Maschine Learning Summary WS21/22

mastermakrela

February 2, 2022

Contents

Introduction

Machine Learning Principles, methods, and algorithms for learning and prediction on the basis of past evidence

Goal of Machine Learning:

Machines that *learn* to *perform* a *task* from *experience*.

Learning most important aspect
We provide *data* and *goal* - machine figures rest out.

Learning Tools:

- statistics
- probability theory
- decision theory
- information theory
- optimization theory

Core Questions

Task: $y = f(x; w)$

Where:

- x Input
- y Output
- w Learned parameters

Regression	Classification
Continuous output	Discrete output

Core Problems

1. How to input data / how to interpret inputted data
2. Features
    - Invariance to irrelevant input variations
    - Selecting the "right" features is crucial
    - Encoding and use of "domain knowledge"
    - Higher-dimensional features are more discriminative.
3. Curse of Dimensionality
    - complexity increases exponentially with number of dimensions

Core Questions

1. Measuring performance of a model
    - eg. % of correct classifications
2. Generalization performance
    - performing on test data is not enough
    - model has to perform on new data
3. What data is available?
    - Supervised vs unsupervised learning
    - mix: semi-supervised
    - reinforcement learning - with feedback

Most often learning is an optimization problem

I.e., maximize $y = f(x; w)$ with regard to performance.

Bayes Decision Theory

Probability Theory

Probability

$$P(X = x) \in [0, 1] \tag{1}$$

Where $X \in \{x_1, ..., x_N\}$ an Occurence of something.

Assuming two random variables $X \in \{x_i\}$ and $Y \in \{y_i\}$. Consider N trials and let:

$n_{ij} = count\{X = X_i \wedge Y = yj\}$

$c_i = count\{X = x_i\}$

$r_i = count\{Y = y_j\}$

Then we can derive *The Rules of Probability*:

Joint Probability	$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$
Marginal Probability	$p(X = x_i) = \frac{c_i}{N}$
Conditional Probability	$p(Y = y_j \| X = x * i) = \frac{n * ij}{c_i}$
Sum Rule	$p(X) = \sum_Y p(X, Y)$
Product Rule	$p(X, Y) = p(X, Y)P(X)$

And *Bayes' Theorem*:

$$p(X|Y) = \frac{p(X|Y)P(X)}{P(X)} \tag{2}$$

where: $p(X) = \sum_Y p(X, Y)p(Y)$

Probability Densities

If the variable is continuous we can't just look at probability for x. We have to look for the interval the x is in, using *Probability Density Function* $p(x)$.

$p(x) = \int_a^b p(x)dx$

The probability that x lies in the interval is given by $(-\infty, z)$ the *cumulative distribution function*:

$P(z) = \int_{-\infty}^z p(x)dx$

Expectations

Expectation average value of some function $f(x)$ under a probability distribution $p(x)$

Discrete: $\mathbb{E}[f] = \sum_x p(x)f(x)$

Continuous: $\mathbb{E}[f] = \int p(x)f(x)dx$

For finite N samples we can approximate:

$\mathbb{E}[f] \backsimeq \frac{1}{N} \sum_{n=1}^N f(x_n)$

Conditional expectation:

$\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x)$

Variances and Covariances

Variance measures how much variability there is in $f(x)$ around its mean value $\mathbb{E}[f(x)]$

$var[f] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2$

Covariance for two random variables x and y

$cov[x, y] = \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y]$

Bayes Decision Theory

Priors a priori probabilities

what can we tell about probability before seeing the data

sum of all priors is 1

Conditional probabilities $p(x|C_k)$ is **likelihood** for class C_k

where x measures/describes certain properties of input

Posterior probabilities $p(C_k|x)$

probability of class C_k given the measurement vector x

$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} = \frac{p(x|C_k)p(C_k)}{\sum_i p(x|C_i)p(C_i)}$

Goal: Minimize the probability of a misclassification.

Decide for C_k when:

$p(C_k|x) > p(C_j|x) \forall j \neq k$

$p(x|C_k)p(C_k) > p(x|C_j)p(C_j) \forall j \neq k$

Classifying with Loss Functions

Motivation: Decide if it's better to choose wrong or nothing.

In general formalized as a matrix L_{jk}

$L_{jk} = loss for decision C_j if C_k is correct selection$

Minimizing the Expected Loss

Optimal solution requires knowing which class is correct - *this is unknown*. So we **minimize the expected loss**:

$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(x, C_k) dx$

This can be done by choosing the $\mathcal{R}_j$ regions such that:

$\mathbb{E}[L] = \sum_k L_{kj} p(C_k|x)$

Probability Density Estimation

How can we estimate (= learn) those probability densities?

In Supervised training case: data and class labels are known. So we can estimate the probability density for each class separately.

The Gaussian (or Normal) Distribution

One-dimensional

- Mean μ
- Variance σ^2

$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}$

Multi-dimensional

- Mean μ
- Variance Σ

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \quad (3)$$

Properties

Central Limit Theorem "The distribution of the sum of N i.i.d. random variables becomes increasingly Gaussian as N grows."

`[There was some more stuff in slides but it was boring math]`

The marginals of a Gaussian are again Gaussians

Parametric Methods

Given some data X with parameters $\theta = (\mu, \sigma)$. What is the probability that X has been generated from a probability density with parameters θ.

$L(\theta) = p(X|\theta)$

Maximum Likelihood Approach

Single data point:

$p(x_n|\theta) = \frac{1}{\sqrt{2\pi\sigma}} exp\{-\frac{(x_n-\mu)^2}{2\sigma^2}\}$

Assumption all data points are independent:

$L($
$theta) = p(X|\theta) = \prod_{n=1}^{N} p(x_n|\theta)$

Log-Likelihood: $E(\theta) = -lnL(\theta) = -\sum_{n=1}^{N} lnp(x_n|\theta)$

Goal: Minimize $E(\theta)$.

How to:

1. Take the derivate of $E(\theta)$
2. Set it to zero
3. Quic Mafs

Warning

MLA is *biased* - it underestimates the true variance. I.e., *overfits to the observed data.*

Frequentist vs Bayesian

Frequentist	Bayesian
probabilities are frequencies of random, repeatable events	quantify the uncertainty about certain states or events

	Frequentist	Bayesian
	fixed, but can be estimated more precisely when more data is available	uncertainty can be revised in the light of new evidence

Non-Parametric Methods

Often the functional form of the distribution is unknown. So we have to estimate probability from data.

Histograms

Partition data into bins with widths Δ_i and count number of observations in each bin.

$p_i = \frac{n_i}{N\Delta_i}$

Often: $\Delta_i = \Delta_j \forall i, j$

Advantages	Disadvantages
works in any dimension D	curse of dimensionality
no need to store data after computation	rather brute force

Bin size:

- too large: too much smoothing
- too small: too much noise

Kernel Density Estimation

Parzen Window

Idea: Hypercube of dimension D with width edge length h.

We place a kernel window k at location x and count how many data points fall inside it. Crude solution because the chosen function k creates hard cuts around Hypercubes.

Kernel Function: $k(u) = \begin{cases} 1, |u_i * \frac{1}{2}, i = 1, ..., D \\ 0, else \end{cases}$

$K = \sum_{i=1}^{N} k(\frac{x - x_n}{h})$

$V = \int k(u)du = h^d$

Then probability density is:

$p(x) \simeq \frac{K}{NV} = \frac{1}{Nh_D} \sum_{i=1}^{N} k(\frac{x - x_n}{h})$

Gausian Kernel

Similar to Parzen Window, but with Gaussian kernel function k.

$k(u) = \frac{1}{(2 \times \pi h^2)^{1/2}} \exp\{-\frac{u^2}{2h^2}\}$

$K = \sum_{i=1}^{N} k(x - x_n)$

$V = \int k(u)du = 1$

Then probability density is:

$p(x) \simeq \frac{K}{NV} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi h^2)^{D/2}} \exp\{-\frac{\|x - x_n\|^2}{2h^2}\}$

K-Nearest Neighbors

Similar to above but we fix K (the number of neighbors) and calculate V (size of the neighbourhood).

Then: $p(x) \simeq \frac{K}{NV}$

Warning: Strictly speaking, the model produced by K-NN is not a true density model, because the integral over all space diverges.

Bayesian Classification

$p(C_j|x) = \frac{p(x|C_j)p(C_j)}{p(x)}$

Summary

- Very General
- Training requires no computation
- Requires storing and computing the entire dataset -> cost linear in the number of data points -> can be saved in implementation

Kernel size K in K-NN?

- Too large: too much smoothing
- Too small: too much noise

Bias-Variance Tradeoff

	Histograms	Kernel methods	K-Nearest Neighbors
	bin size? Δ	*kernel size?* h	K*?*
too large	Δ : too smoth	h : too smooth	K : too smooth
too small	Δ : not smooth enough	h : not smooth enough	K : not smooth enough

Mixture Distribution

Motivation: single parametric distribution is often not sufficient, e.g., for multimodal data.

Mixture of Gaussians

Idea: we take multiple Gaussians and combine them, giving each one a weight that also depends on input.

$p(x|\theta) = \sum_{j=1}^{N} p(x|\theta_j)p(j)$

Where:

- $p(x|\theta_j)$ is the probability of measurement x given the j-th mixture component
- $p(j)$ is the prior of component j

  Note:

  $\int p(x)dx = 1$

Maximum Likelihood

$E = -lnL(\theta) = -\sum_{n=1}^{N} lnp(x_n|\theta)$

$lnp(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N}\{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)\}$

-> leads to infinite loop, because gausians depend on gausians -> It is possible to apply iterative numerical optimization here, but in the following, we will see a simpler method.

K-Means Clustering

Iterative procedure:

1. Pick K random data points as initial cluster centres
2. Assign each data point to the closest cluster centre
3. Compute the new cluster centres as the mean of the assigned data points
4. Repeat until convergence

Guaranteed to converge after finite number of iterations

But:

- local optimum
- depends on the initial cluster centres

  Note:

  Can be used e.g. for image compression

Advantages	Disadvantages
simple — fast to compute	how to select k
converges to local minimum	sensitive to initial cluster centres
	sensitive to outliers
	only spherical clusters

EM Algorithm

Expectation-maximization algorithm

E-Step	softly assign samples to mixture components
M-Step	update the parameters of the mixture components

Technical Advice

- When implementing EM, we need to take care to avoid singularities in the estimation!
  => Enforce minimum width for the Gaussians
- EM is very sensitive to the initialization - will converge to local minimum of E => Initialize with k-Means to get better results
- Typical procedure
  - Run k-Means M times (e.g. M = 10-100)
  - Pick the best result (lowest error J).
  - Use this result to initialize EM
    - Set μ_j to the corresponding cluster mean from k-Means.
    - Initialize Σ_j, to the sample covariance of the associated data points.

Summary: Gaussian Mixture Models

Properties:

- Very general, can represent any (continuous) distribution.
- Once trained, very fast to evaluate.
- Can be updated online.

Problems:

- Need to apply regularization in order to avoid singularities
- EM for MoG is computationally expensive

- Need to select the number of mixture components K
    - Model selection problem

Applications

Computer Vision

- Model distributions of pixel colors.
- Each pixel is one data point in, e.g., RGB space.
- Learn a MoG to represent the class-conditional densities.
- Use the learned models to classify other pixels.

Background Model for Tracking
Train background MoG for each pixel

- Model "common" appearance variation for each background pixel
- Initialization with an empty scene.
- Update the mixtures over time
    - Adapt to lighting changes, etc.

Anything that cannot be explained by the background model is labelled as foreground

Image Segmentation
User assisted image segmentation

- User marks two regions for foreground and background.
- Learn a MoG model for the color values in each region.
- Use those models to classify all other pixels.
    - Simple segmentation procedure (building block for more complex applications)

Support Vector Machines

Softmax Regression

Multi-class generalization of logistic regression

Assume binary labels $t_n \in \{0, 1\}$.

Softmax generalizes thisto K values in 1-of-K notation.

$$y(x; w) = \begin{bmatrix} P(y = 1|x; u) \\ P(y = 2|x; w) \\ \vdots \\ P(y = k|x; W) \end{bmatrix} \tag{4}$$

With *softmax* function: $\frac{exp(a_k)}{\sum_j exp(a_j)}$

Logistic Regression
Alternative way of writing the cost function.

$E(w) = \sum_{n=1}^{N} \sum_{k=0}^{1} \{\mathbb{I}(t_n = k)\} \ln P(y_n = k|x_n; w)\}$

Softmax Regression
Generalization to K classes using indicator functions.

$E(w) = \sum_{n=1}^{N} \sum_{k=0}^{1} \{\mathbb{I}(t_n = k)\} \ln \frac{exp(w_k^T x)}{\sum_{j=1}^{K} exp(w_j^T x)}\}$

Optimization

Again, no closed-form solution is available — Resort again to Gradient Descent.

$\nabla_{w_k} E(w) = -\sum_{n=1}^{N} [\mathbb{I}(t_n = k) \ln P(y_n = k|x_n; w)]$

We can now plug this into a standard optimization package.

Support Vector Machines

Motivation

Goal: predict class labels of new observations.

But: as training progresses we start to overfit to training data.

Popular solution: Cross-validation.

- Split the available data into training and validation sets.
- Estimate the generalization error based on the error on the validation set.
- Choose the model with minimal validation error.

With linearly separable data there are many ways to split the data. The problem is how to choose the best split?

Intuitively, we would like to select the classifier which leaves maximal "safety room" for future data points.

This can be obtained by maximizing the margin between positive and negative data points.

=> The SVM formulates this problem as a convex optimization problem. I.e., we can find optimal solution.

SVM

Consider linearly separable data:

- N training points $\{(x_i, y_i)\}_{i=1}^{N}$ $x_i \in \mathbb{R}^d$
- Target values $t_i \in \{-1, 1\}$
- Hyperplane separating the two classes: $w^T x + b = 0$

Then Canonical representation of the decision hyperplane:

$t_n(w^T x_n + b) \geq 1 \forall n$

Optimization problem

Find the hyperplane satisfying: $arg_{w,b} min \frac{1}{2}\|w\|^2$

under constraints: $t_n(w^T x_n + b) \geq 1 \forall n$

- Quadratic programming problem with linear constraints.
- Can be formulated using Lagrange multipliers (see slides).

Lagrangian Formulation

$L_p = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n\{t_n y(x_n) - 1\}$

Under conditions:

- $a_n \geq 0$
- $t_n y(x_n) - 1 \geq 0$
- $a_n\{t_n y(x_n) - 1\} = 0$

Solution

Computed as a linear combination of the training examples: $w = \sum_{n=1}^{N} a_n t_n x_n$.

Because of the KKT conditions, the following must also hold: $a_n(t_n(w^T x_n + b) - 1) = 0$.

This implies that $a_n > 0$ only for training data points for which $t_n(w^T x_n + b) - 1 = 0$.

=> *Only some of the data points actually influence the decision boundary!*

To define the decision boundary, we still need to know b:

$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} (t_n - \sum_{m \in \mathcal{S}} a_m t_m x_m^T x_n)$

Discussion

Linear SVM

- Linear classifier
- SVMs have a "guaranteed" generalization capability.
- Formulation as convex optimization problem.

=> Globally optimal solution!

Primal form formulation

- Solution to quadratic prog. problem in M variables is in $\mathcal{O}(M^3)$.
- Here: D variables -> $\mathcal{O}(D^3)$.
- Problem: scaling with high-dim. data ("curse of dimensionality")

Dual form formulation

`Does something that I don't understand.`

At the end we should maximize:

$L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_m^T x_n)$

Under conditions: $a_n \geq 0 \forall n$ and $\sum_{n=1}^{N} a_n t_n = 0$.

Then the hyperplane is given by the $N_{\mathcal{S}}$ support vectors:

$w = \sum_{n=1}^{\mathcal{S}} a_n t_n x_n$

Soft-margin classification

Solution above works only if data linearly separable. But we can add some tolerance to this division to make it work even if outliers present.

We add "*slack variable*" ξ to the formulation:

$w^T x_n + b \geq 1 - \xi$ for $t_n = 1$ $w^T x_n + b * -1 + \xi$ for $t_n = -1$

where $\xi_n \geq 0 \forall n$.

Interpertation

- $\xi_n = 0$: point on correct side of the margin
- $\xi_n = |t_n - y(x_n)|$: otherwise
  - $\xi_n > 1$: misclassified point

  Note:

  We do not have to set the slack variables ourselves! They are jointly optimized together with w.

New Dual Formulation

$L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_m^T x_n)$

Under conditions: $a_n ? 0 ? C$ and $\sum_{n=1}^{N} a_n t_n = 0$.

Nonlinear Support Vector Machines

Not everything can be linearly separated, so we need nonlinear classifiers.

General idea:

The original input space can be mapped to some higher-dimensional feature space where the training set is separable.

Nonlinear transformation ϕ of the data points x_n:

$x \in \mathbb{R}^D \ \phi : \mathbb{R}^D \to \mathcal{H}$

Hyperplane in higher dimensional space $\mathcal{H}$:

$w^T \phi(x) + b = 0$

Problem with High-dim. Basis Functions

In order to apply the SVM, we need to evaluate the function: $y(x) = w^T \phi(x) + b$.

Using hyperplane $w = \sum_{n=1}^{N} a_n t_n \phi(x_n)$.

Which leads to Problems in high dimensional feature space.

Solution:

We can replace dot product $\phi(x)^T \phi(x)$ by a kernel function: $k(x, y)$.

Then $y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b$.

The kernel function implicitly maps the data to the higher dimensional space (without having to compute $\phi(x)$ explicitly)!

But it only works fore specific kernel functions.

"Every positive definite symmetric function is a kernel." Mercer's theorem (modernized version)

(positive definite = all eigenvalues are > 0)

Example kernels:

- Polynomial kernel: $k(x, y) = (x^T y + 1)^p$
- Radial Basis Function kernel (e.g. Gaussian): $k(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$

New New Dual Formulation

$L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_m, x_n)$

Under conditions: $a_n ? 0 ? C$ and $\sum_{n=1}^{N} a_n t_n = 0$.

Summary

Properties

- in practice work very well
- among the best performers for a number of classification tasks ranging from text to genomic data
- can be applied to complex data types by designing kernel functions for such data
- The kernel trick has been used for a wide variety of applications. It can be applied wherever dot products are in use

Limitaitons

- How to select the right kernel?
  - Best practice guidelines are available for many applications
- How to select the kernel parameters?
  - (Massive) cross-validation.
  - Usually, several parameters are optimized together in a grid search.
- Solving the quadratic programming problem
  - Standard QP solvers do not perform too well on SVM task.
  - Dedicated methods have been developed for this, e.g. SMO.
- Speed of evaluation
  - Evaluating $y(x)$ scales linearly in the number of SVs.
  - Too expensive if we have a large number of support vectors. -> There are techniques to reduce the effective SV set.
- Training for very large datasets (millions of data points)
  - Stochastic gradient descent and other approximations can be used

Error Function

SVMs result in so-called *hinge error*. Where the error is minimized and correct classification is constant.

This leads to:

- sparsity - Zero error for points outside the margin
- robustness - Linear penalty for misclassified points

Applications

Text Classification

Problem: Classify a document in a number of categories.

Representation:

- "Bag-of-words" approach
- Histogram of word counts (on learned dictionary)

Usage:

- spam filters
- ocr - optical character recognition
- object detection

Adaboost

Ensembles of Classifiers

Idea

- Assume K classifiers.
- They are independent.
- Each has error probability $p < 0.5$ on training data.

Then: Majority vote of all classifiers should have a lower error than each individual classifier

Constructing Ensembles

How do we get different classifiers?

Simplest Case

1. Subsample the training data
   Reuse the same training algorithm several times on different subsets of the training data.
2. Train same classifier on different data.

=> Well-suited for "unstable" learning algorithms.

Where:

Unstable	Stable
Decision trees	Nearest neighbor
neural networks	linear regression
rule learning algorithms	SVMs

Unstable := small differences in training data can produce very different classifiers

Bagging

Bagging := "Bootstrap aggregation"

1. In each run of the training algorithm, randomly select M samples with replacement from the full set of N training data points.
2. If $M = N$, then on average, 63.2% of the training points will be represented. The rest are duplicates.

Injecting randomness

- Many (iterative) learning algorithms need a random initialization (e.g. k-means, EM)
- Perform multiple runs of the learning algorithm with different random initializations.

Model Averaging

- Suppose we have H different models $h = 1, ..., H$ with prior probabilities $p(h)$.
- Construct the marginal distribution over the data set: $p(X) = \sum_{h=1}^{H} p(X|h)p(h)$

Interpretation

- Just one model is responsible for generating the entire data set.
- The probability distribution over h just reflects our uncertainty which model that is.
- As the size of the data set increases, this uncertainty reduces, and $p(X|h)$ becomes focused on just one of the models.

Model Combination

- (e.g., Mixtures of Gaussians)
- Different data points generated by different model components.
- Uncertainty is about which component created which data point.

-> One latent variable z_n, for each data point: $p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{z_n} p(x_n, z_n)$

Model Averaging: Expected Error

```
Some math I don't understand. See slides 19-21 in VL 12.
```

Average error of committee

$\mathbb{E}_{COM} = \frac{1}{M}\mathbb{E}_{AV}$

This suggests that the average error of a model can be reduced by a factor of M simply by averaging M versions of the model!

In reality the errors are **not** uncorrelated - usually highly correlated.

Adaboost

Idea

- Iteratively select an ensemble of component classifiers
- After each iteration, reweight misclassified training examples.
  - Increase the chance of being selected in a sampled training set.
  - Or increase the misclassification cost when training on the full set.

Components

$h_m(x)$ "weak" / base classifier
$H(x)$ "strong" / final classifier

Adaboost

Construct a strong classifier as a thresholded linear combination of the weighted weak classifiers:

$H(x) = sign(\sum_{m=1}^{M} \alpha_m h_m(x))$

Minimizing Exponential Error

Exponential error function: $E = \sum_{n=1}^{N} exp\{-t_n f_m(x_n)\}$

where $f_m(x)$ is a classifier defined as a linear combination of base classifiers $h_l(x)$:

$f_m(x) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l h_l(x)$

Goal:

Minimize E with respect to both the weighting coefficients α_l and the parameters of the base classifiers $h(x)$.

`Here some step by step math that minimizes this.`

Final Algorithm

1. Initailization: $w_n^{(1)} = \frac{1}{N}$ for $n = 1, ..., N$.
2. For $m = 1, ..., M$ iterations

   a) Train a new weak classifier $h_m(x)$ using the current weighting coefficients $W(m)$ by minimizing the weighted error function:
   $J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)$

   b) Estimate the weighted error of this classifier on X:
   TODO

   c) TODO

   d) TODO

Summary

Properties

- Simple combination of multiple classifiers.
- Easy to implement.
- Can be used with many different types of classifiers.
  - None of them needs to be too good on its own.
  - In fact, they only have to be slightly better than chance.
- Commonly used in many areas.
- Empirically good generalization capabilities.

Limitations

- Original AdaBoost sensitive to misclassified training data points.
    - Because of exponential error function.
    - Improvement by GentleBoost
- Single-class classifier
    - Multiclass extensions available

Deep Learning