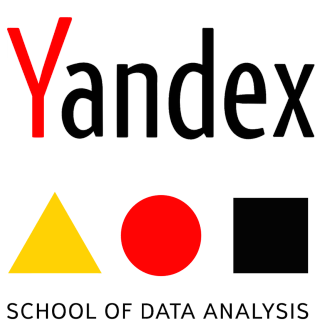
Machine Learning in High Energy Physics

Lectures 5 & 6

Alex Rogozhnikov

Lund, MLHEP 2016



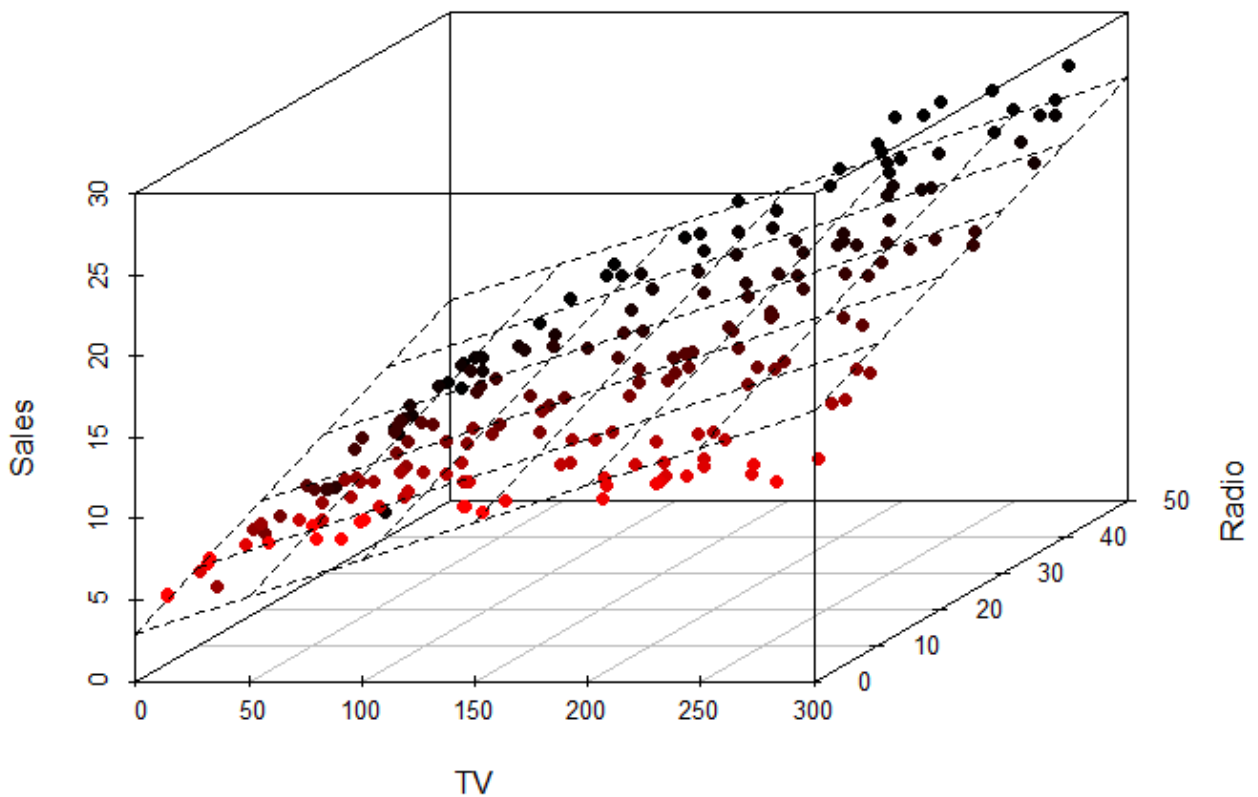
Linear models: linear regression

$$d(x) =< w, x > +w_0$$

Minimizing MSE:

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

$$L_{\text{MSE}}(x_i, y_i) = (d(x_i) - y_i)^2$$

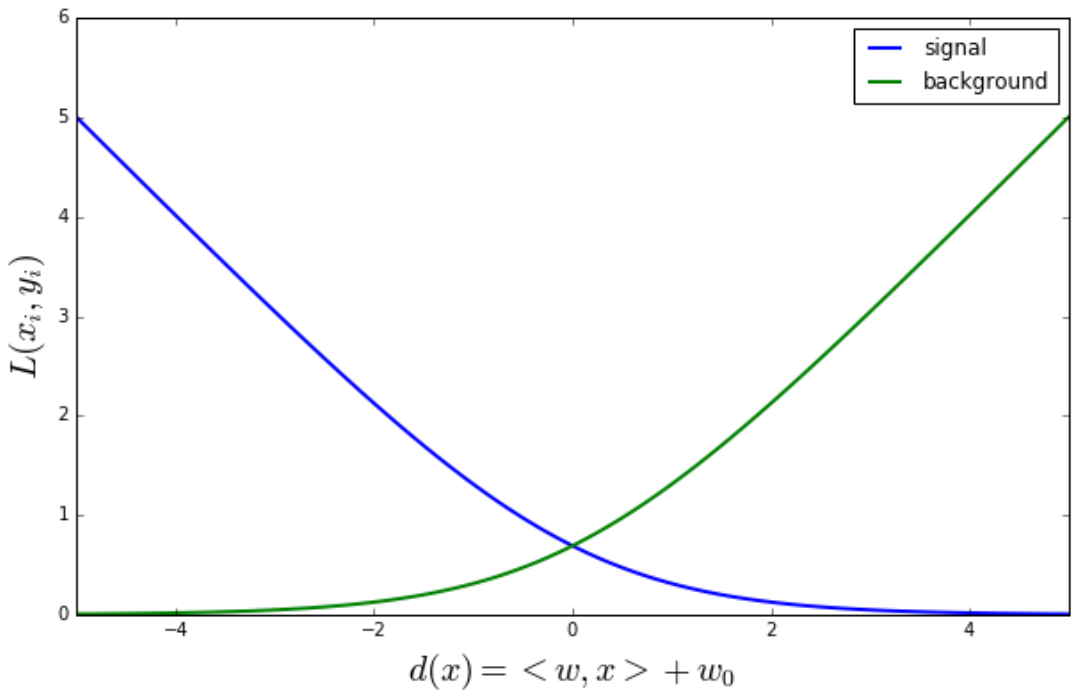


Linear models: logistic regression

$d(x) =< w, x > +w_0$

Minimizing logistic loss:
$\mathcal{L} = \sum_i L_{\text{logistic}}(x_i, y_i) \to \min$

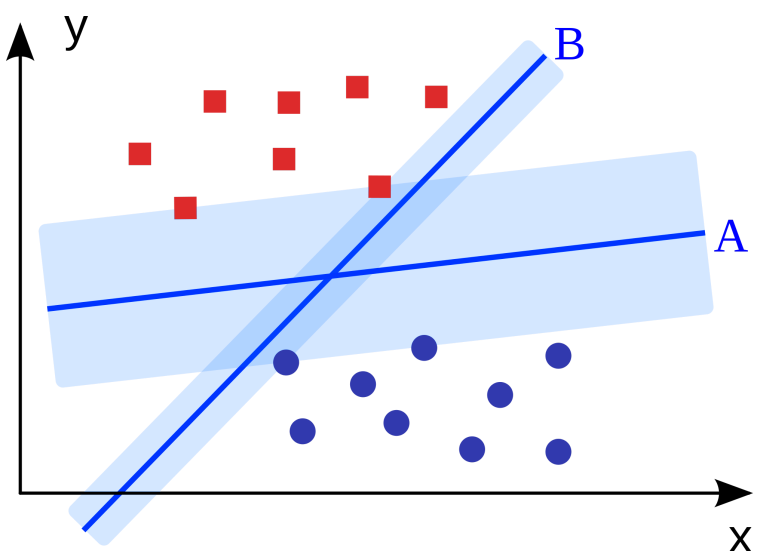
Penalty for single observation $y_i \pm 1$:
$L_{\text{logistic}}(x_i, y_i) = \ln(1 + e^{-y_i d(x_i)})$

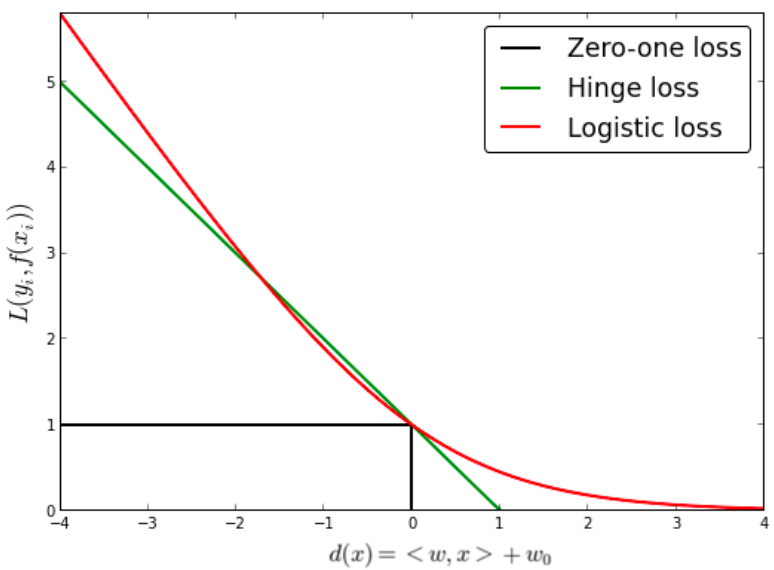


Linear models: support vector machine (SVM)

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

Margin $y_i d(x_i) > 1 \rightarrow$ no penalty

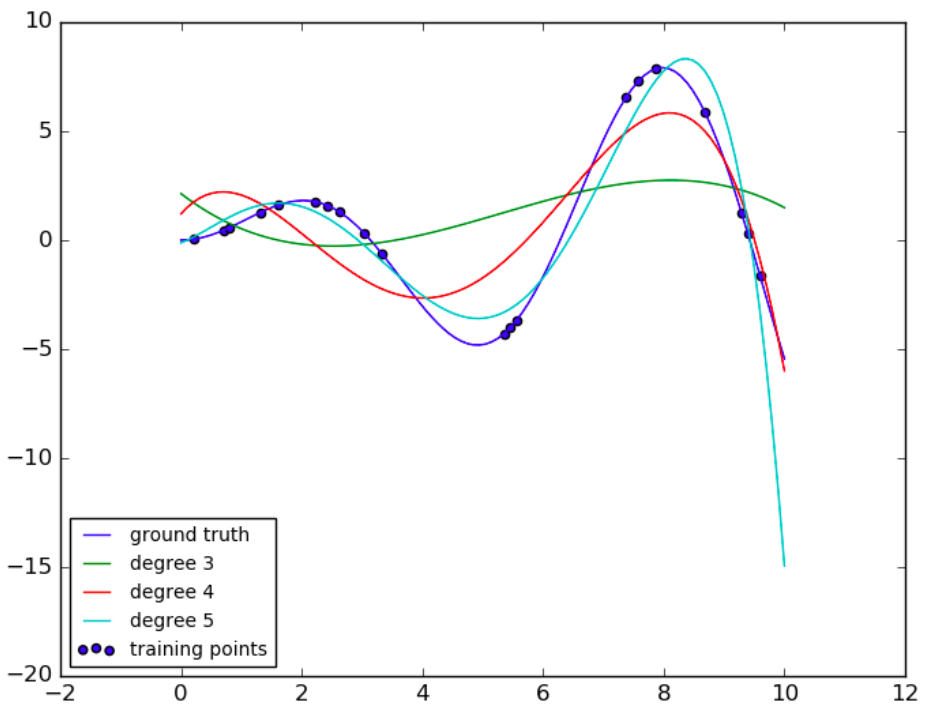


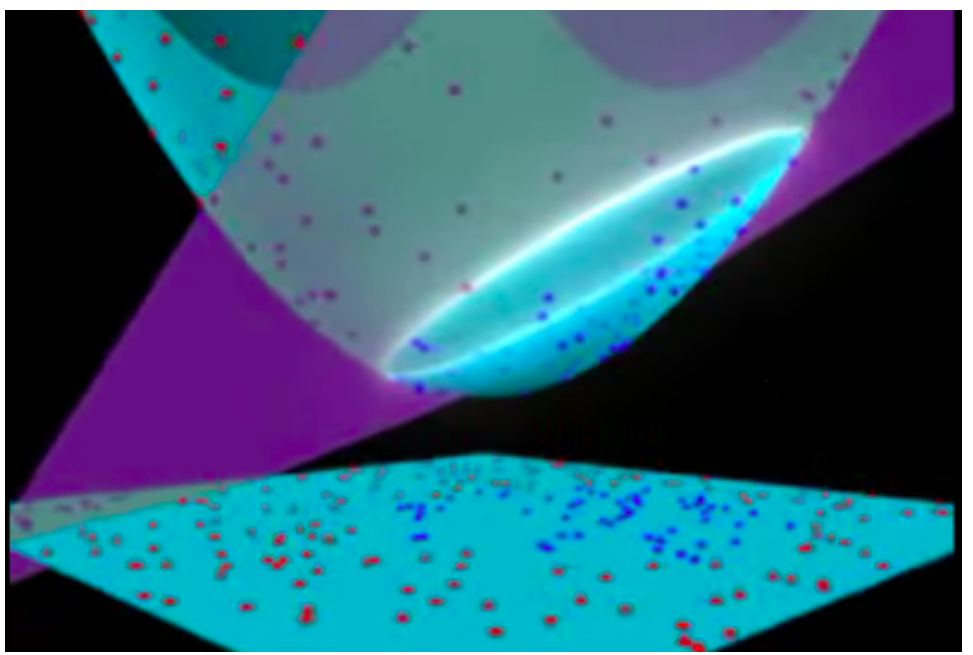


Kernel trick

we can project data into higher-dimensional space, e.g. by adding new features.

Hopefully, in the new space distributions are separable





Kernel trick

P is a projection operator:
$w = \sum_i \alpha_i P(x_i)$

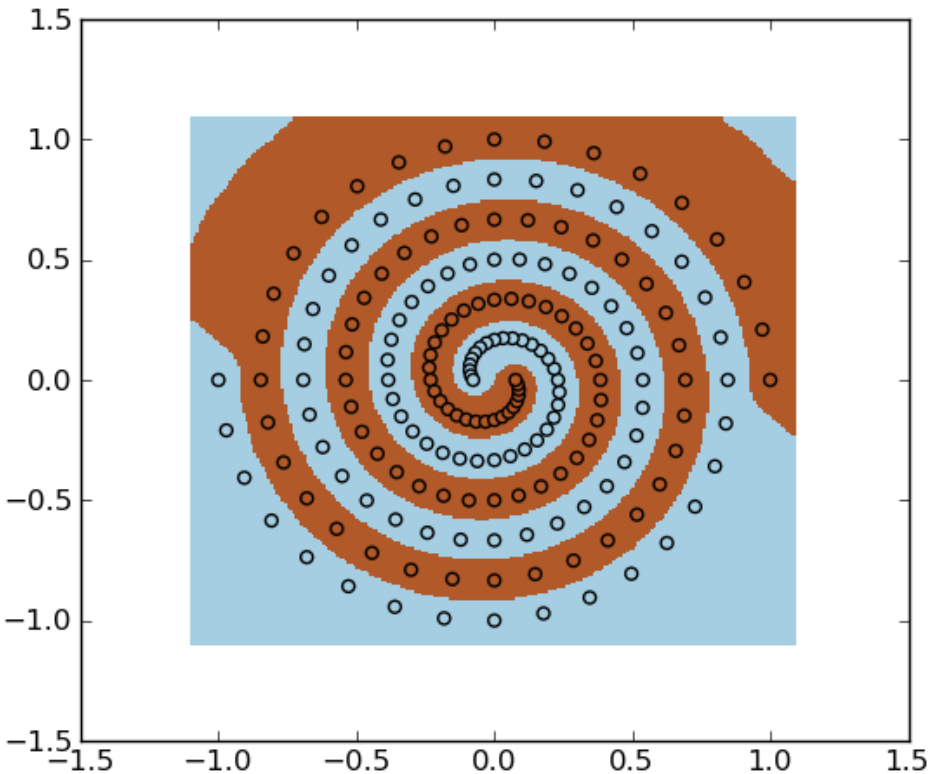
$d(x) = < w, P(x) >_{\text{new}}$

$d(x) = \sum_i \alpha_i K(x_i, x)$

We need only *kernel*:

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

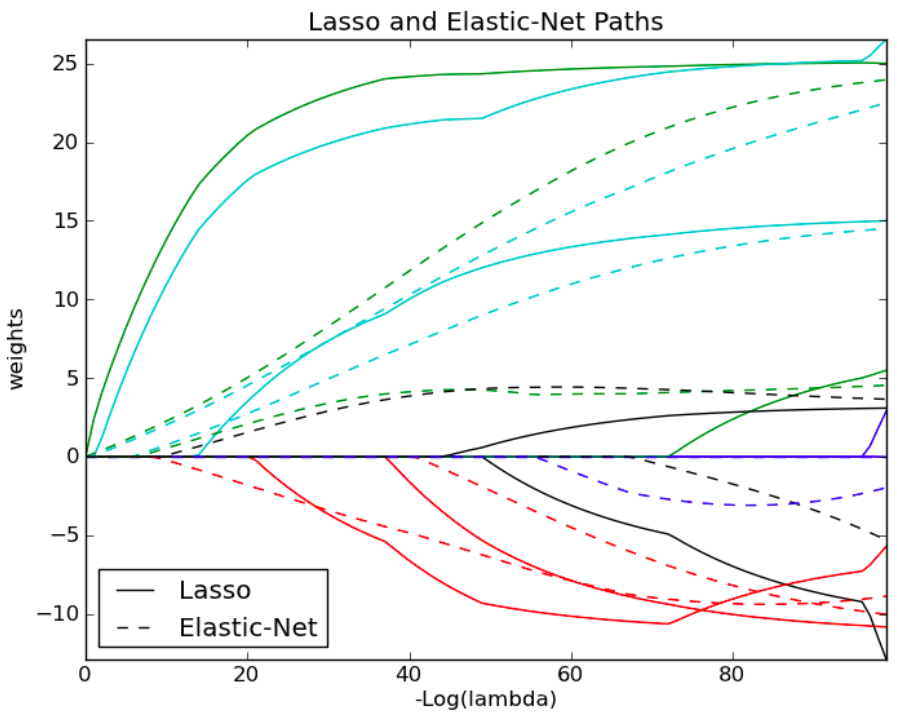
Popular choices: polynomial kernel and RBF kernel.



Regularizations

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

- L_2 regularization : $\mathcal{L}_{\text{reg}} = \alpha \sum_j |w_j|^2$
- L_1 regularization: $\mathcal{L}_{\text{reg}} = \beta \sum_j |w_j|$
- $L_1 + L_2$: $\mathcal{L}_{\text{reg}} = \alpha \sum_j |w_j|^2 + \beta \sum_j |w_j|$

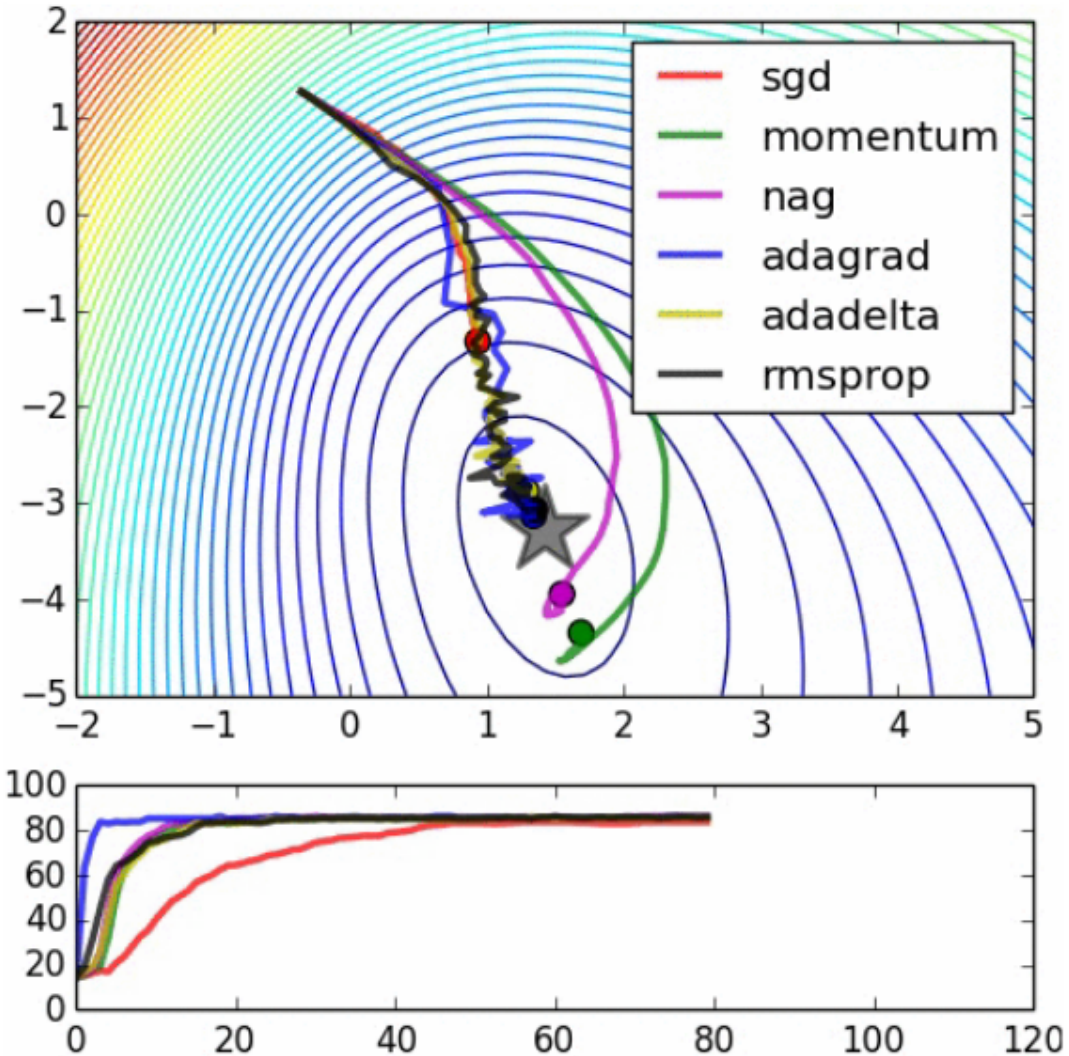


Stochastic optimization methods

Stochastic gradient descent

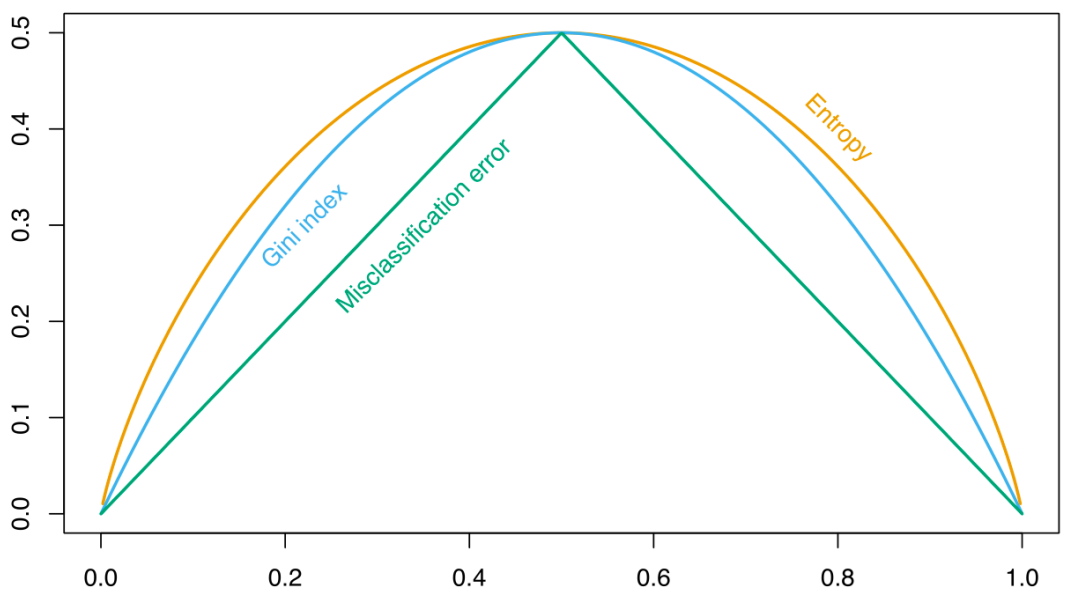
- take i — random event from training data
- $w \leftarrow w - \eta \frac{\partial L(x_i, y_i)}{\partial w}$

(can be applied to additive loss functions)



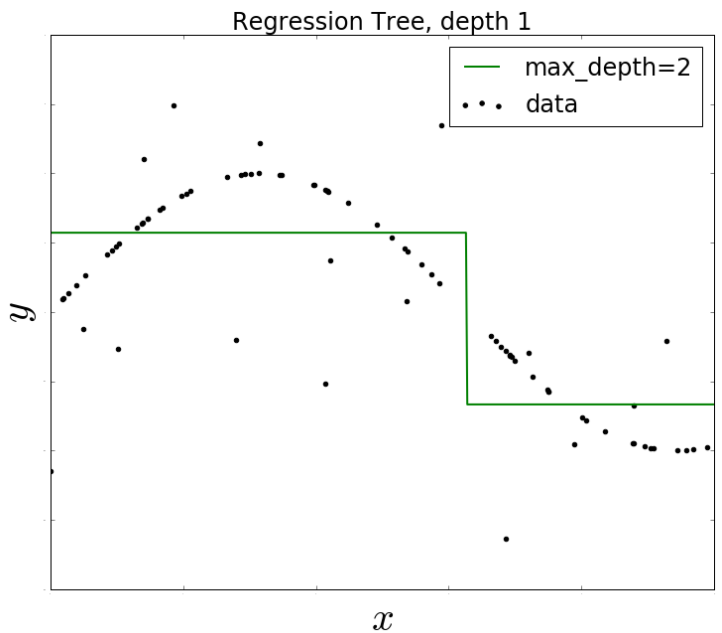
Decision trees

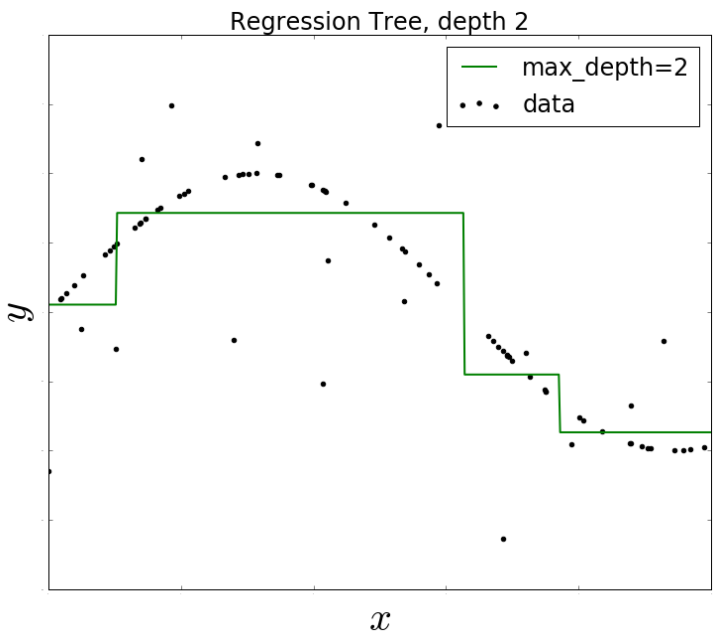
- NP complex to build
- heuristic: use greedy optimization
- optimization criterions (impurities): misclassification, Gini, entropy

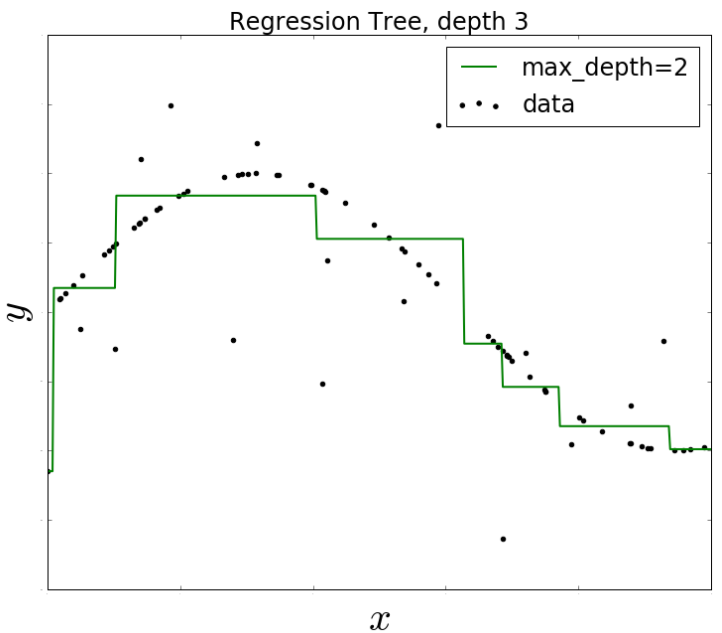


Decision trees for regression

Optimizing MSE, prediction inside a leaf is constant.

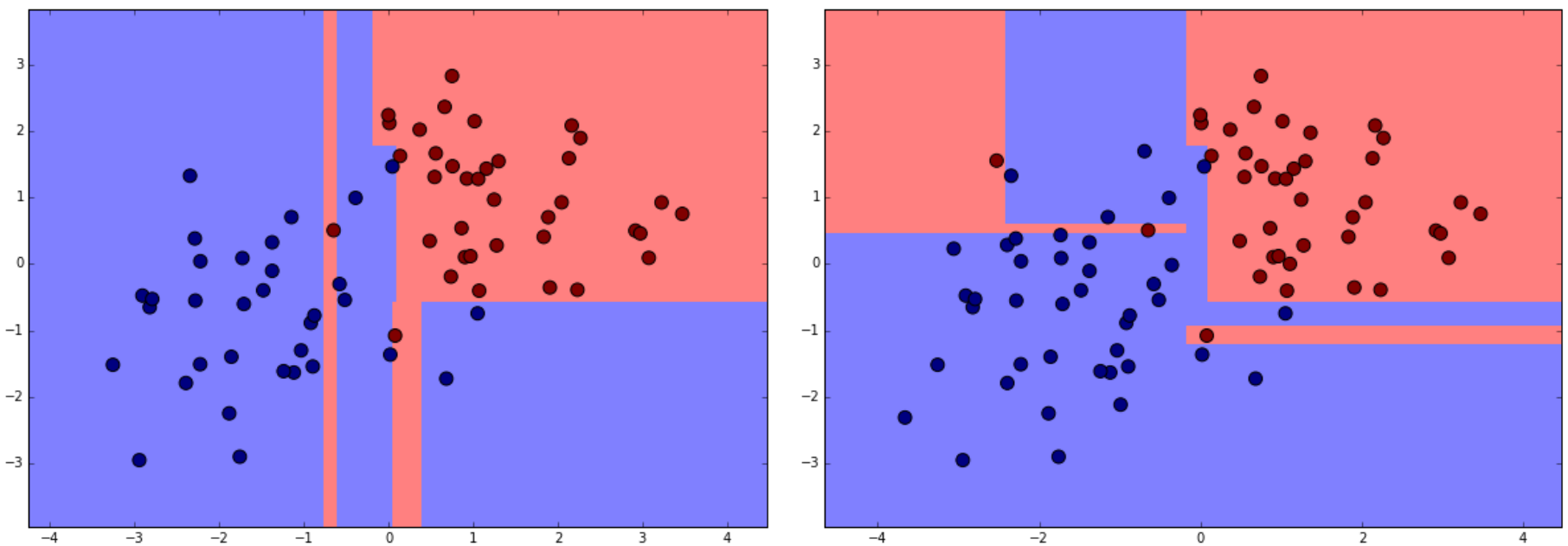






Overfitting in decision tree

- pre-stopping
- post-pruning
- unstable to the changes in training dataset

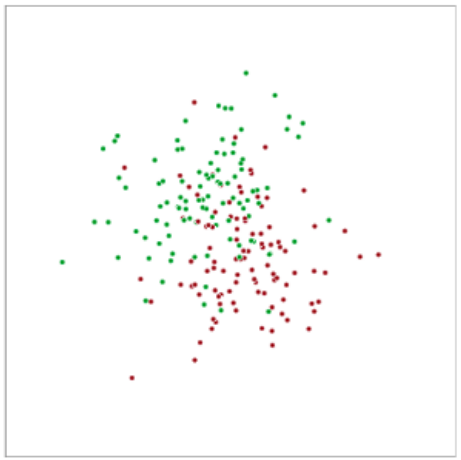


Random Forest

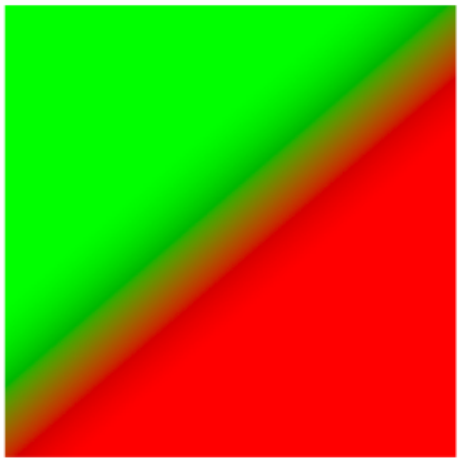
Many trees built independently

- bagging of samples
- subsampling of features

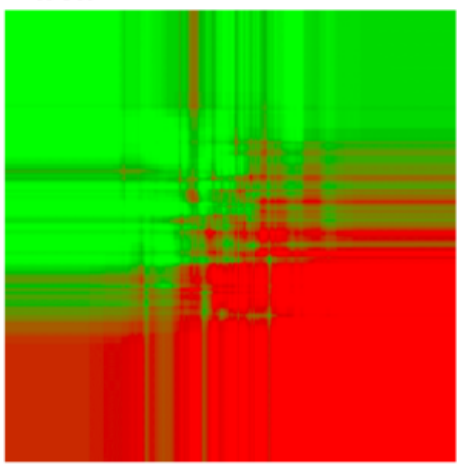
Simple voting is used to get prediction of an ensemble



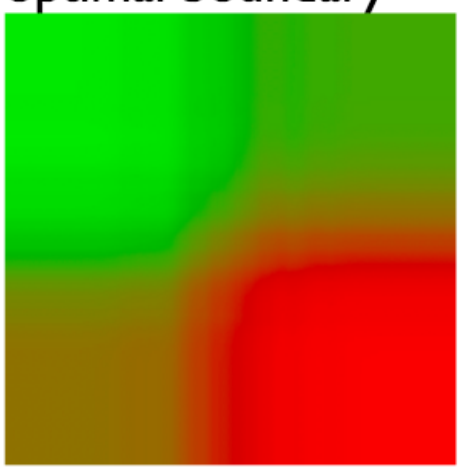
data



optimal boundary

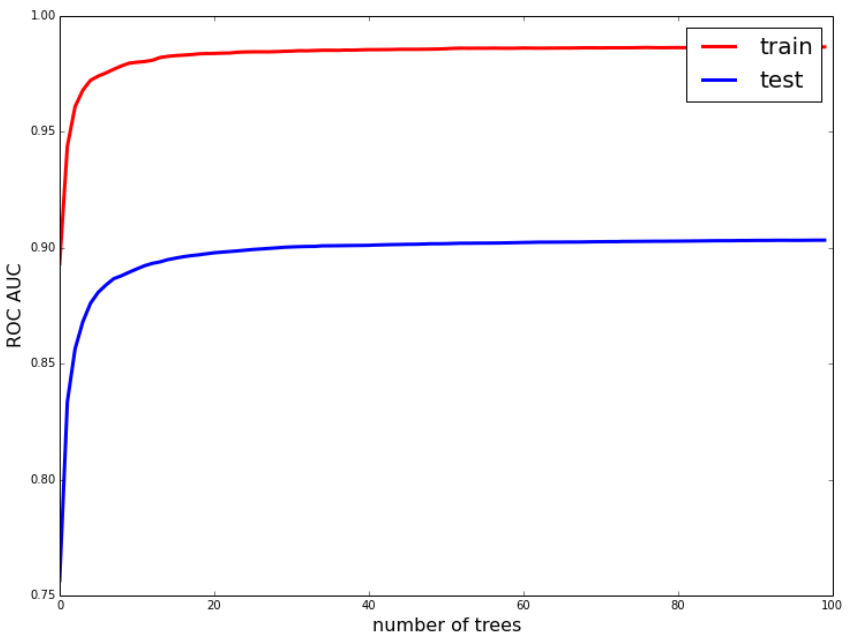


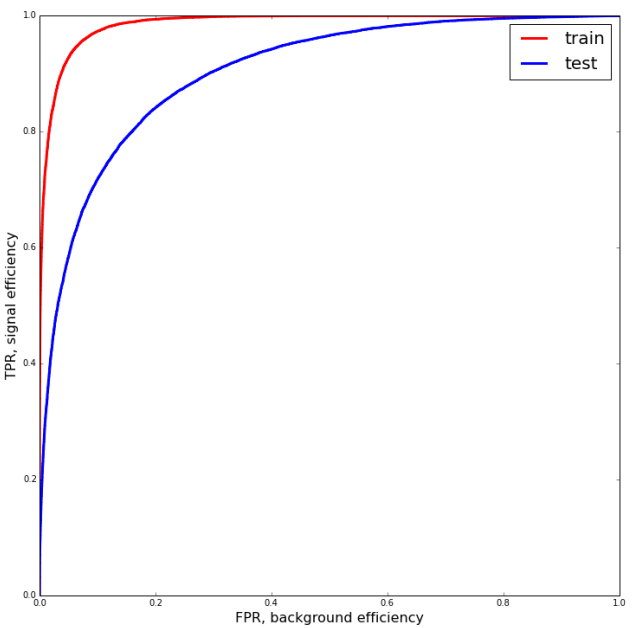
50 trees



2000 trees

Random Forest





- overfitted (in the sense that predictions for train and test are different)
- doesn't overfit: increasing complexity (adding more trees) doesn't spoil a classifier

Random Forest

- simple and parallelizable
- doesn't require much tuning
- hardly interpretable
  - but feature importances can be computed
- doesn't fix samples poorly classifies at previous stages

Ensembles

- Averaging decision functions

$$D(x) = \frac{1}{J} \sum_{j=1}^{J} d_j(x)$$

- Weighted decision

$$D(x) = \sum_j \alpha_j d_j(x)$$

Sample weights in ML

Can be used with many estimators. We now have triples

$$x_i, y_i, w_i \qquad i - \text{ index of an event}$$

- weight corresponds to frequency of observation
- expected behavior: $w_i = n$ is the same as having n copies of ith event
- global normalization of weights doesn't matter

Sample weights in ML

Can be used with many estimators. We now have triples

$$x_i, y_i, w_i \qquad i - \text{ index of an event}$$

- weight corresponds to frequency of observation
- expected behavior: $w_i = n$ is the same as having n copies of ith event
- global normalization of weights doesn't matter

Example for logistic regression:

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

Weights (parameters) of a classifier ≠ sample weights

In code:

```
tree = DecisionTreeClassifier(max_depth=4)
tree.fit(X, y, sample_weight=weights)
```

Sample weights are convenient way to regulate importance of training events.

Only sample weights when talking about AdaBoost.

AdaBoost [Freund, Shapire, 1995]

Bagging: information from previous trees not taken into account.

Adaptive Boosting is a weighted composition of weak learners:

$$D(x) = \sum_j \alpha_j d_j(x)$$

We assume $d_j(x) = \pm 1$, labels $y_i = \pm 1$,

jth weak learner misclassified ith event iff $y_i d_j(x_i) = -1$

AdaBoost

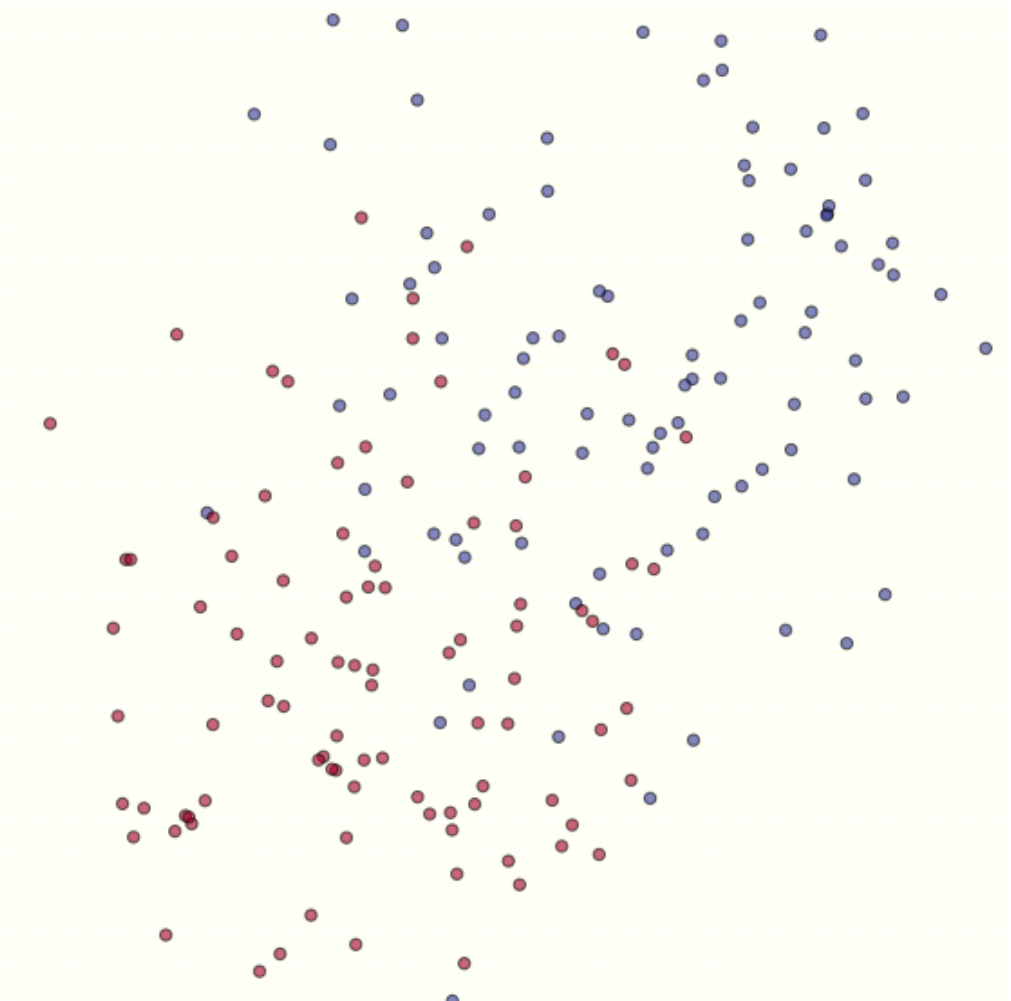
$$D(x) = \sum_j \alpha_j d_j(x)$$

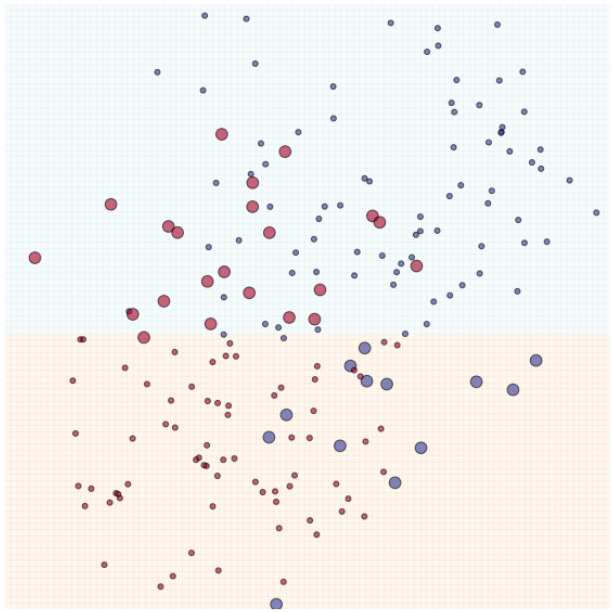
Weak learners are built in sequence, each classifier is trained using different weights

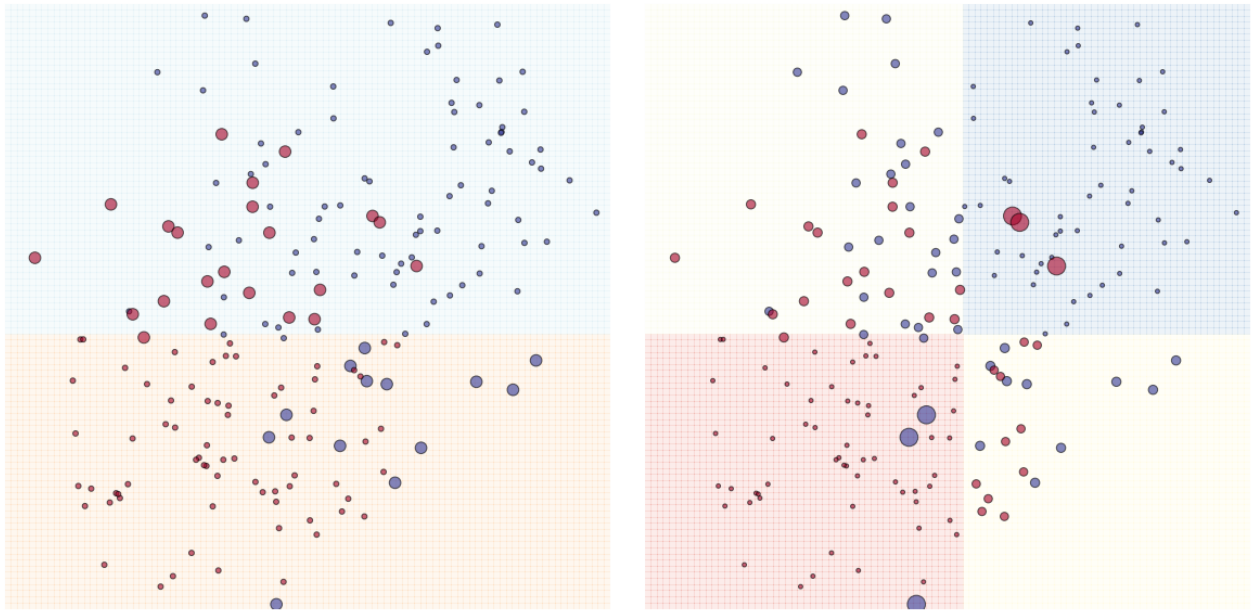
- initially w_i = 1 for each training sample
- After building jth base classifier:
  1. compute the total weight of correctly and wrongly classified events
     $\alpha_j = \frac{1}{2}\ln\left(\frac{w_{\text{correct}}}{w_{\text{wrong}}}\right)$
  2. increase weight of misclassified samples
     $w_i \leftarrow w_i \times e^{-\alpha_j y_i d_j(x_i)}$

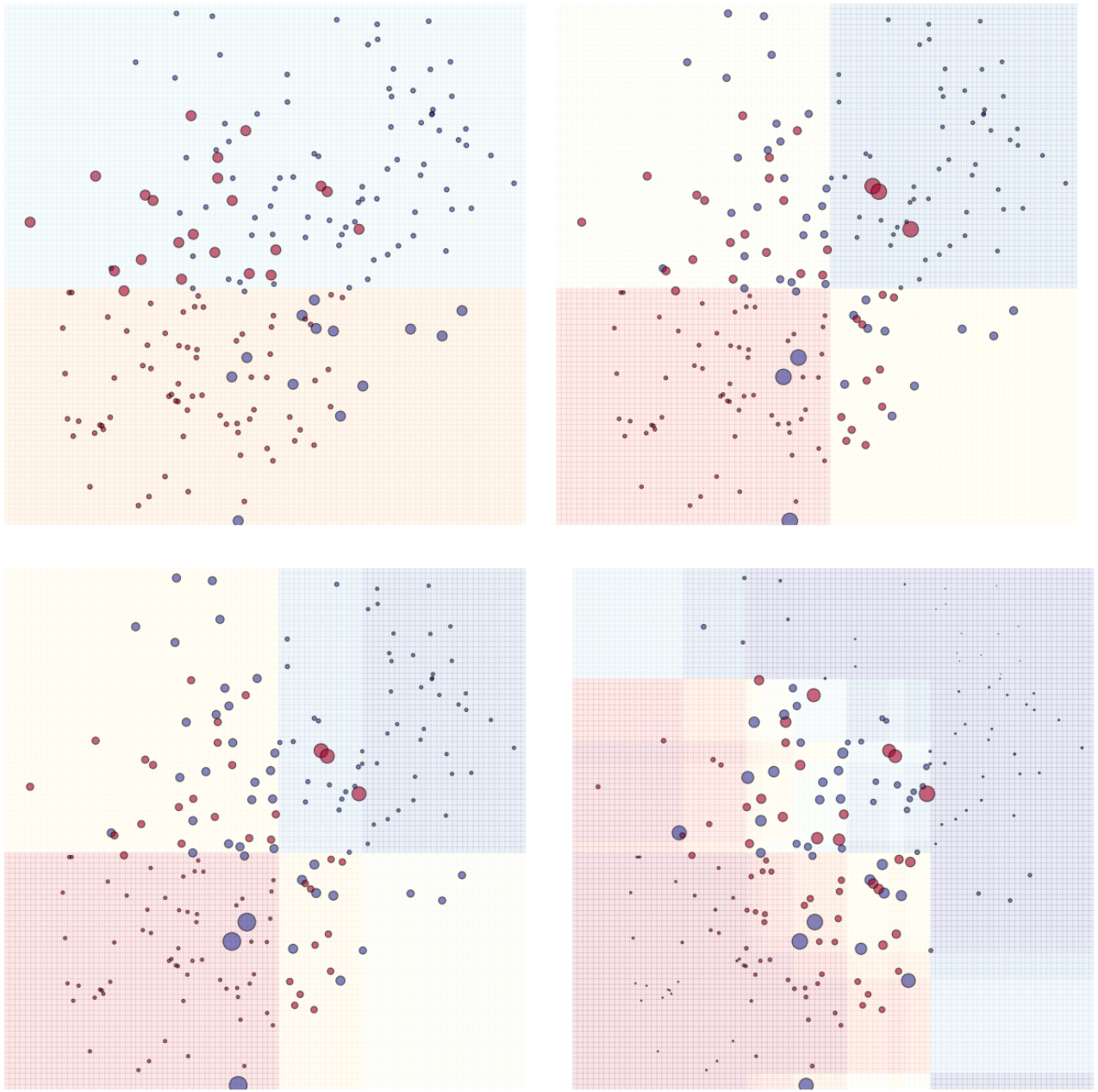
AdaBoost example

Decision trees of depth 1 will be used.









(1, 2, 3, 100 trees)

AdaBoost secret

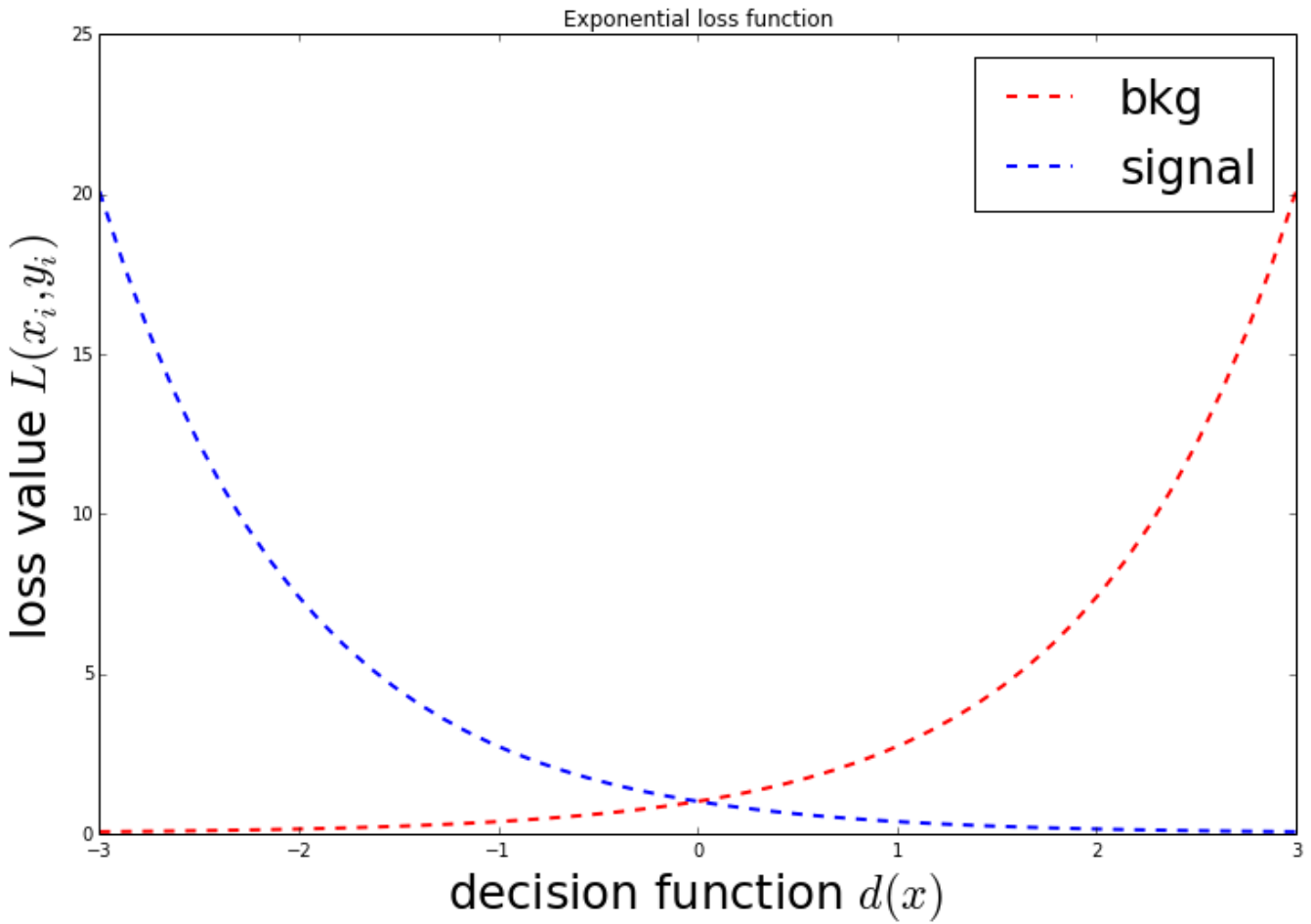
$$D(x) = \sum_j \alpha_j d_j(x)$$

$$\mathcal{L} = \sum_i L(x_i, y_i) = \sum_i \exp(-y_i D(x_i)) \to \min$$

- α_j is obtained as result of analytical optimization
- sample weight is equal to penalty for event

$$w_i = L(x_i, y_i) = \exp(-y_i D(x_i))$$

Loss function of AdaBoost



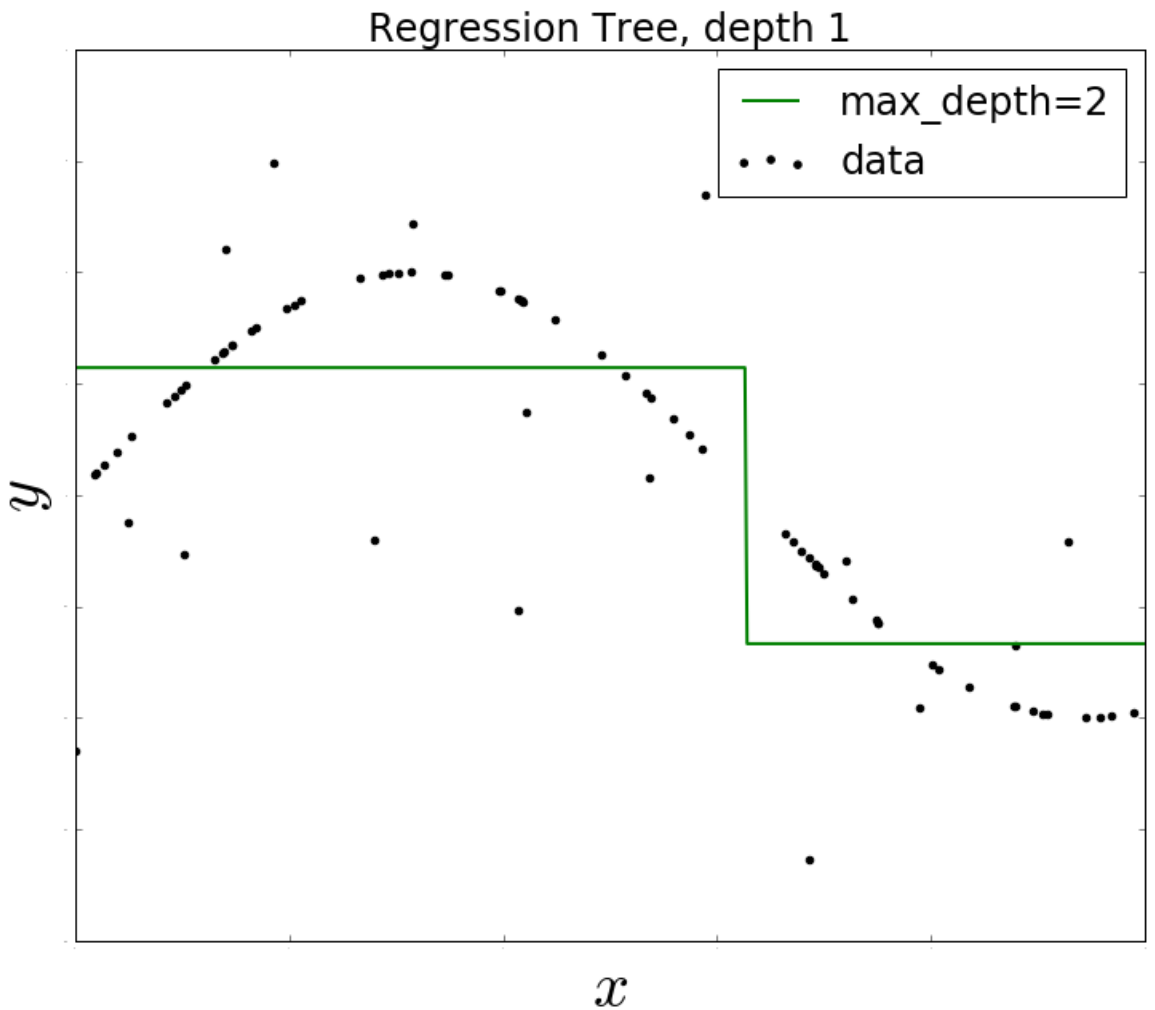
AdaBoost summary

- is able to combine many weak learners
- takes mistakes into account
- simple, overhead is negligible
- too sensitive to outliers

In scikit-learn, one can run AdaBoost over other algorithms.

Gradient Boosting

Decision trees for regression



Gradient boosting to minimize MSE

Say, we're trying to build an ensemble to minimize MSE:

$$\sum_i (D(x_i) - y_i)^2 \to \min$$

When ensemble's prediction is obtained by taking weighted sum

$D(x) = \sum_j d_j(x)$

$D_j(x) = \sum_{j'=1}^{j} d_{j'}(x) = D_{j-1}(x) + d_j(x)$

Assuming that we already built $j - 1$ estimators, how do we train a next one?

Natural solution is to greedily minimize MSE:

$$\sum_i (D_j(x_i) - y_i)^2 = \sum_i (D_{j-1}(x_i) + d_j(x_i) - y_i)^2 \to \min$$

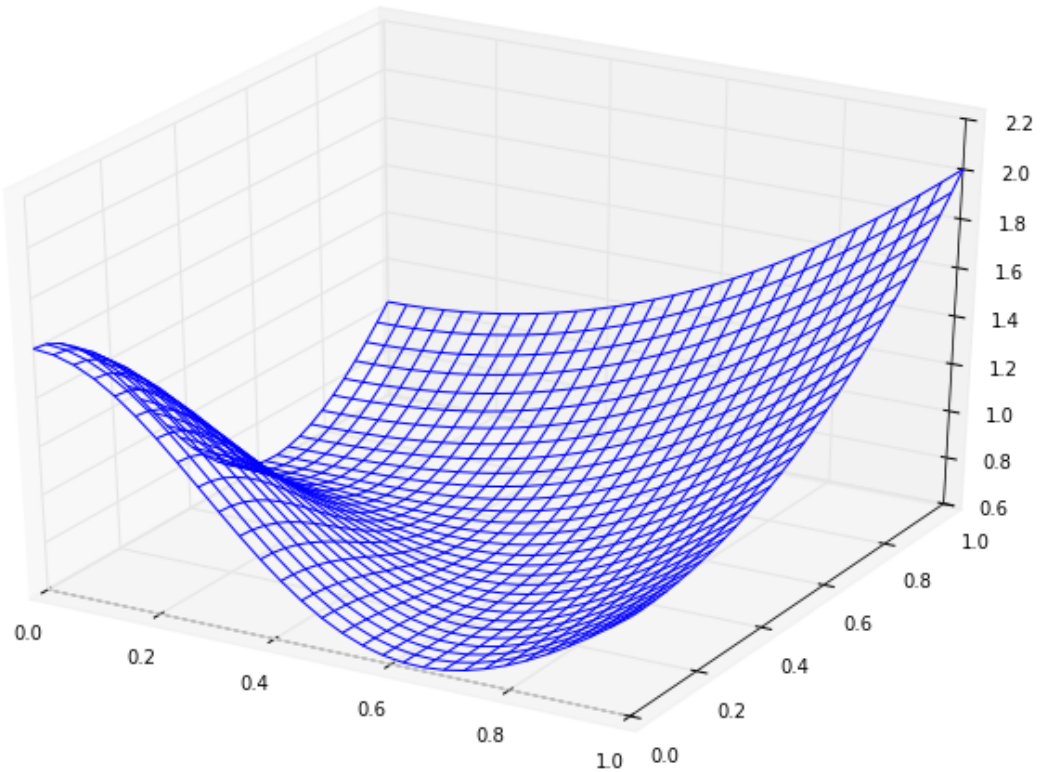
Introduce *residual*: $R_j(x_i) = y_i - D_{j-1}(x_i)$, now we need to simply minimize MSE

$$\sum_i (d_j(x_i) - R_j(x_i))^2 \to \min$$

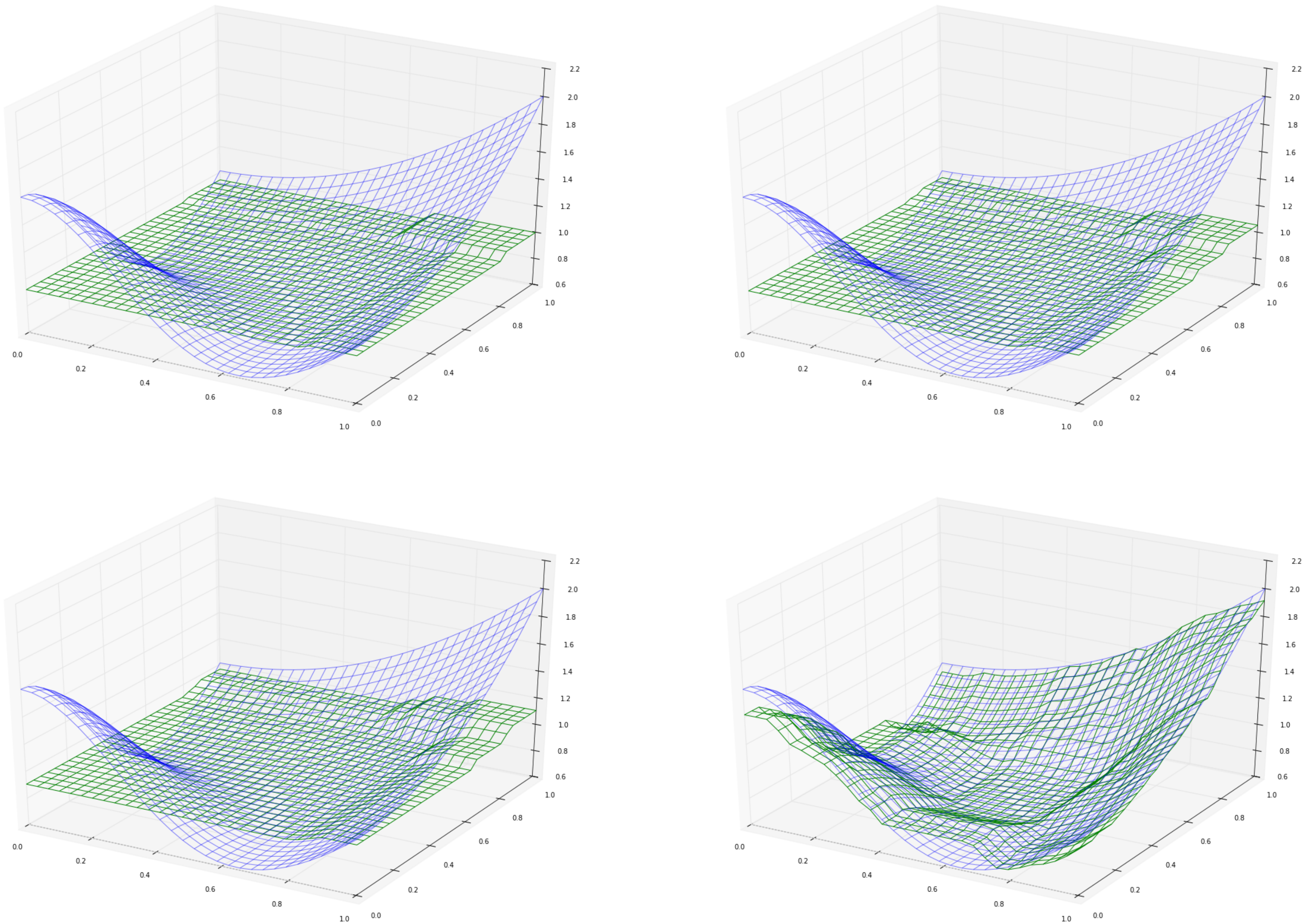
So the jth estimator (tree) is trained using the following data:

$$x_i, R_j(x_i)$$

Example: regression with GB



using regression trees of depth=2



number of trees = 1, 2, 3, 100

Gradient Boosting visualization

Gradient Boosting [Friedman, 1999]

composition of weak regressors,

$$D(x) = \sum_j \alpha_j d_j(x)$$

Borrow an approach to encode probabilities from logistic regression

$$p_{+1}(x) = \sigma(D(x))$$
$$p_{-1}(x) = \sigma(-D(x))$$

Optimization of log-likelihood ($y_i = \pm 1$):

$$\mathcal{L} = \sum_i L(x_i, y_i) = \sum_i \ln\left(1 + e^{-y_i D(x_i)}\right) \to \min$$

Gradient Boosting

$$D(x) = \sum_j \alpha_j d_j(x)$$

$$\mathcal{L} = \sum_i \ln\left(1 + e^{-y_i D(x_i)}\right) \to \min$$

- Optimization problem: find all α_j and weak leaners d_j
- Mission impossible

Gradient Boosting

$$D(x) = \sum_j \alpha_j d_j(x)$$

$$\mathcal{L} = \sum_i \ln\left(1 + e^{-y_i D(x_i)}\right) \to \min$$

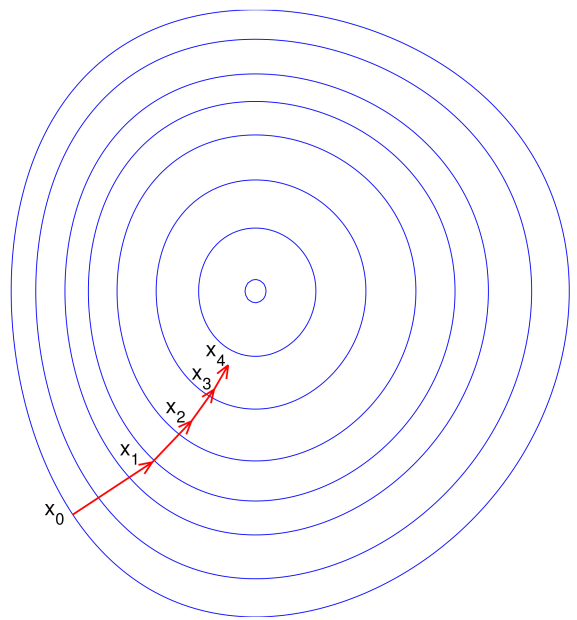
- Optimization problem: find all α_j and weak leaners d_j
- Mission impossible
- Main point: greedy optimization of loss function by training one more weak learner d_j
- Each new estimator follows the gradient of loss function

Gradient Boosting

Gradient boosting ~ steepest gradient descent.

$$D_j(x) = \sum_{j'=1}^{j} \alpha_{j'} d_{j'}(x)$$

$$D_j(x) = D_{j-1}(x) + \alpha_j d_j(x)$$



At *j*th iteration:

- compute *pseudo-residual*
  $$R(x_i) = -\frac{\partial}{\partial D(x_i)} \mathcal{L}\Big|_{D(x)=D_{j-1}(x)}$$
- train regressor d_j to minimize MSE: $\sum_i (d_j(x_i) - R(x_i))^2 \to min$
- find optimal α_j

Important exercise: compute pseudo-residuals for MSE and logistic losses.

Additional GB tricks

to make training more stable, add *learning rate* η:

$$D_j(x) = \sum_j \eta \alpha_j d_j(x)$$

randomization to fight noise and build different trees:

- subsampling of features
- subsampling of training samples

AdaBoost is a particular case of gradient boosting with different target loss function*:

$$\mathcal{L}_{\text{ada}} = \sum_i e^{-y_i D(x_i)} \to \min$$

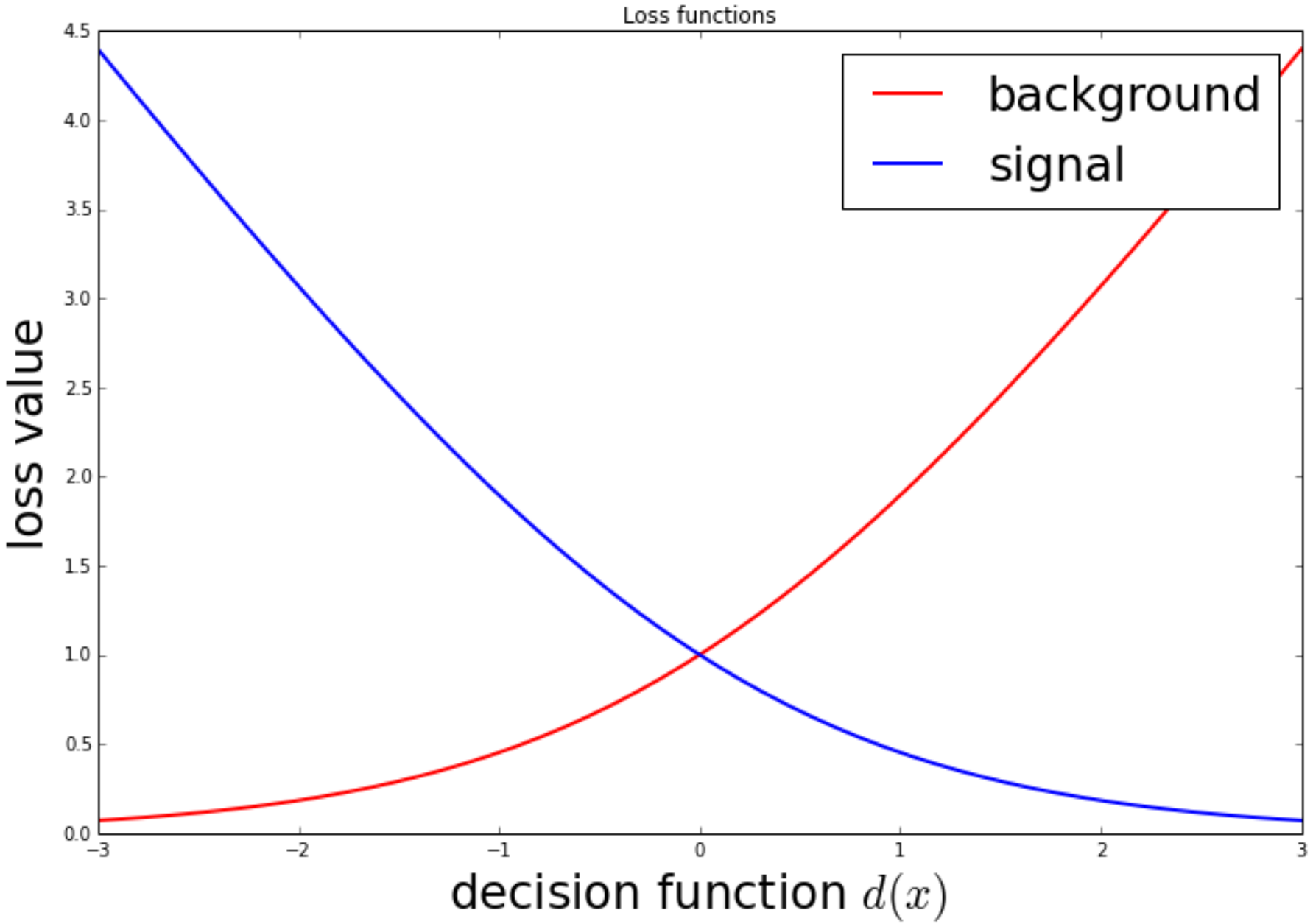
This loss function is called *ExpLoss* or *AdaLoss*.

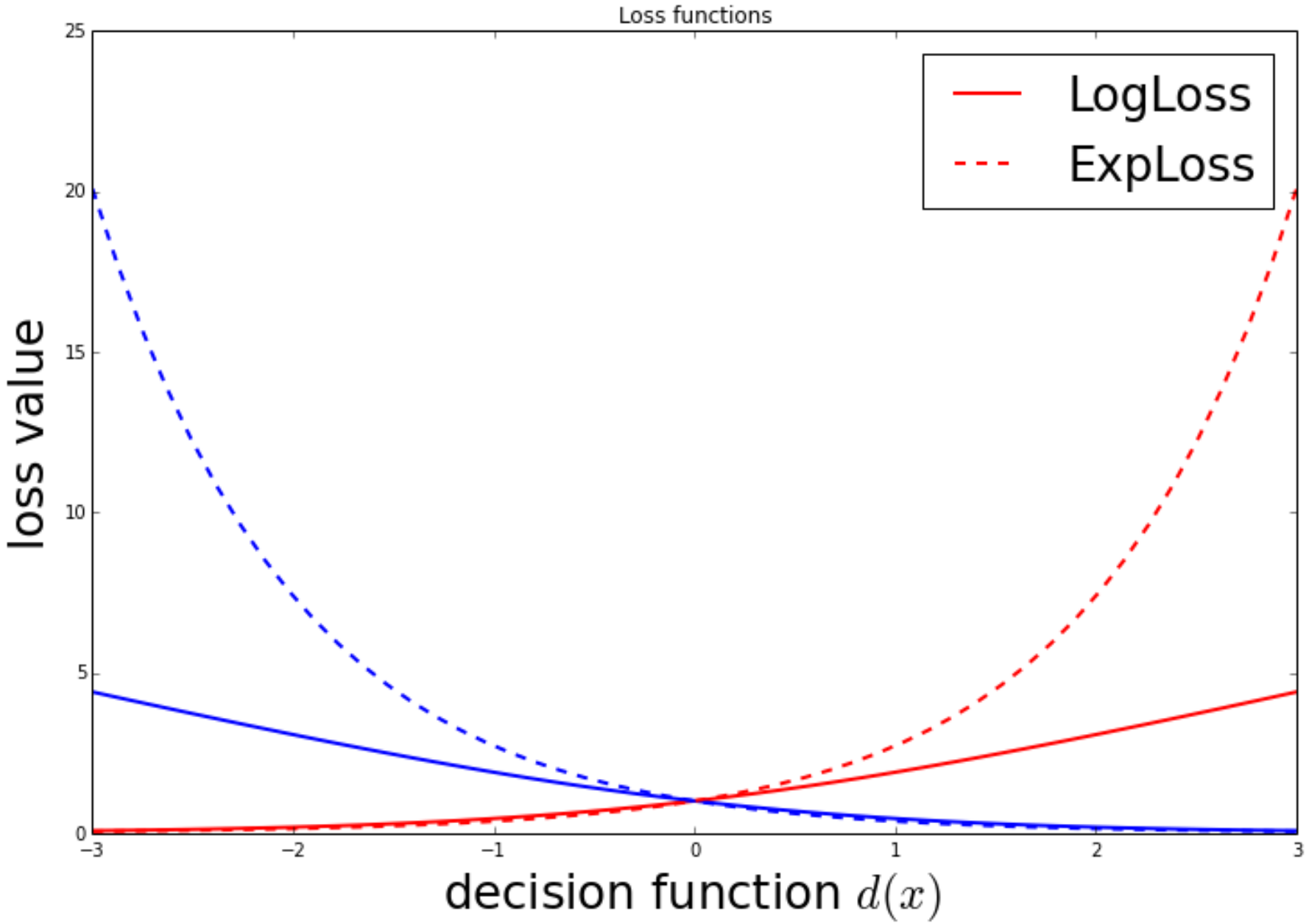
*(also AdaBoost expects that $d_j(x_i) = \pm 1$)

Loss functions

Gradient boosting can optimize different smooth loss function.

- regression, $y \in \mathbb{R}$
  - Mean Squared Error $\sum_i (d(x_i) - y_i)^2$
  - Mean Absolute Error $\sum_i |d(x_i) - y_i|$
- binary classification, $y_i = \pm 1$
  - ExpLoss (ada AdaLoss) $\sum_i e^{-y_i d(x_i)}$
  - LogLoss $\sum_i \log(1 + e^{-y_i d(x_i)})$


Loss functions
background
signal
loss value
decision function $d(x)$
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1
2
3


Loss functions
LogLoss
ExpLoss
loss value
decision function $d(x)$
25
20
15
10
5
0
−3
−2
−1
0
1
2
3

Usage of second-order information

For additive loss function apart from gradient g_i, we can make use of second derivatives h_i.

$$\mathcal{L} = \sum_i L(D_j(x_i), y_i) = \sum_i L(D_{j-1}(x_i) + d_j(x), y_i) \approx$$

$$\approx \sum_i L(D_{j-1}(x_i), y_i) + g_i\, d_j(x) + \frac{h_i}{2} d_j^2(x)$$

E.g. select leaf value using second-order step:

$$\mathcal{L} = \mathcal{L}_{j-1} + \sum_{\text{leaf}} (g_{\text{leaf}} w_{\text{leaf}} + \frac{h_{\text{leaf}}}{2} w_{\text{leaf}}^2) \to \min$$

Using second-order information

$$\mathcal{L} \approx \mathcal{L}_{j-1} + \sum_{\text{leaf}} (g_{\text{leaf}} w_{\text{leaf}} + \frac{h_{\text{leaf}}}{2} w_{\text{leaf}}^2) \to \min$$

$$\text{where} \qquad g_{\text{leaf}} = \sum_{i \in \text{leaf}} g_i, \qquad h_{\text{leaf}} = \sum_{i \in \text{leaf}} h_i.$$

Independent optimization. Explicit solution for optimal values in the leaves:

$$w_{\text{leaf}} = -\frac{g_{\text{leaf}}}{h_{\text{leaf}}}$$

Using second-order information: recipe

On each iteration of gradient boosting

1. train a tree to follow gradient (minimize MSE with gradient)
2. change the values assigned in leaves to:

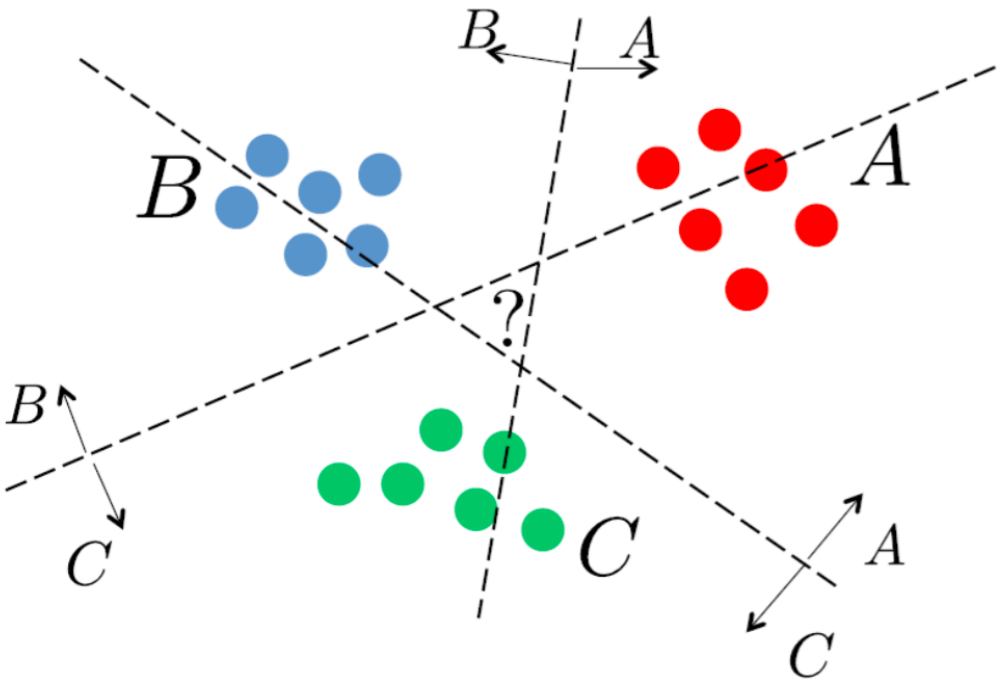
$$w_{\text{leaf}} \leftarrow -\frac{g_{\text{leaf}}}{h_{\text{leaf}}}$$

3. update predictions (no weight for estimator: $a_j = 1$)

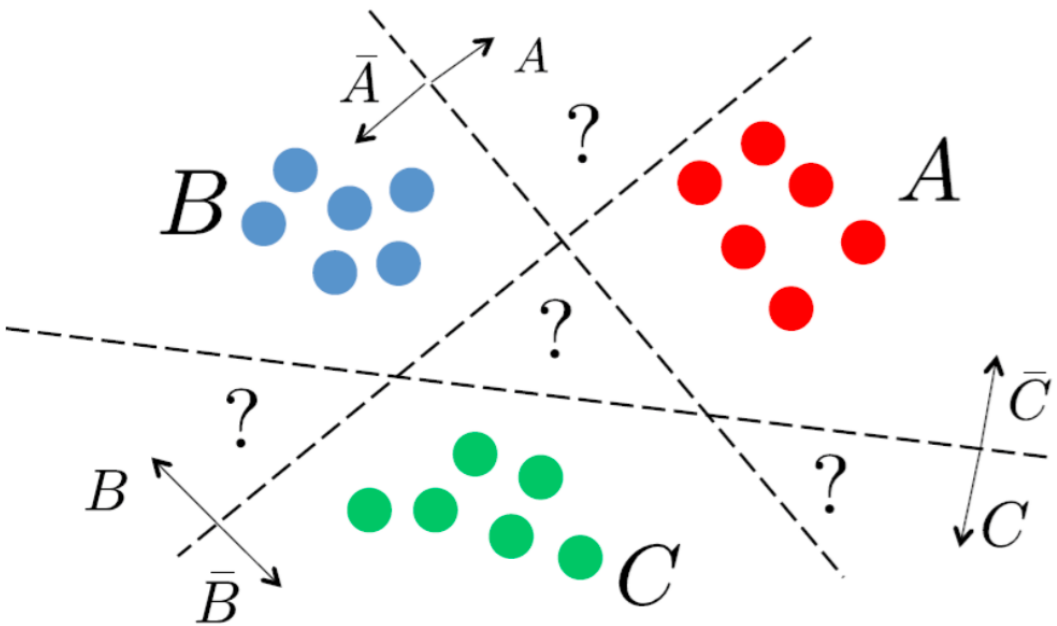
$$D_j(x) = D_{j-1}(x) + \eta d_j(x)$$

This improvement is quite cheap and allows smaller GBs to be more effective.
We can use information about hessians on the tree building step (step 1),

Multiclass classification: ensembling



One-vs-one, $\frac{n_{\text{classes}} \times (n_{\text{classes}} - 1)}{2}$



One-vs-rest, n_{classes}

scikit-learn implements those as meta-algorithms.

Multiclass classification: modifying an algorithm

Most classifiers have natural generalizations to multiclass classification.

Example for logistic regression: introduce for each class $c \in 1, 2, \ldots, C$ a vector w_c .

$$d_c(x) =< w_c, x >$$

Converting to probabilities using *softmax* function:

$$p_c(x) = \frac{e^{d_c(x)}}{\sum_c e^{d_c(x)}}$$

And minimize LogLoss: $\mathcal{L} = \sum_i -\log p_{y_i}(x_i)$

Softmax function

Typical way to convert n numbers to n probabilities.

Mapping is surjective, but not injective (n dimensions to $n-1$ dimension).
Invariant to global shift:

$$d_c(x) \to d_c(x) + \text{const}$$

For the case of two classes: $p_1(x) = \dfrac{e^{d_1(x)}}{e^{d_1(x)} + e^{d_2(x)}} = \dfrac{1}{1 + e^{d_2(x) - d_1(x)}}$

Coincides with logistic function for $d(x) = d_1(x) - d_2(x)$

Loss function: ranking example

In ranking we need to order items by y_i :

$$y_i < y_j \;\; \Rightarrow \;\; d(x_i) < d(x_j)$$

We can penalize for misordering:

$$\mathcal{L} = \sum_{i,\tilde{i}} L(x_i, x_{\tilde{i}}, y_i, y_{\tilde{i}})$$

$$L(x, \tilde{x}, y, \tilde{y}) = \begin{cases} \sigma(d(\tilde{x}) - d(x)), & y < \tilde{y} \\ 0, & \text{otherwise} \end{cases}$$

Adapting boosting

By modifying boosting or changing loss function we can solve different problems

- classification
- regression
- ranking

HEP-specific examples in Tatiana's lecture tomorrow.

n_3 -minutes break

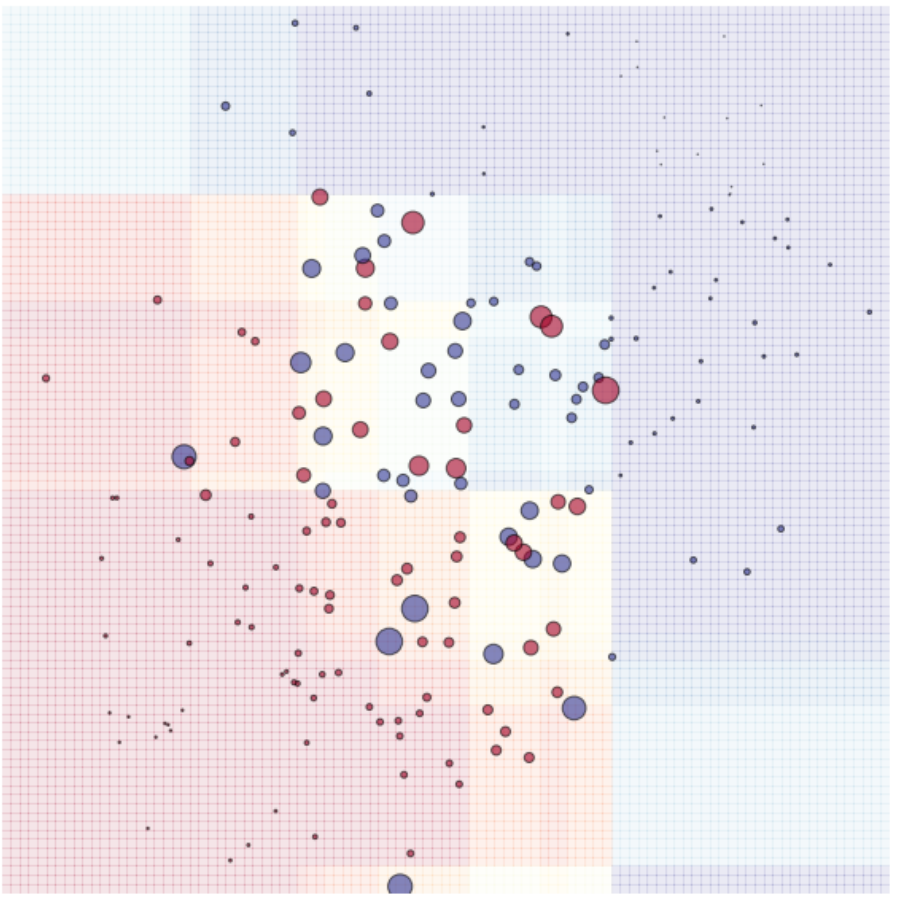
Recapitulation: AdaBoost

Minimizes

$$\mathcal{L}_{\text{ada}} = \sum_i e^{-y_i D(x_i)} \rightarrow \min$$

by increasing weights of misclassified samples:

$$w_i \leftarrow w_i \times e^{-\alpha_j y_i d_j(x_i)}$$



Gradient Boosting overview

A powerful ensembling technique (typically used over trees, GBDT)

- a general way to optimize differentiable losses
  - can be adapted to other problems
- 'following' the gradient of loss at each step
- making steps *in the space of functions*
- gradient of poorly-classified events is higher
- increasing number of trees can drive to overfitting
  (= getting worse quality on new data)
- requires tuning, better when trees are not complex
- widely used in practice

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
  - challenge example: maybe subleading jets matter?
- Validate your guesses

Feature engineering

- Analyzing available features
  - scale and shape of features
- Analyze which information lacks
  - challenge example: maybe subleading jets matter?
- Validate your guesses
- Machine learning is a proper tool for checking your understanding of data

Linear models example

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

Heavy-tailed distributions are harmful, pretransforming required

- logarithm
- power transform
- and throwing out outliers

Same tricks actually help to more advanced methods.

Which transformation is the best for Random Forest?

One-hot encoding

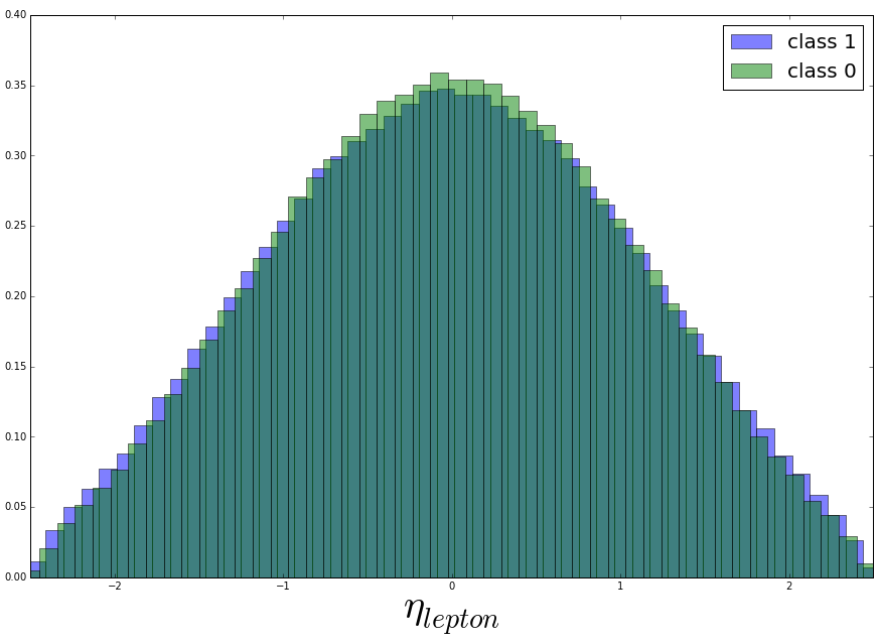
Categorical features (= 'not orderable'), being one-hot encoded, are easier for ML to operate with

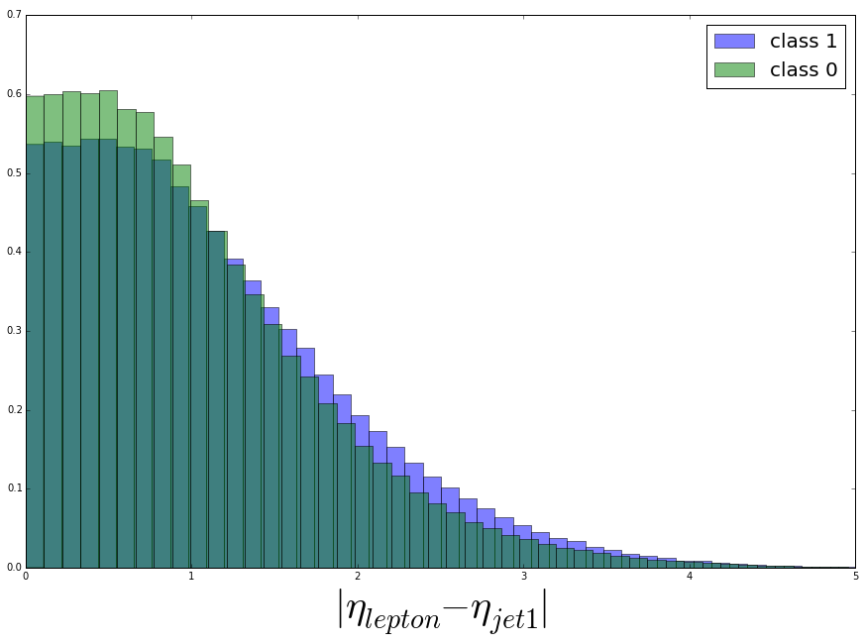
Original data:	
id	Color
1	White
2	Red
3	Black
4	Purple
5	Gold

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.

What about invariant mass of 3 particles?

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.

What about invariant mass of 3 particles? (see Vicens' talk today).

Good features are ones that are explainable by physics. Start from simplest and most natural.

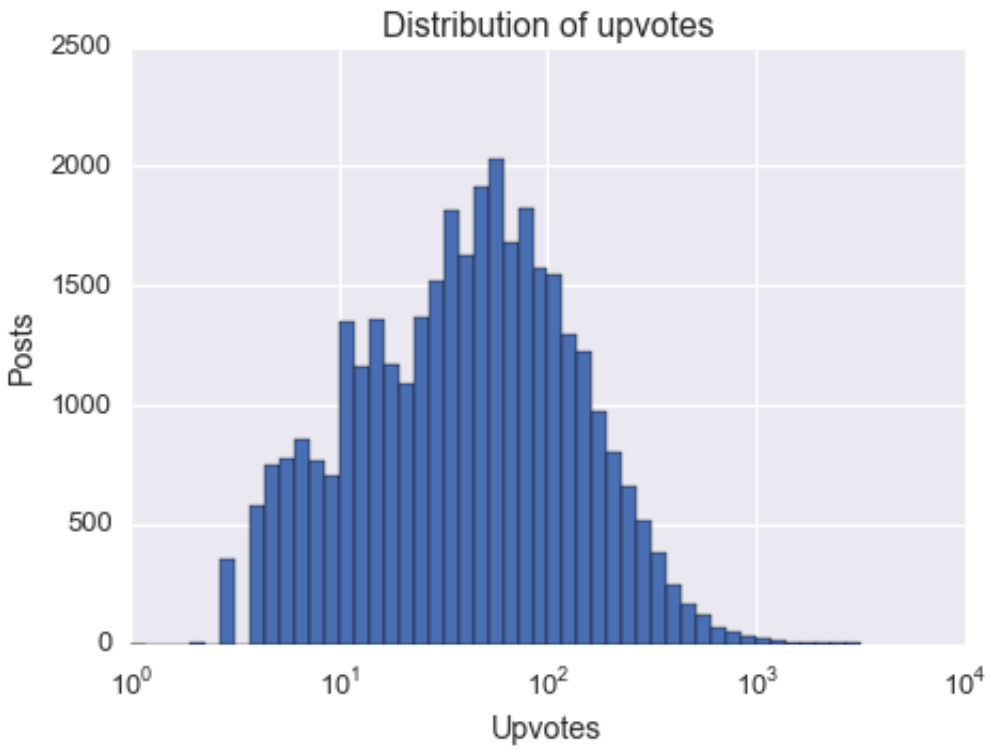
Mind the cost of computing the features.

Output engineering

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

Example: predicting number of upvotes for 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 event (how often we expect this to happen).

Sample weights in some situations also matter.

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

Feature selection

Why?

- speed up training / prediction
- reduce time of data preparation in the pipeline
- help algorithm to 'focus' on finding reliable dependencies
  - useful when amount of training data is limited

Problem:

find a subset of features, which provides best quality

Feature selection

Exhaustive search: 2^d cross-validations.

- incredible amount of resources
- too many cross-validation cycles drive to overly-optimistic quality on the test data

Feature selection

Exhaustive search: 2^d cross-validations.

- incredible amount of resources
- too many cross-validation cycles drive to overly-optimistic quality on the test data
- basic nice solution: estimate importance with RF / GBDT.

Feature selection

Exhaustive search: 2^d cross-validations.

- incredible amount of resources
- too many cross-validation cycles drive to overly-optimistic quality on the test data
- basic nice solution: estimate importance with RF / GBDT.

Filtering methods

Eliminate variables which seem not to carry statistical information about the target.
E.g. by measuring Pearson correlation or mutual information.

Example: all ϕ angles will be thrown out.

Feature selection: embedded methods

Feature selection is a part of training.
Example: L_1 — regularized linear models.

Forward selection

Start from empty set of features.
For each feature in the dataset check if adding this feature improves the quality.

Backward elimination

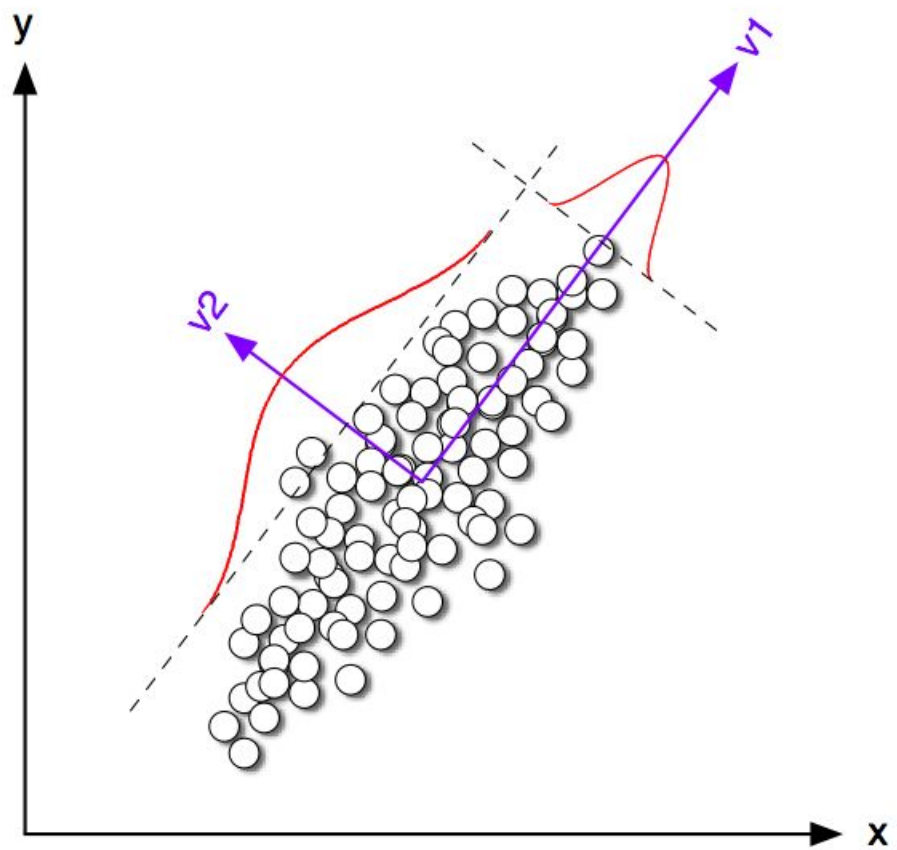
Almost the same, but this time we iteratively eliminate features.

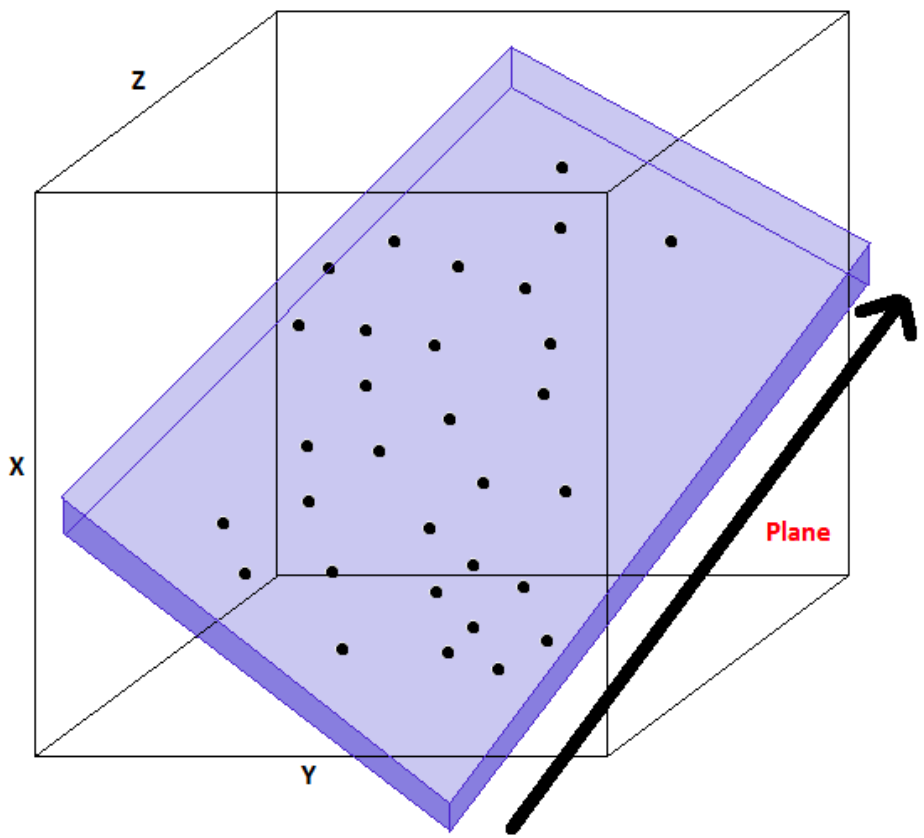
Bidirectional combination of the above is possible, some algorithms can use previously trained model as the new starting point for optimization.

Unsupervised dimensionality reduction

Principal component analysis [Pearson, 1901]

PCA is finding axes along which variance is maximal





PCA description

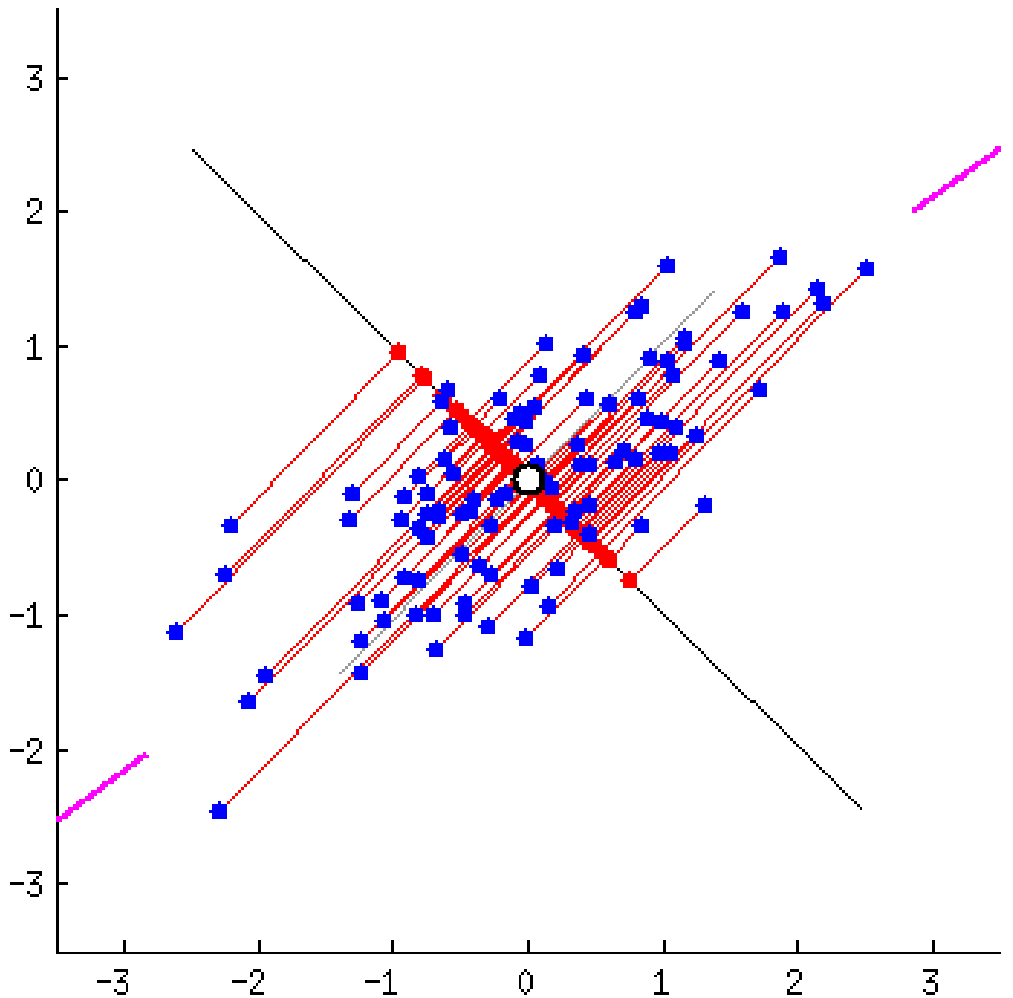
PCA is based on the principal axis theorem

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

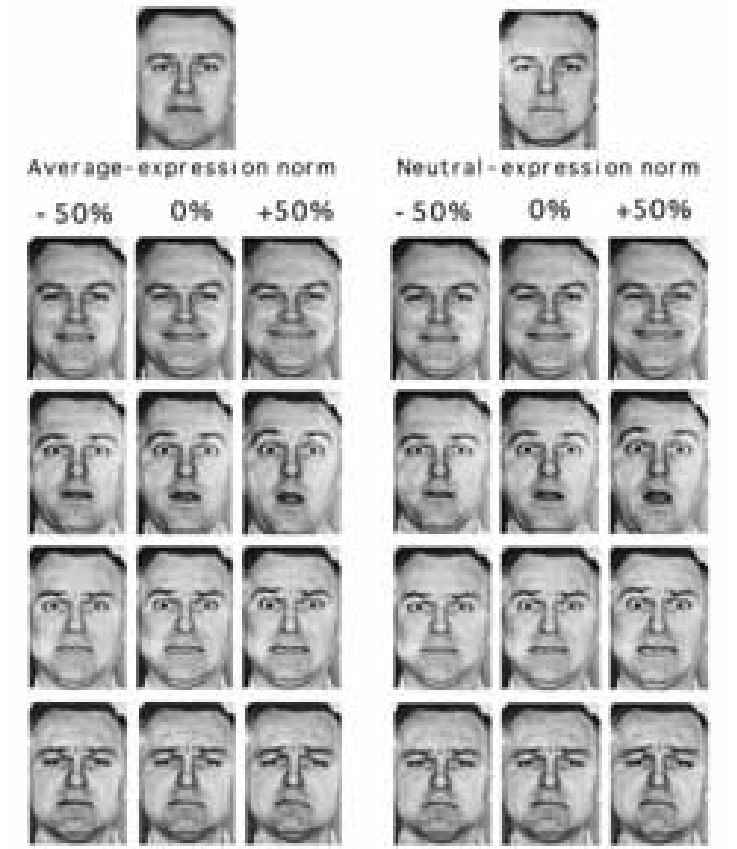
Q is covariance matrix of the dataset, U is orthogonal matrix, Λ is diagonal matrix.

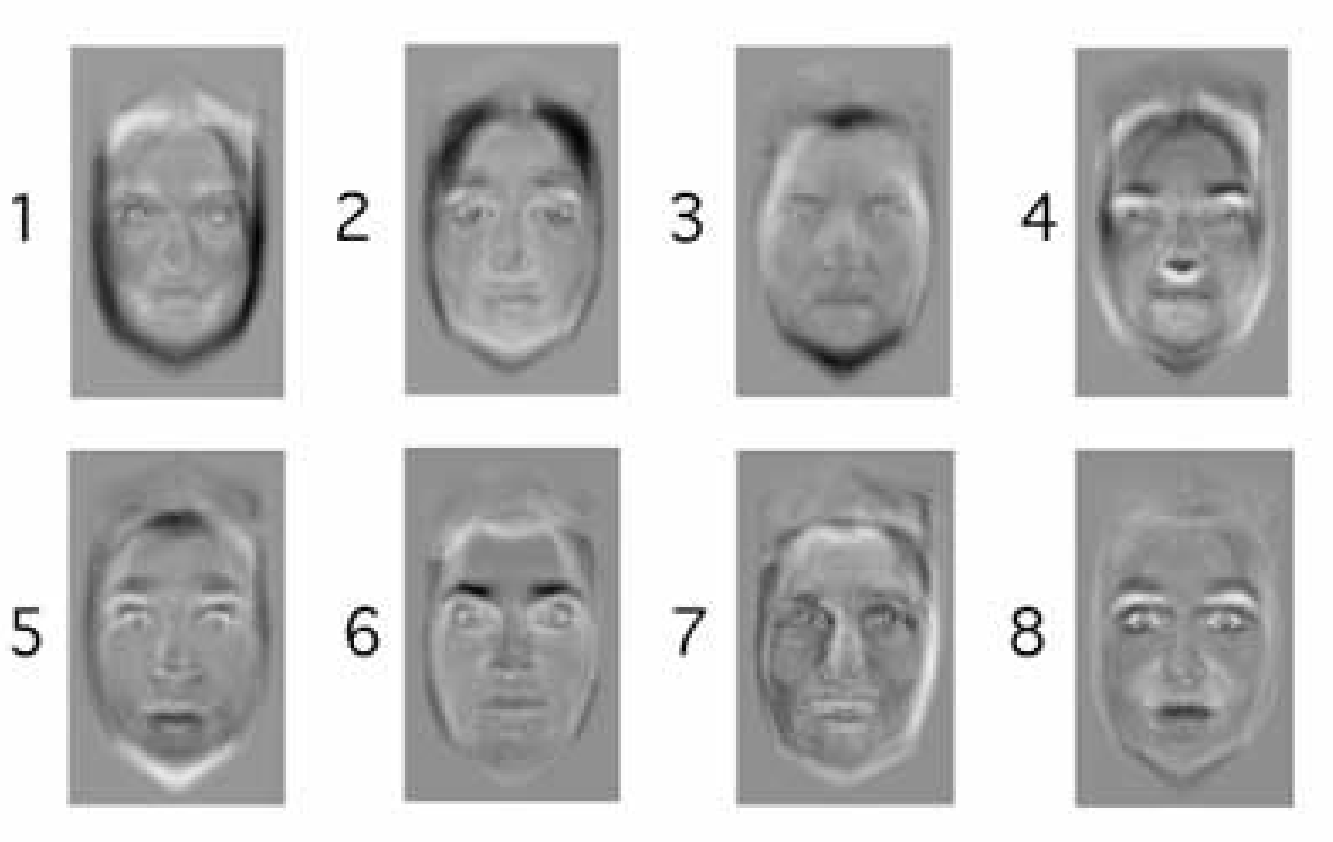
$$\Lambda = diag(\lambda_1, \lambda_2, \dots, \lambda_n), \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$$

PCA optimization visualized



PCA: eigenfaces

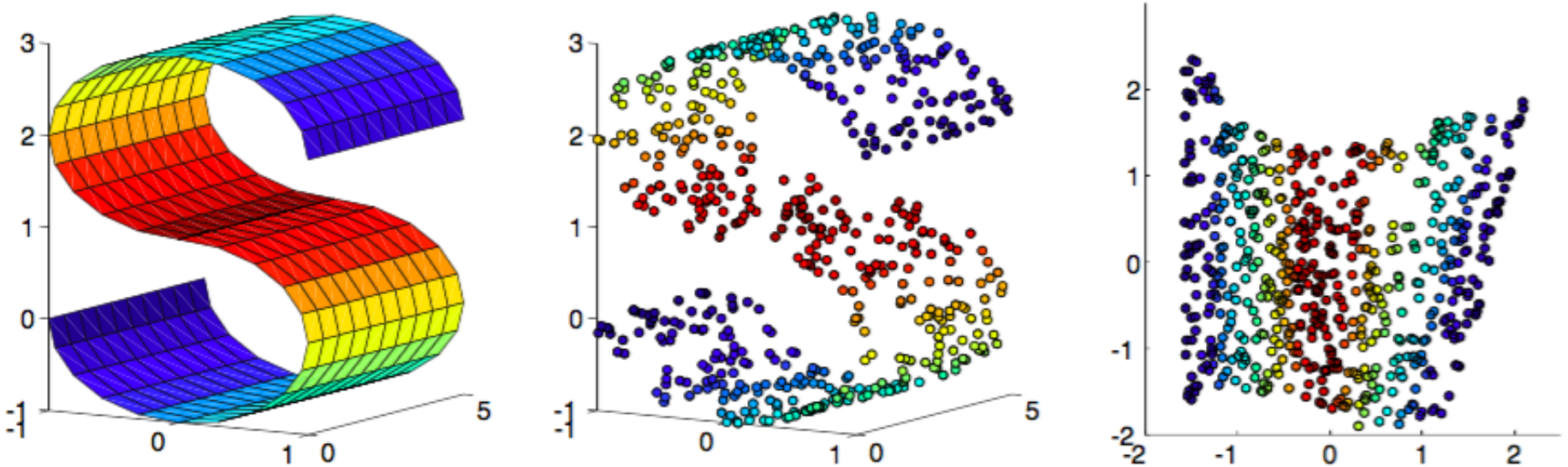




Emotion = α[scared] + β[laughs] + γ[angry]+. . .

Locally linear embedding

handles the case of non-linear dimensionality reduction



Express each sample as a convex combination of neighbours

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

Locally linear embedding

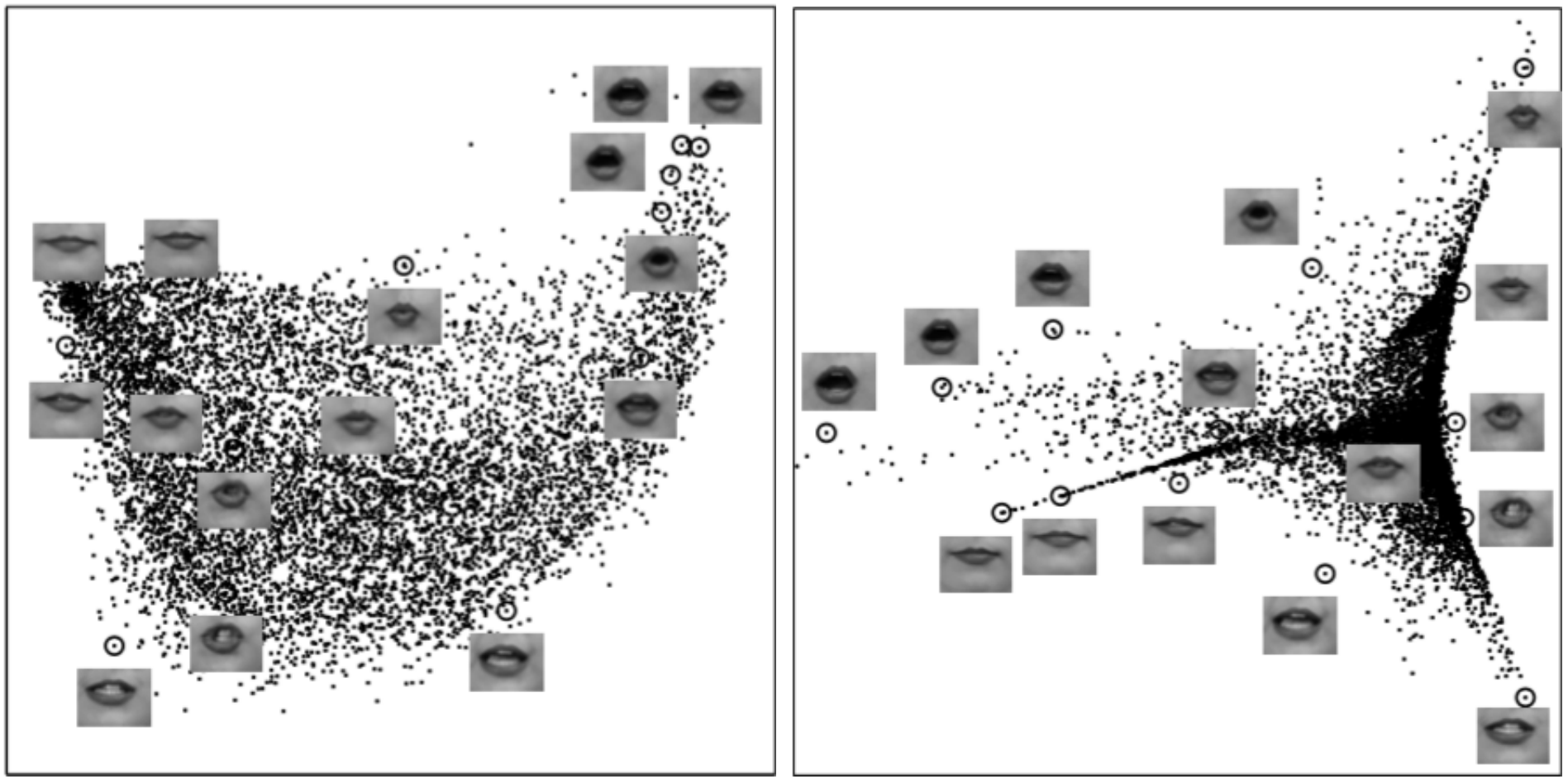
$$\sum_i \left| x_i - \sum_{\tilde{i}} w_{i\tilde{i}} x_{\tilde{i}} \right| \to \min_w$$

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):

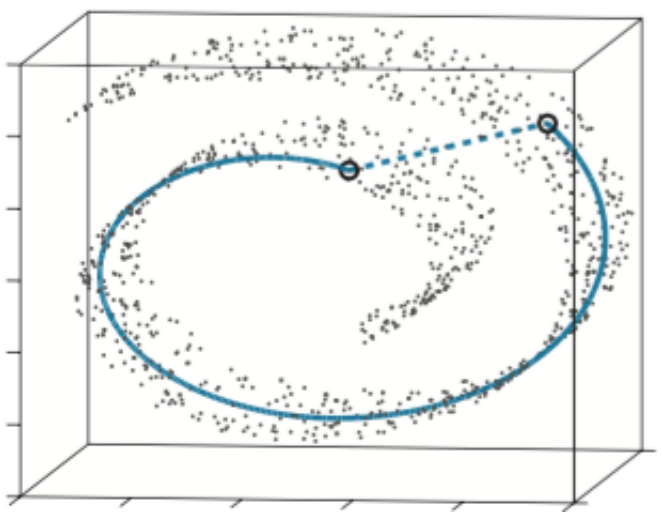
$$\sum_i \left| y_i - \sum_{\tilde{i}} w_{i\tilde{i}} y_{\tilde{i}} \right| \to \min_y$$

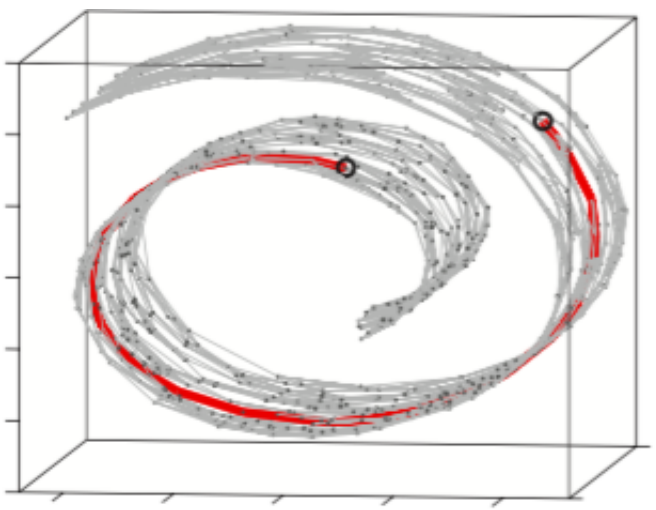
PCA and LLE

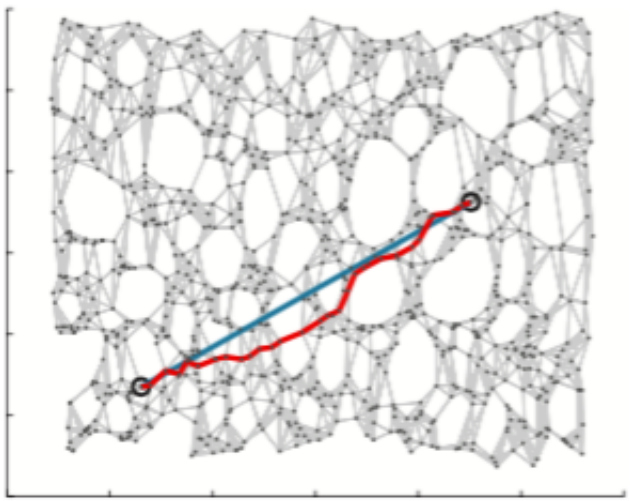


Isomap

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



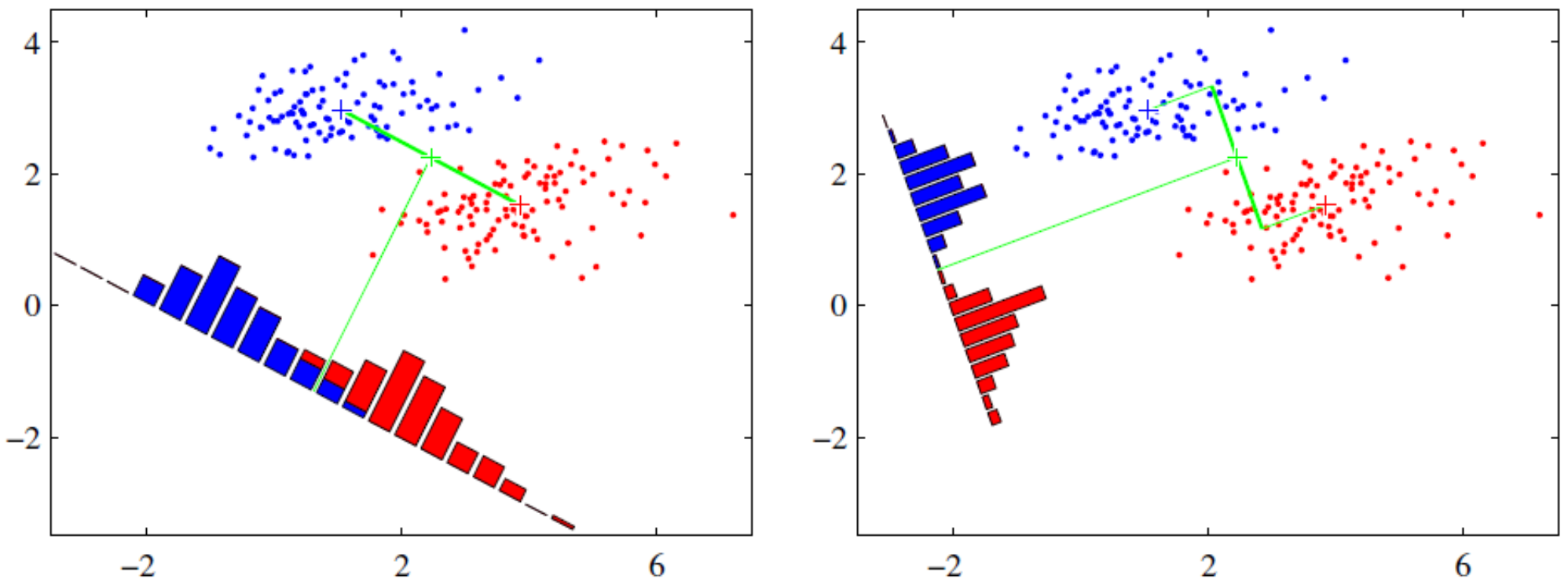




Supervised dimensionality reduction

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

Original idea: find a projection to discriminate classes best



Fisher's LDA

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

$\mu_k =< x >_{\text{events of class c}}$

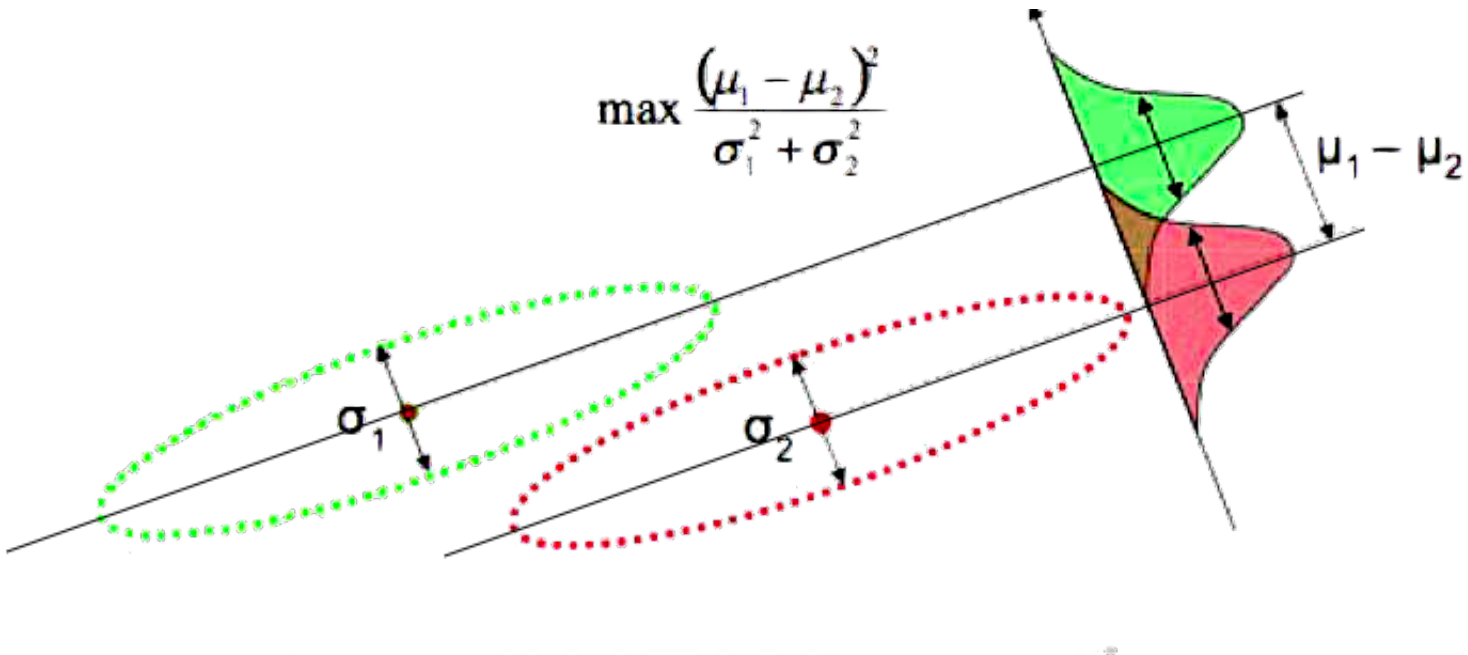
$\sigma_k =< ||x - \mu_k||^2 >_{\text{events of class c}}$

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 maximize a ratio $\dfrac{\sigma_{\text{between}}}{\sigma_{\text{within}}}$

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

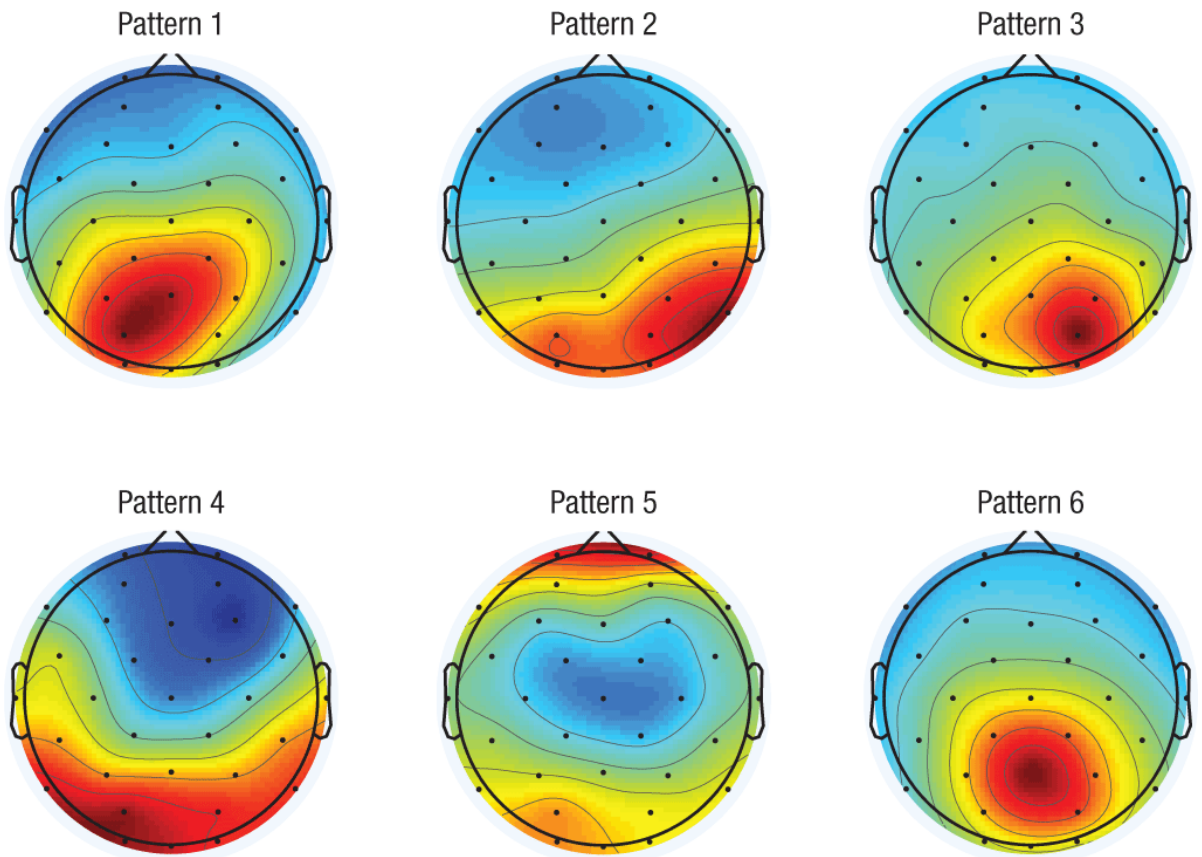
$$\mathrm{tr} W^T \Sigma_{\text{class}} W \to \max$$

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

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

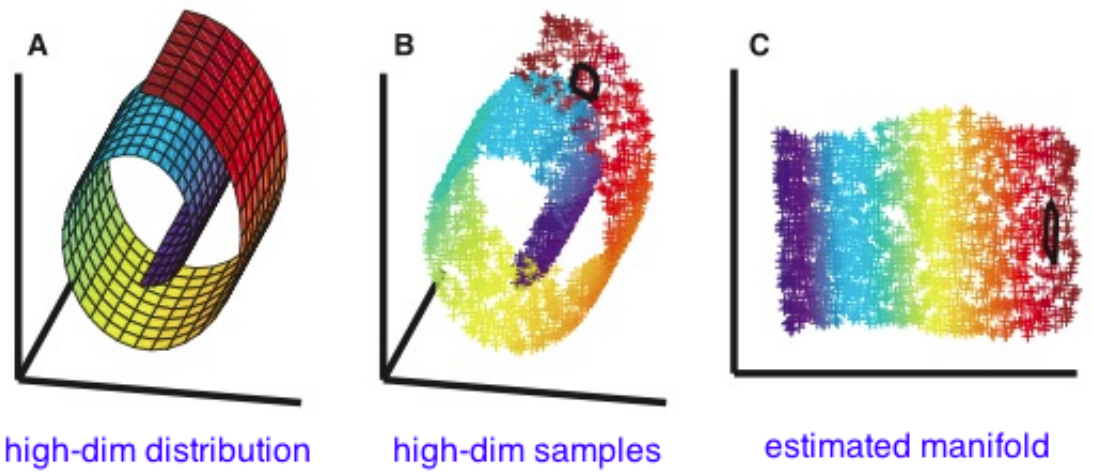
Common spacial patterns

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 same nature



Finding optimal hyperparameters

- some algorithms have many parameters (regularizations, depth, learning rate, ...)
- not all the parameters are guessed
- checking all combinations takes too long

Finding optimal hyperparameters

- some algorithms have many parameters (regularizations, depth, learning rate, ...)
- not all the parameters are guessed
- checking all combinations takes too long

We need automated hyperparameter optimization!

Finding optimal parameters

- randomly picking parameters is a partial solution
- given a target optimal value we can optimize it

Finding optimal parameters

- randomly picking parameters is a partial solution
- given a target optimal value we can optimize it
- no gradient with respect to parameters
- noisy results

Finding optimal parameters

- randomly picking parameters is a partial solution
- given a target optimal value we can optimize it
- no gradient with respect to parameters
- noisy results
- function reconstruction is a problem

Finding optimal parameters

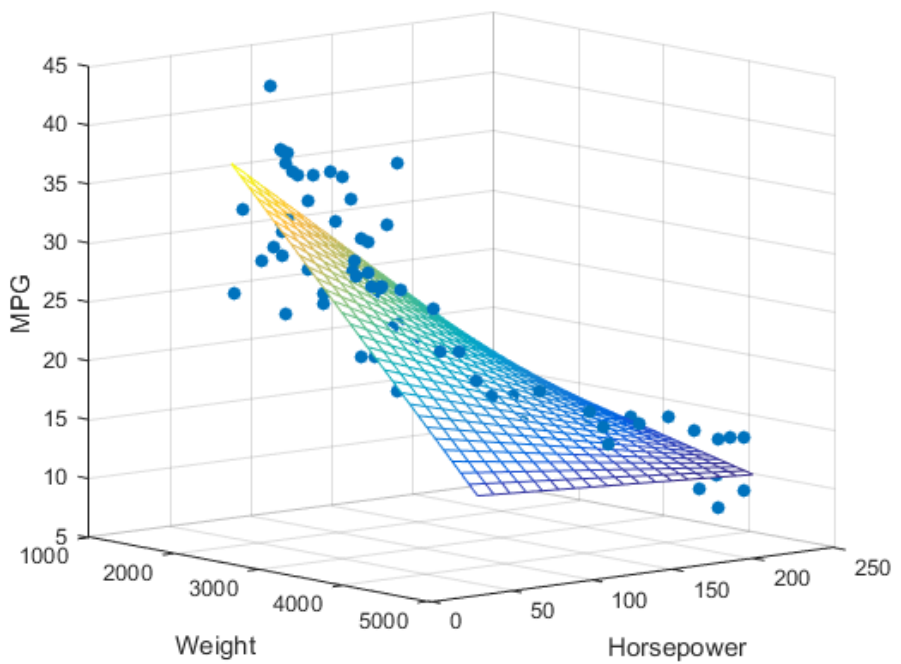
- randomly picking parameters is a partial solution
- given a target optimal value we can optimize it
- no gradient with respect to parameters
- noisy results
- function reconstruction is a problem

Before running grid optimization make sure your metric is stable (i.e. by train/testing on different subsets).

Overfitting (=getting too optimistic estimate of quality on a holdout) by using many attempts a is real issue.

Optimal grid search

- stochastic optimization (Metropolis-Hastings, annealing)
- regression techniques, reusing all known information (ML to optimize ML!)



Optimal grid search using regression

General algorithm (point of grid = set of parameters):

1. evaluations at random points
2. build regression model based on known results
3. select the point with best expected quality according to trained model
4. evaluate quality at this points
5. Go to 2 if not enough evaluations

Why not using linear regression?

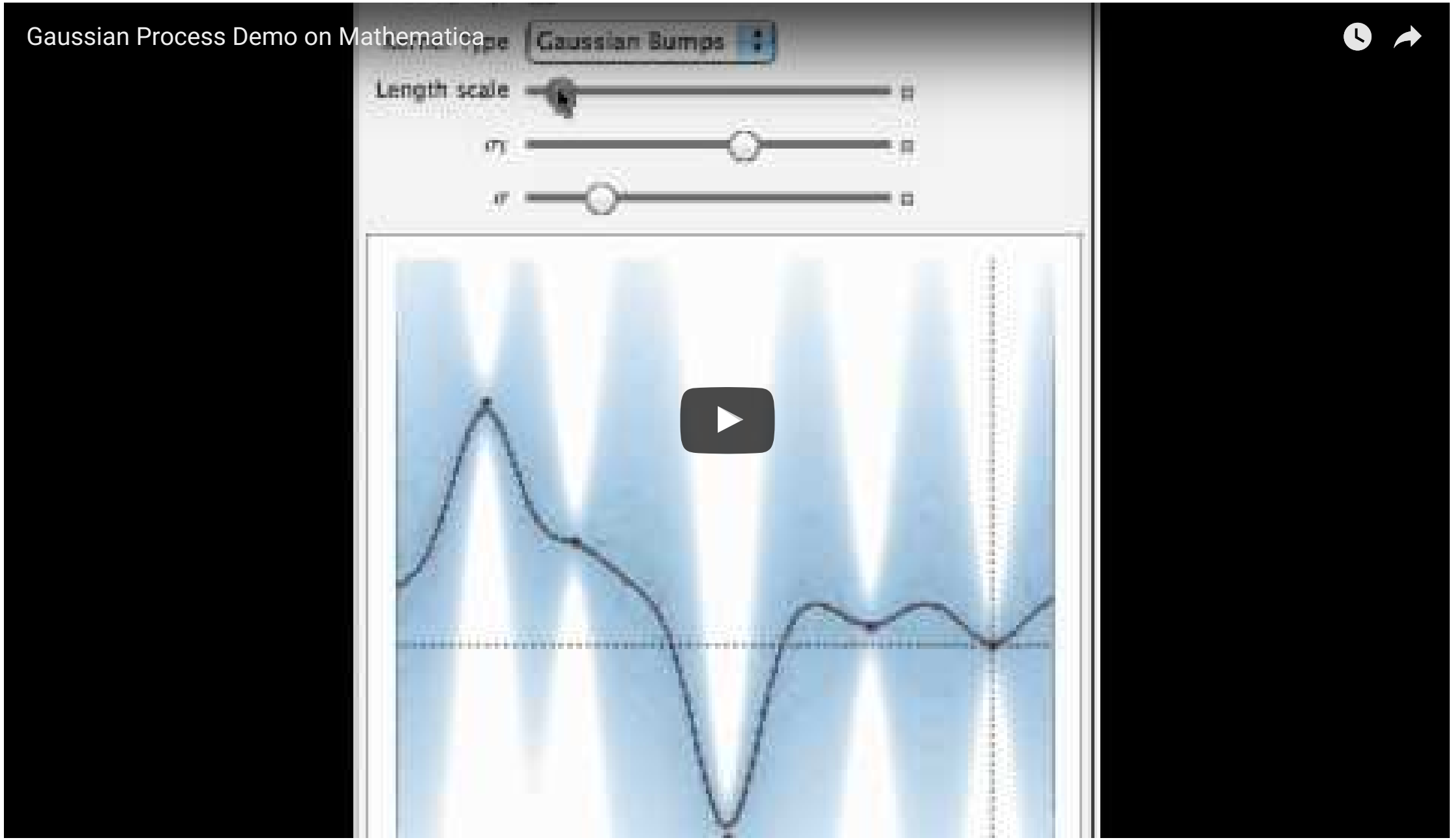
Exploration vs. exploitation trade-off: should we try explore poorly-covered regions or try to enhance currently seen to be optimal?

Gaussian processes for regression

Some definitions: $Y \sim GP(m, K)$, where m and K are functions of mean and covariance: $m(x)$, $K(x, \tilde{x})$

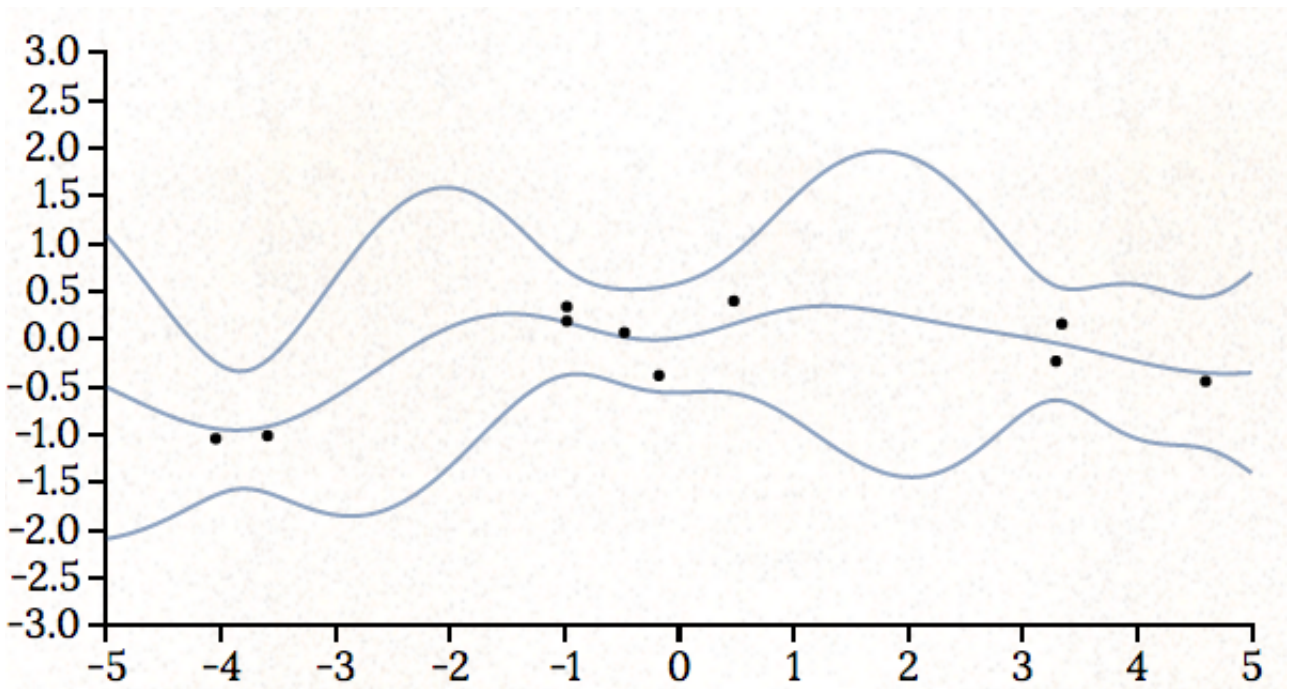
- $m(x) = \mathbb{E}Y(x)$ represents our prior expectation of quality (may be taken constant)
- $K(x, \tilde{x}) = \mathbb{E}Y(x)Y(\tilde{x})$ represents influence of known results on the expectation of values in new points
- RBF kernel is used here too: $K(x, \tilde{x}) = \exp(-c|x - \tilde{x}|^2)$
- Another popular choice: $K(x, \tilde{x}) = \exp(-c|x - \tilde{x}|)$

We can model the posterior distribution of results in each point.


Gaussian Process Demo on Mathematica
Gaussian Bumps
Length scale

Gaussian processes

- Gaussian processes model posterior distribution at each point of the grid.
- we know at which point we have already well-estimated quality
- and we are able to find regions which need exploration. See also this demo.

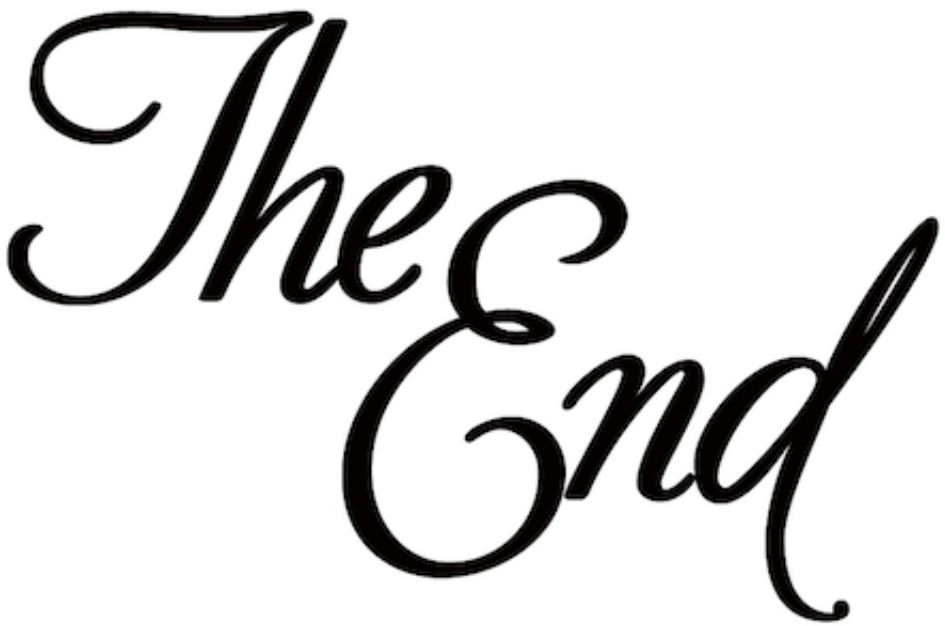


Summary about hyperoptimization

- parameters can be tuned automatically
- be sure that metric being optimized is stable
- mind the optimistic quality estimation (resolved by one more holdout)

Summary about hyperoptimization

- parameters can be tuned automatically
- be sure that metric being optimized is stable
- mind the optimistic quality estimation (resolved by one more holdout)
- *but this is not what you should spend your time on*
  - the gain from properly cooking features / reconsidering problem is much higher


The End