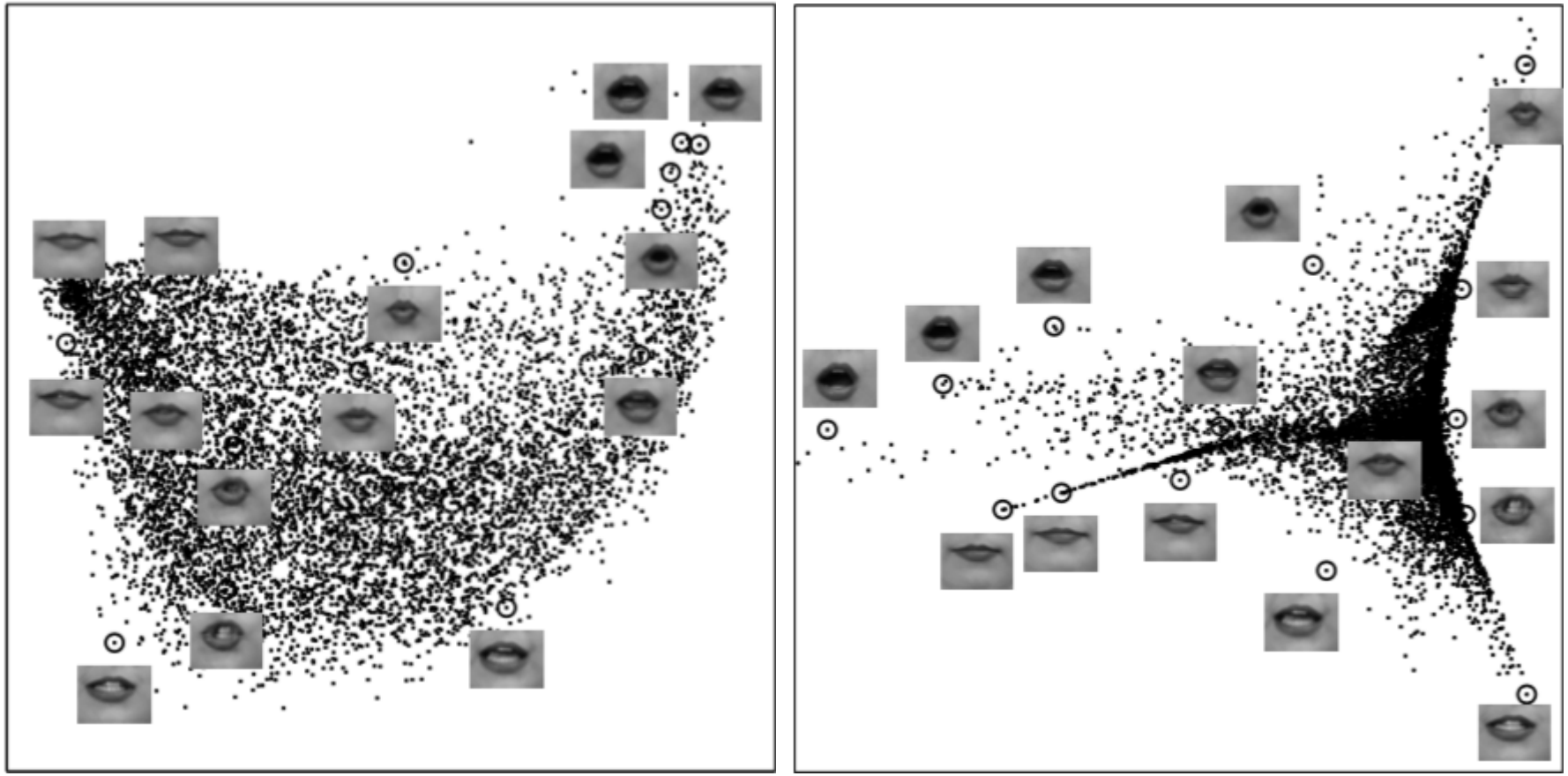
$$\sum_i \left| x_i - \sum_{\tilde{i}} w_{i\tilde{i}} x_{\tilde{i}} \right| \to \min_w$$

subject to constraints: $\sum_{\tilde{i}} w_{i\tilde{i}} = 1$, $w_{i\tilde{i}} >= 0$, and $w_{i\tilde{i}} = 0$ if i, $\tilde{i}$ are not neighbors.

Finding an optimal mapping for all points simultaneously (y_i are images — positions in the new space):

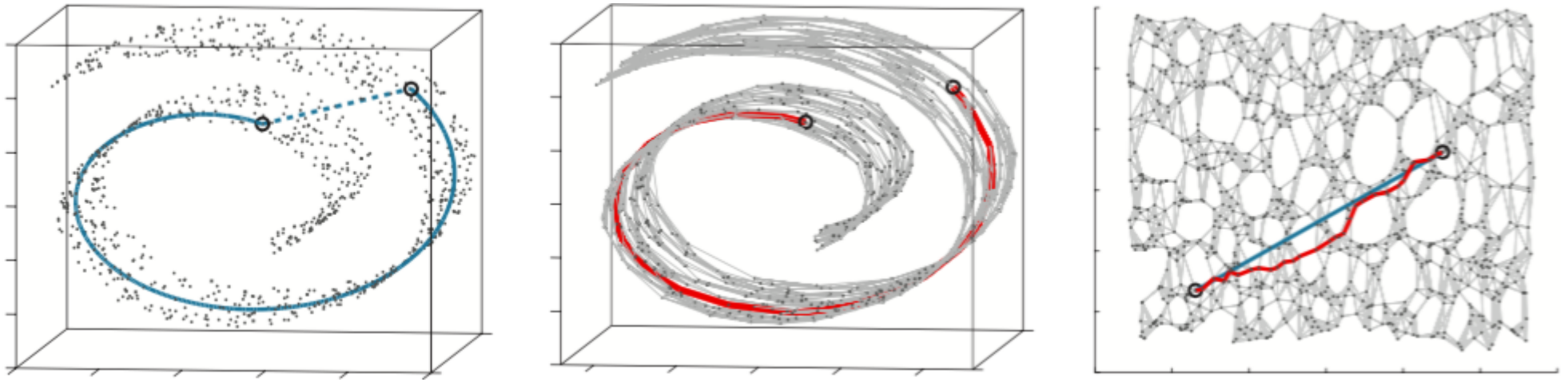
$$\sum_i \left| y_i - \sum_{\tilde{i}} w_{i\tilde{i}} y_{\tilde{i}} \right| \to \min_y$$

PCA and LLE



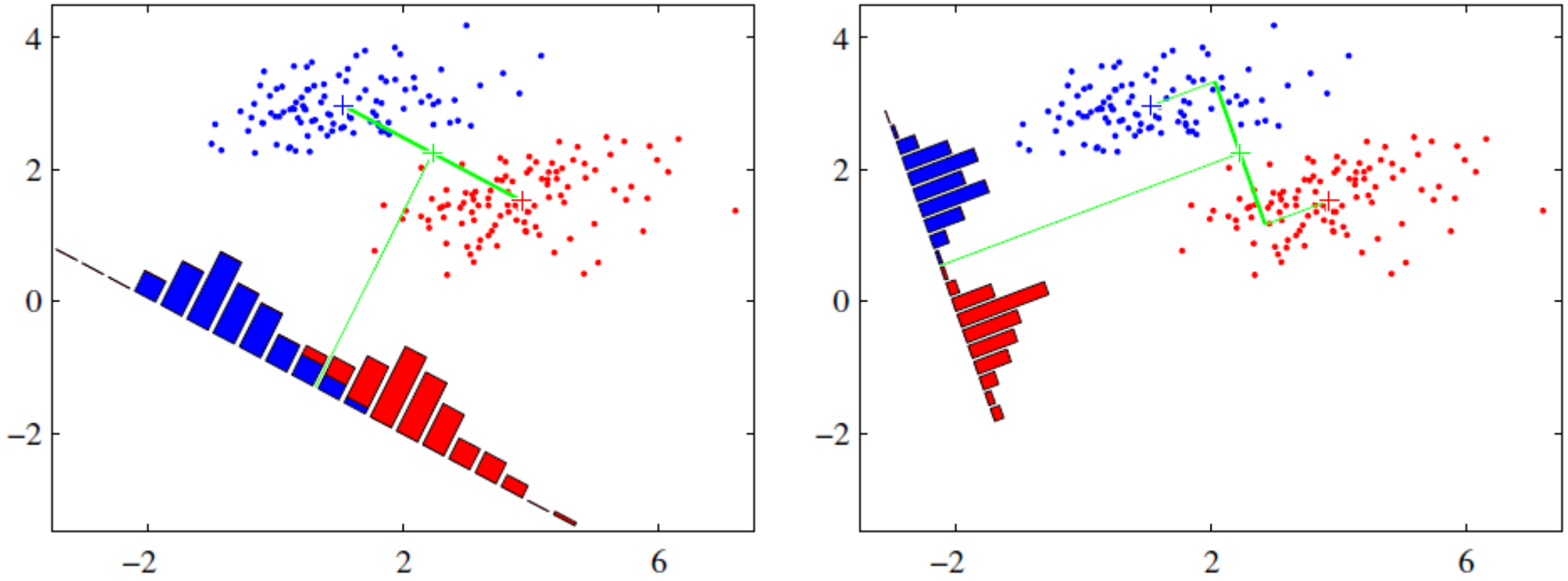
Isomap

Isomap is targeted to preserve geodesic distance on the manifold between two points



Fisher's LDA (Linear Discriminant Analysis) [1936]

Original idea: find a projection to discriminate classes best



Fisher's LDA is supervised method.

Fisher's LDA

Mean and variance within a single class c ($c \in \{1, 2, ..., C\}$):

$$\mu_k =< x >_{\text{samples of class c}}$$
$$\sigma_k =< ||x - \mu_k||^2 >_{\text{samples of class c}}$$

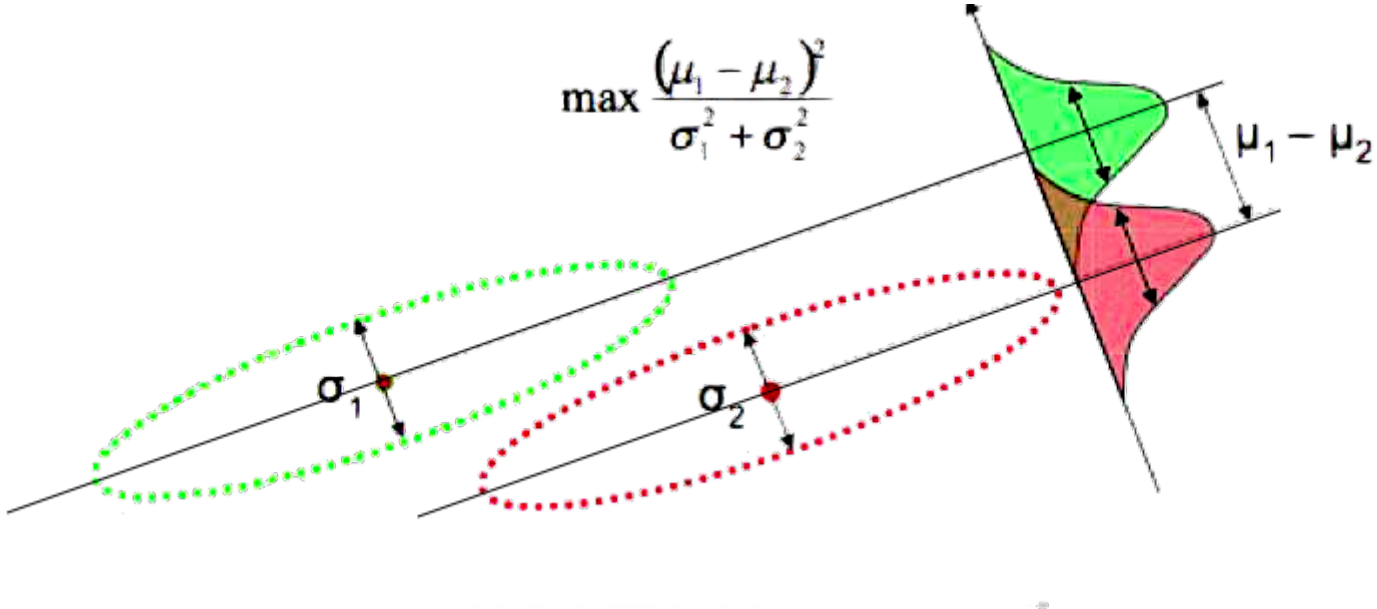
Total within-class variance: $\sigma_{\text{within}} = \sum_c p_c \sigma_c$

Total between-class variance: $\sigma_{\text{between}} = \sum_c p_c \, ||\mu_c - \mu||^2$

Goal: find a projection to optimize a ratio $\dfrac{\sigma_{\text{between}}}{\sigma_{\text{within}}}$

Question: should we maximize or minimize the ratio above?

Fisher's LDA



LDA: solving optimization problem

We are interested in finding 1-dimensional projection w:

$$\frac{w^T \Sigma_{\text{within}} w}{w^T \Sigma_{\text{between}} w} \to \max_w$$

- Naturally connected to the generalized eigenvalue problem
- Projection vector corresponds to the highest generalized eigenvalue
- Finds a subspace of $C - 1$ components when applied to a classification problem with C classes

Fisher's LDA is a basic popular binary classification technique.

Common spacial patterns

When we expect that each class is close to some linear subspace, we can

- (naively) find for each class this subspace by PCA
- (better idea) take into account variation of other data and optimize

Natural generalization is to take several components: $W \in \mathbb{R}^{n \times n_1}$ is a projection matrix; n and n_1 are number of dimensions in original and new spaces

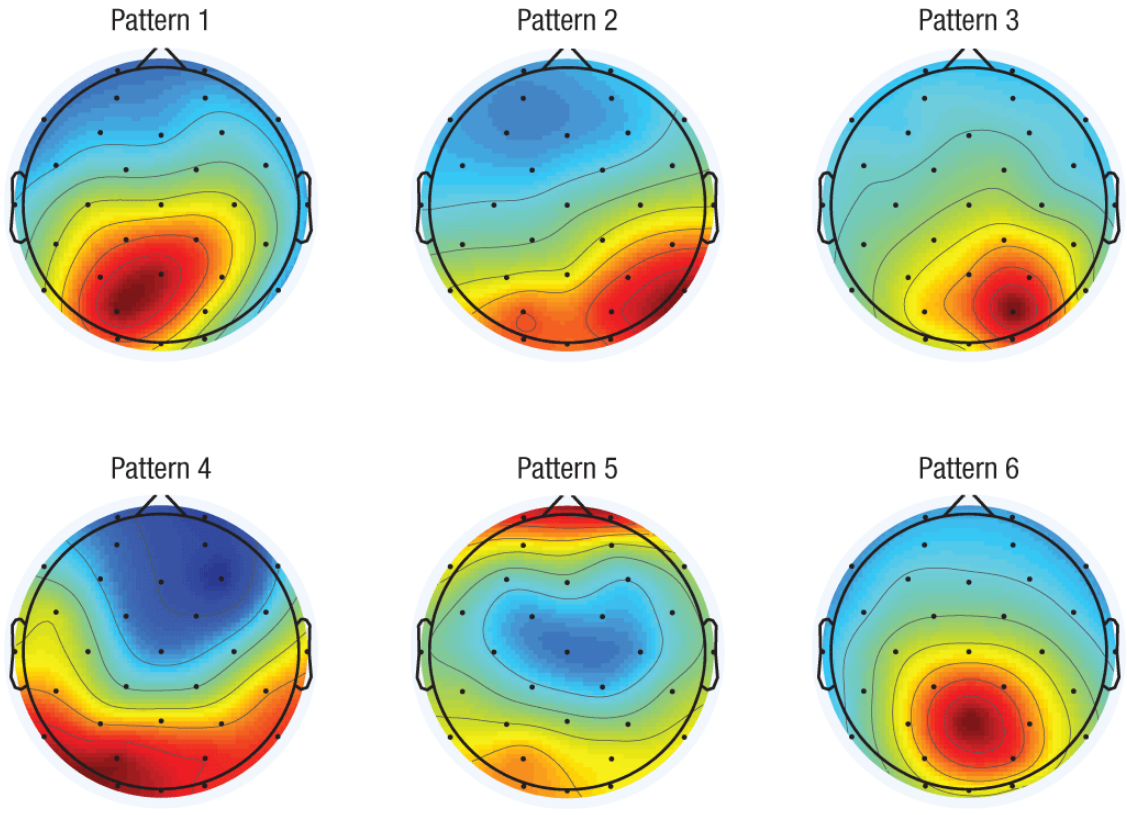
$$\operatorname{tr} W^T \Sigma_{\text{class}} W \to \max$$

$$\text{subject to} \qquad W^T \Sigma_{\text{total}} W = I$$

Frequently used in neural sciences, in particular in BCI based on EEG / MEG.

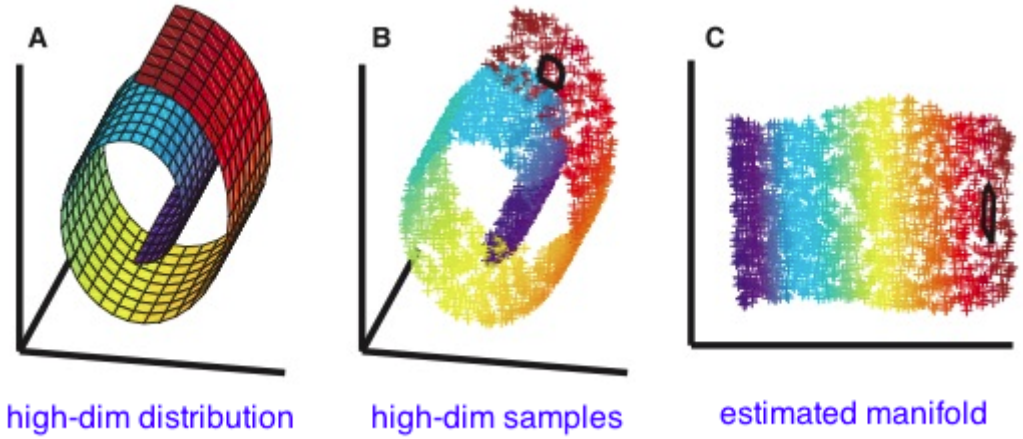
Common spacial patterns

Patters found describe the projection into 6-dimensional space



Dimensionality reduction summary

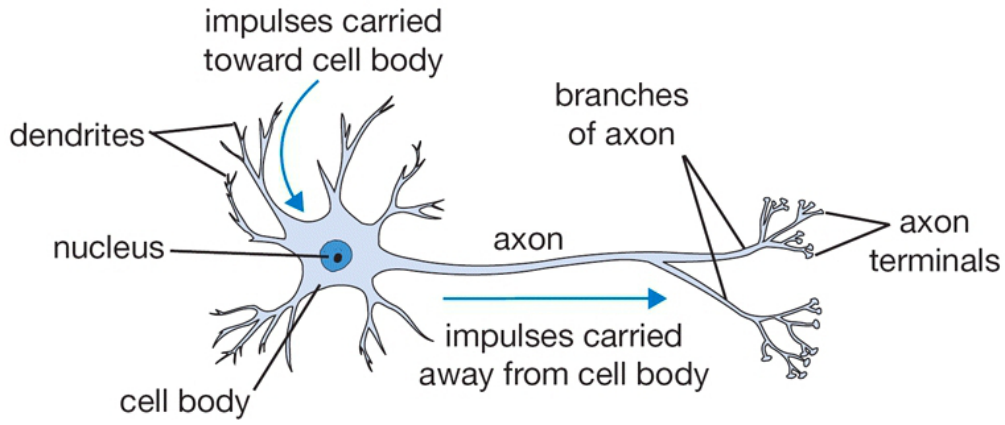
- is capable of extracting sensible features from highly-dimensional data
- frequently used to 'visualize' the data
- nonlinear methods rely on the distance in the space
- works well with highly-dimensional spaces with features of the same nature



Artificial Neural Networks

Pre-history

- 1943, W. McCulloch & W. Pitts — first computational model
- 1949, D. Hebb — method for learning
- 1954, W. Clark — first simulations



1980s

in applications linear models (in particular, SVM) clearly demonstrate better performance than NN

2006 — till now

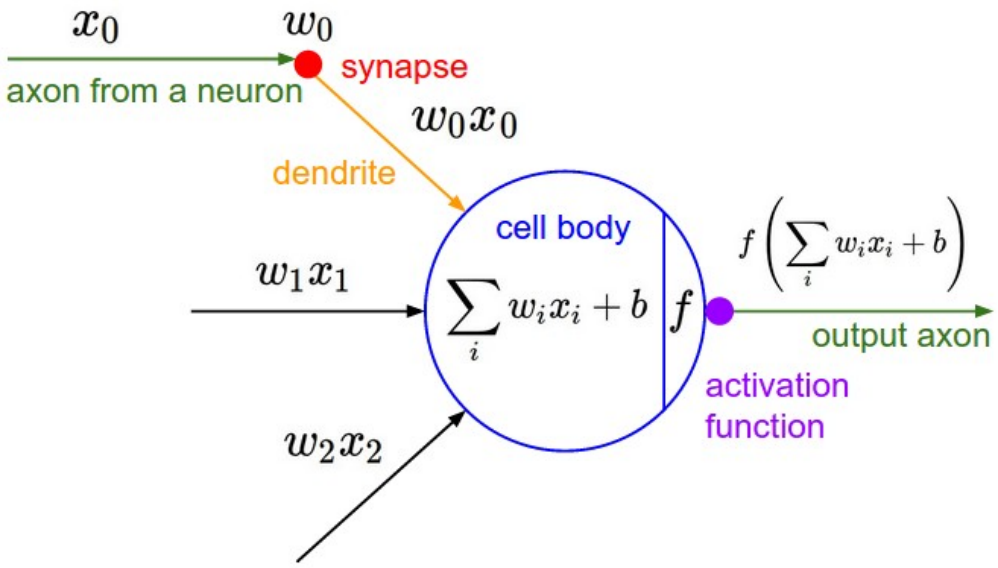
Neural networks Renaissance

Single neuron

Neuron can be activated or not ($h = 1$, $h = 0$) by signals from receptors or other neurons.

In simplest case:

- $h = \Theta\left(\sum_i w_i x_i + w_0\right)$, Θ — Heaviside function (linear model!)
- non-smooth $\Rightarrow$ hard to optimize



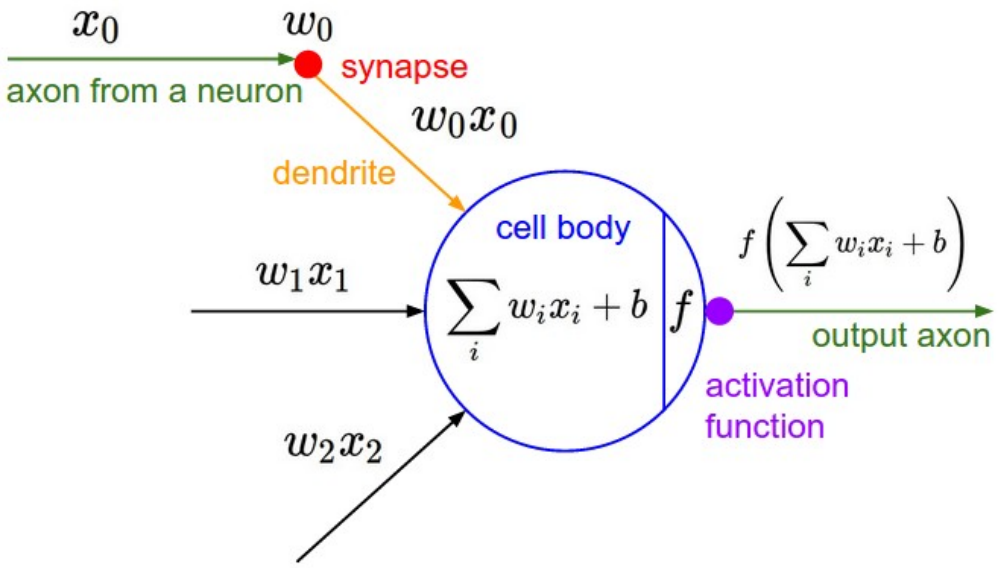
Single neuron

Neuron can be activated or not ($h = 1$, $h = 0$) by signals from receptors or other neurons.

In simplest case:

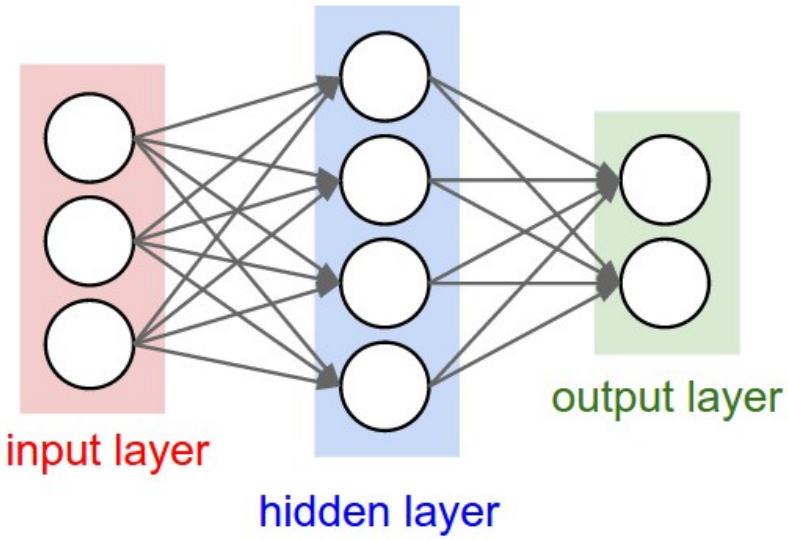
- $h = \Theta\left(\sum_i w_i x_i + w_0\right)$, Θ — Heaviside function (linear model!)
- non-smooth $\Rightarrow$ hard to optimize
- smooth version: $h = \sigma\left(\sum_i w_i x_i + w_0\right)$

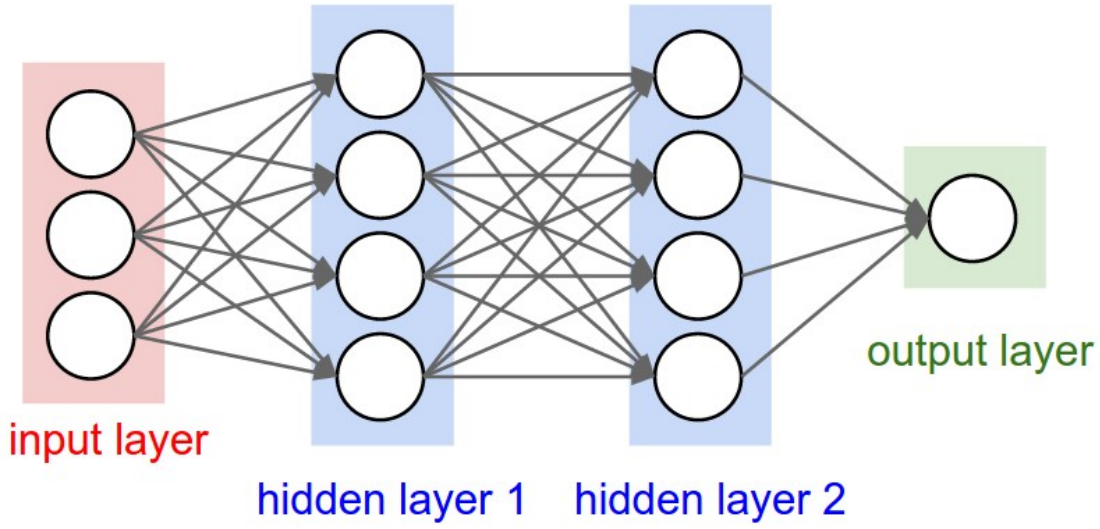
  (single neuron behaves like logistic regression!)



Neural network

Simplest case is multi-layer perceptron





How many parameters are there between *hidden layer 1* and *hidden layer 2*?

Computations in neural network

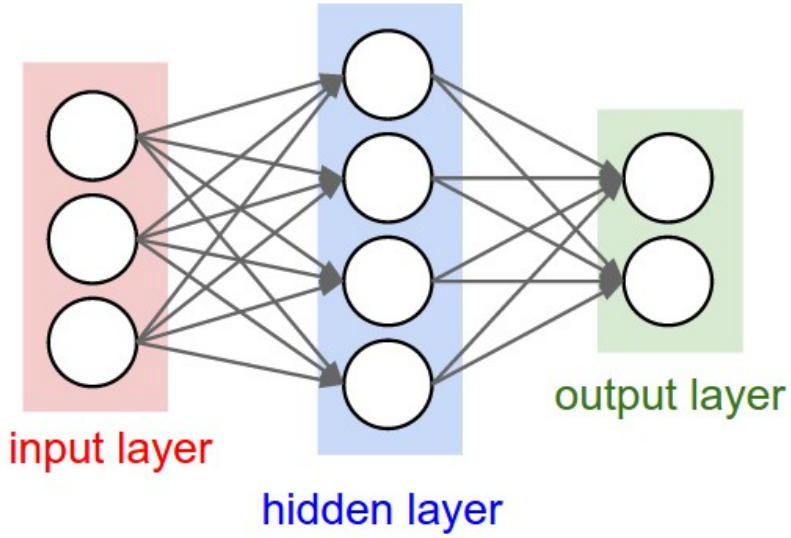
$\text{input} = (x_1, x_2, ..., x_d)$

Activations of hidden layer:

$h_i = \sigma(\sum_{j=1}^{\text{n\_inputs}} W_{ij} x_j + v_i)$

Activations of output layer:

$o_i = \sigma(\sum_{j=1}^{\text{n\_hidden}} \hat{W}_{ij} h_j + \hat{v}_i)$



Computations in neural network

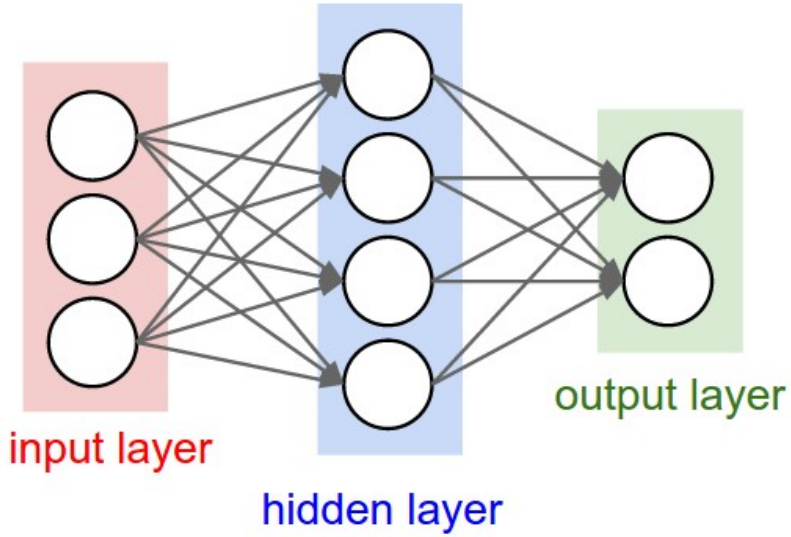
$\text{input} = (x_1, x_2, ..., x_d)$

Activations of hidden layer:

$h_i = \sigma(\sum_{j=1}^{\text{n\_inputs}} W_{ij} x_j + v_i)$

Activations of output layer:

$o_i = \sigma(\sum_{j=1}^{\text{n\_hidden}} \hat{W}_{ij} h_j + \hat{v}_i)$



Now the same in matrix form:

$$
\begin{aligned}
h_{\text{input}} &= Wx + v && \text{linear} \\
h &= \sigma(h_{\text{input}}) && \text{non-linear} \\
o_{\text{input}} &= \hat{W}h + \hat{v} && \text{linear} \\
h &= \sigma(h_{\text{input}}) && \text{non-linear}
\end{aligned}
$$

Many layers

In matrix form (two steps are merged together):

$$
\begin{aligned}
h_1 &= \sigma(W_1 x + v_1) \\
h_2 &= \sigma(W_2 h_1 + v_2) \\
h_3 &= \sigma(W_3 h_2 + v_3) \\
&\dots \\
\text{output} &= \sigma(W_n h_{n-1} + v_n)
\end{aligned}
$$

Other non-linear functions can be used instead of logistic. Popular choice is ReLU (Rectifier Linear Unit):

$$\text{ReLU}(x) = \max(0, x)$$

Training a neural network

Mathematically, neural network is a function with many parameters, so we use smooth optimization. Output of network can be a single value (in one neuron) or several.

The only thing needed is to have some smooth function for optimization. Typically same losses are used

- For regression (f — output of a neuron):

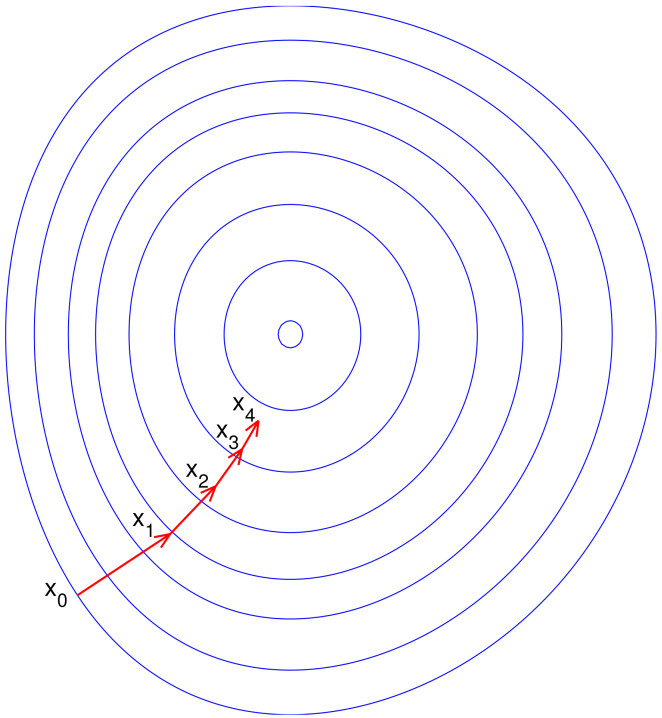
$$\mathcal{L} = \sum_i (y_i - f(x_i))^2$$

- For classification (n output neurons, one for each class):

$$\mathcal{L} = -\sum_i \log(p_{y_i}(x_i))$$

Convnet.js playground

Gradient descent



Problem: find w to minimize $\mathcal{L}$.

Gradient descent:

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$$

η is a step size
(also called *shrinkage*, *learning rate*)

Other optimizations?

Gradient boosting is optimization itself.

In case of logistic/linear regression the function is smooth and convex.

- many advanced method of smooth optimization work very fine (BFGS, CG, Newton-CG)

In case of factorization model specific algorithms are good because of tractability

- alternating least squares (ALS) is the most common

Neural networks (thousands to millions of parameters)

- not convex, optimization is not tractable, computing gradient w.r.t. to all dataset is very slow

Stochastic gradient descent (SGD)

$$\mathcal{L} = \frac{1}{N} \sum_i L(x_i, y_i) \to \min$$

On each iteration make a step using only one observation:

- take i — random observation from training data
- $w \leftarrow w - \eta \dfrac{\partial L(x_i, y_i)}{\partial w}$

Stochastic gradient descent (SGD)

$$\mathcal{L} = \frac{1}{N} \sum_i L(x_i, y_i) \to \min$$

On each iteration make a step using only one observation:

- take i — random observation from training data
- $w \leftarrow w - \eta \dfrac{\partial L(x_i, y_i)}{\partial w}$

Each iteration is done much faster, but training process is less stable.
Making smaller steps.

Stochastic gradient descent

We can decrease the *learning rate* over time: η_t.

At iteration t:

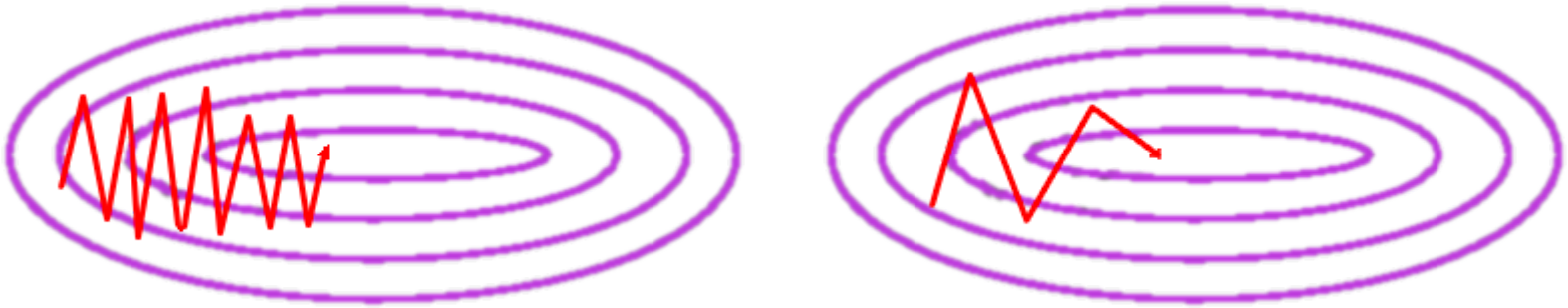
$$w \leftarrow w - \eta_t \frac{\partial L(x_i, y_i)}{\partial w}$$

This process converges to local minima if:

$$\sum_t \eta_t = \infty, \quad \sum_t \eta_t^2 < \infty, \qquad \eta_t > 0$$

SGD with momentum

SGD (and GD) has problems with narrow valley (when hessian is very far from identity).

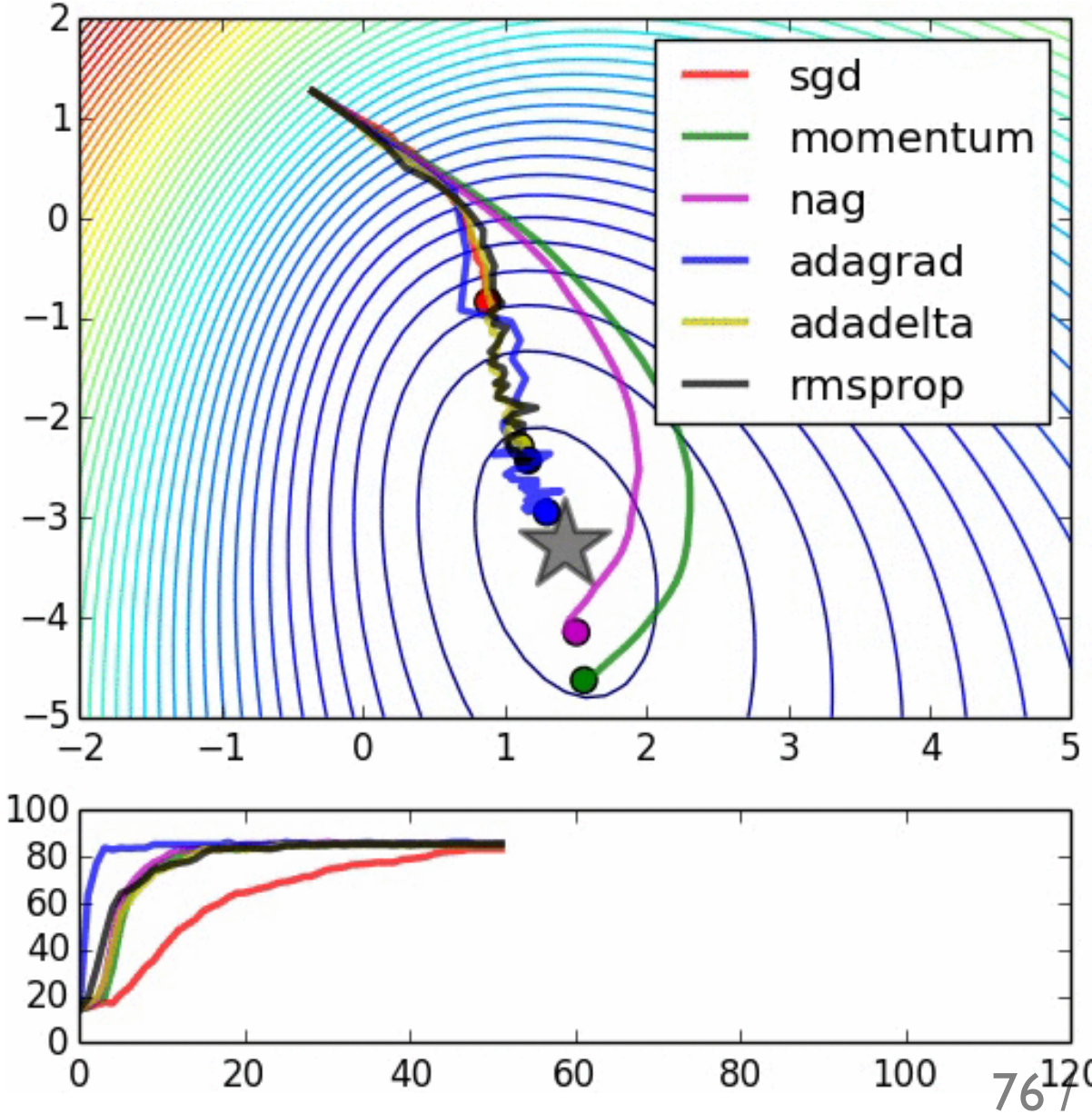


Improvement: use *momentum* which accumulates gradient, $0.9 < \gamma < 1$

$$v \leftarrow \gamma v + \eta_t \frac{\partial L(x_i, y_i)}{\partial w}$$
$$w \leftarrow w - v$$

Stochastic optimization methods

Animation of optimization trajectories for some toy 2-parameter optimization



Stochastic optimization methods

- applied to additive loss function

$$\mathcal{L} = \sum_i L(x_i, y_i)$$

- should be preferred when optimization time is the bottleneck
- more advanced modifications exist:
  - AdaDelta, RMSProp, Adam
  - those are using adaptive step size (individually for each neuron)
  - crucial when scale of gradients is very different
- in practice predictions are computed using minibatches (small groups of 16 to 256 samples) not on sample-by-sample basis

Regularizations

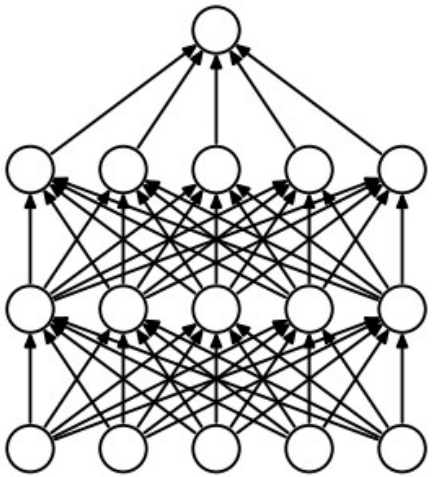
Many parameters $\rightarrow$ overfitting (remember demo?)

- L_1, L_2 — regularizations are still used
- but this isn't enough!

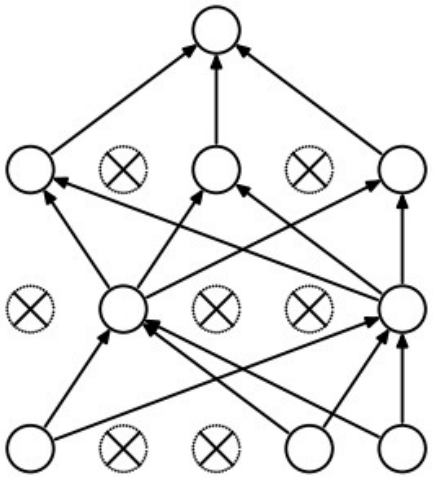
Regularizations

Many parameters $\rightarrow$ overfitting (remember demo?)

- L_1, L_2 — regularizations are still used
- but this isn't enough!
- a better idea: dropout, at each step of optimization some subset of neurons is dropped randomly
- this prevents co-adaptation of neurons and makes the neural network more stable



(a) Standard Neural Net



(b) After applying dropout.

Let's implement this!

Some practicalities

of machine learning

Feature engineering

Feature engineering = creating features to get the best result with ML

- important step
- mostly relying on domain knowledge
- requires some understanding
- most of practitioners' time is spent at this step

Feature engineering

- Analyzing available features
  - scale and shape of features
- Analyze which information lacks and can be inferred
- Cross-validate your guesses

Feature engineering

- Analyzing available features
  - scale and shape of features
- Analyze which information lacks and can be inferred
- Cross-validate your guesses
- Machine learning is a proper tool for checking your understanding of data

Linear models example

Single observation with sufficiently large value of feature can break almost all linear models.

Heavy-tailed distributions are harmful, pretransforming required

- standardization (linear scaling)
- logarithm transform
- power transform
- and/or throwing out outliers from dataset

The same tricks actually help to more advanced methods.

Question: Which transformation is the best for Random Forest?

One-hot encoding

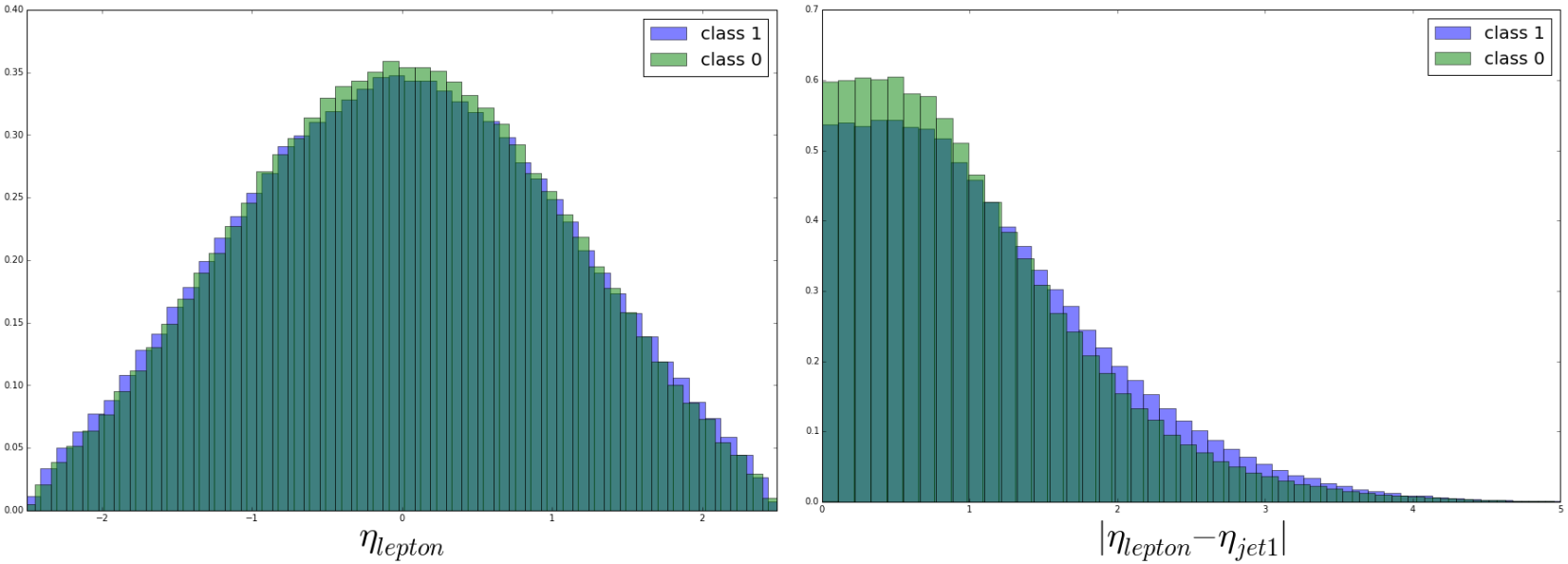
Categorical features (= 'not orderable'), being one-hot encoded, are easier for ML to operate with

Original data:	
id	Color
1	White
2	Red
3	Black
4	Purple
5	Gold

One-hot encoding format:					
id	White	Red	Black	Purple	Gold
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

Decision tree example

η_{lepton} is hard for tree to use, since provides no good splitting



Don't forget that a tree can't reconstruct linear combinations — take care of this.

Example of feature: invariant mass

Using HEP coordinates, invariant mass of two products is:

$$m^2_{\text{inv}} \approx 2p_{T1}p_{T2}\left(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2)\right)$$

Can't be recovered with ensembles of trees of depth < 4 when using only canonical features.

Invariant mass of 3 particles — much harder.
Such features should be provided (if those are appropriate for the problem).

Example of feature: invariant mass

Using HEP coordinates, invariant mass of two products is:

$$m^2_{\text{inv}} \approx 2p_{T1}p_{T2}\left(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2)\right)$$

Can't be recovered with ensembles of trees of depth < 4 when using only canonical features.

Invariant mass of 3 particles — much harder.
Such features should be provided (if those are appropriate for the problem).

Good features are ones that are explainable by domain knowledge (physics).
Start from simplest and most natural.

Output engineering

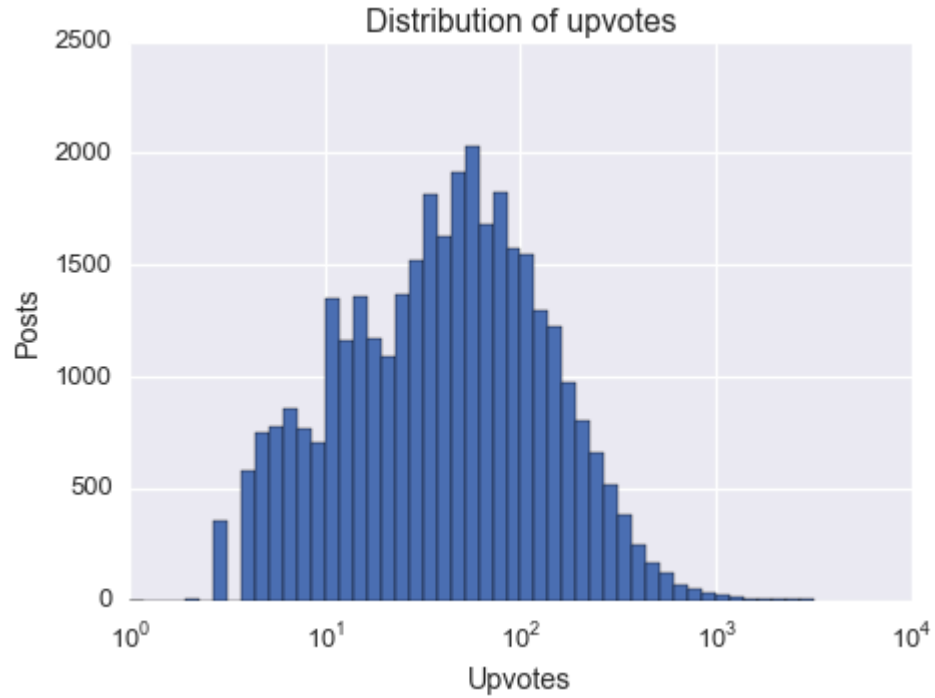
Typically not discussed, but the target of learning plays an important role.

Example: predicting number of upvotes for post or comment. Assuming the error is MSE

$$\mathcal{L} = \sum_i (d(x_i) - y_i)^2 \to \min$$

Consider two cases:

- 100 comments when predicted 0
- 1200 comments when predicted 1000



Output engineering

Ridiculously large impact of highly-commented articles.
We need to predict order, not exact number of comments.

Possible solutions:

- alternate loss function. E.g. use MAPE
- apply logarithm to the target, predict $\log(\#\text{comments})$

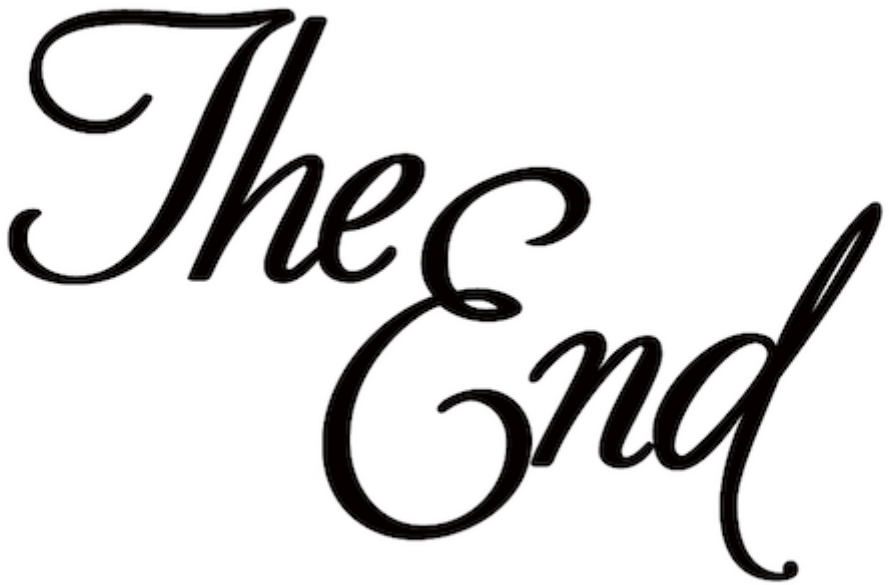
Evaluation score should be changed accordingly.

Sample weights

Typically used to estimate the contribution of observation (how often we expect this to happen).

Sample weights in some situations also matter.

- highly misbalanced dataset (e.g. 99 % of observations in class 0) tend to have problems during optimization.
- changing sample weights to balance the dataset frequently helps.


The End