Maschinelles Lernen - Uebung 00

Sebastian Buschjäger

Technische Universität Dortmund - Fakultät Informatik - Lehrstuhl 8

October 22, 2018

Structure of this course

Goals

→ Learn the basics of Applied Machine Learning
→ Offer a place to discuss questions
→ Prepare you for the exam

Structure of this course

Goals

→ Learn the basics of Applied Machine Learning
→ Offer a place to discuss questions
→ Prepare you for the exam

Structure

- There are no requirements to enter the exam (keine Studienleistung)
- There will be weekly exercises
- We will (probably) not publish sample solutions
- I will try to give a quick recap about the lecture each week
- We discuss what you deem necessary / interesting

Recap: Notation

Note The input space can be (nearly) everything
Make things easier focus on d–dimensional vectors $\vec{x} \in \mathcal{X} \subseteq \mathbb{R}^n$

$\mathcal{D}$	Feature 1	Feature 2	$\dots$	Feature d	Label
Example 1	x_{11}	x_{12}	$\dots$	x_{1d}	y_1
Example 2	x_{21}	x_{22}	$\dots$	x_{2d}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
Example N	x_{N1}	x_{N1}	$\dots$	x_{Nd}	y_N

Recap: Notation

Note The input space can be (nearly) everything
Make things easier focus on d–dimensional vectors $\vec{x} \in \mathcal{X} \subseteq \mathbb{R}^n$

$\mathcal{D}$	Feature 1	Feature 2	$\dots$	Feature d	Label
Example 1	x_{11}	x_{12}	$\dots$	x_{1d}	y_1
Example 2	x_{21}	x_{22}	$\dots$	x_{2d}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
Example N	x_{N1}	x_{N1}	$\dots$	x_{Nd}	y_N

Matrix $X \in \mathbb{R}^{d \times N}$

Vector $\vec{y} \in \mathcal{Y}^N$

then in short $\mathcal{D} = (X, \vec{y})$

Recap: Problems (1)

Supervised Learning

- **Given** Set of labeled training examples / data $\mathcal{D} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N) \mid (\vec{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$
- **Find** A model $f : \mathcal{X} \to \mathcal{Y}$ so that $f(\vec{x}_i) \approx y_i$

Recap: Problems (1)

Supervised Learning

- **Given** Set of labeled training examples / data $\mathcal{D} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N) \mid (\vec{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$
- **Find** A model $f : \mathcal{X} \to \mathcal{Y}$ so that $f(\vec{x}_i) \approx y_i$

Note 1 If $|\mathcal{Y}| = 2$ its called binary classification
Note 2 If $\mathcal{Y} = \mathbb{R}$ its called regression
Example Classify songs into pop music vs. rap music

Recap: Problems (2)

Unsupervised Learning

- **Given** Set of un-labeled training data $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_N\}$
- **Find** A plausible model M that explains the data well

Example Put songs into similar groups to find new songs

Recap: Problems (2)

Unsupervised Learning

- **Given** Set of un-labeled training data $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_N\}$
- **Find** A plausible model M that explains the data well

Example Put songs into similar groups to find new songs

Semi-Supervised Learning

- **Given** Set of partially-labeled training data $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_m, (\vec{x}_{m+1}, y_{m+1}), \ldots, (\vec{x}_n, y_n)\}$
- **Find** A plausible model M that explains the data well with $M(\vec{x}_i) \approx y_i$ for known y_i

Example Put songs into similar groups knowing that some belong together / should never be in the same group

Recap: Problems (3)

Active Learning

- **Given** Set of partially-labeled training data $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_m, (\vec{x}_{m+1}, y_{m+1}), \ldots, (\vec{x}_n, y_n)\}$
- **Find** $\vec{x}_i \in \mathcal{D}$, so that we acquire the true y_i

Example For which songs should I hire an expert to classify them?

Recap: Problems (3)

Active Learning

- **Given** Set of partially-labeled training data $\mathcal{D} = \{\vec{x}_1, \ldots, \vec{x}_m, (\vec{x}_{m+1}, y_{m+1}), \ldots, (\vec{x}_n, y_n)\}$
- **Find** $\vec{x}_i \in \mathcal{D}$, so that we acquire the true y_i

Example For which songs should I hire an expert to classify them?

Reinforcement Learning

- **Given** A set of actions A and a reward function r
- **Goal** Find series of actions, so that reward is maximized

Example Maximize time the user listens to music by proposing new/similar songs

Recap: Principles of Machine Learning

Occam's Razor Favor simple models before complex ones

Recap: Principles of Machine Learning

Occam's Razor Favor simple models before complex ones

No free lunch All methods are equally good, but some favor certain data before other

Recap: Principles of Machine Learning

Occam's Razor Favor simple models before complex ones

No free lunch All methods are equally good, but some favor certain data before other

Bias-Variance trade-off For squared error

Recap: Principles of Machine Learning

Occam's Razor Favor simple models before complex ones

No free lunch All methods are equally good, but some favor certain data before other

Bias-Variance trade-off For squared error
Assume: $t = f(x) + \varepsilon$ with $\varepsilon \in \mathcal{N}(0, \sigma^2)$, then

$$\begin{aligned} \mathbb{E}_{x,t,\mathcal{D}}[(t - f_{\mathcal{D}}(x))^2] &= (f(x) - \mathbb{E}_{\mathcal{D}|x}[f_{\mathcal{D}}(x)])^2 + \mathbb{V}_{\mathcal{D}|x}[f_D(x)] + \mathbb{V}_{t|x}[\varepsilon] \\ &= (bias)^2 + variance + noise \end{aligned}$$

Extra: Measure Model quality

Question So, what is model quality?

Extra: Measure Model quality

Question So, what is model quality?

1. how well explains the model training data?
2. can we give any guarantees for new predictions?
3. how well generalizes the model to new and unseen data?

Extra: Measure Model quality (2)

Fact In binary classification we have two choices: predict 0 or 1
$\longrightarrow$ 2 possible wrong predictions and 2 possible correct predictions

Extra: Measure Model quality (2)

Fact In binary classification we have two choices: predict 0 or 1
→ 2 possible wrong predictions and 2 possible correct predictions

Visualization Confusion matrix

	Predicted value	
True value	True positive (TP)	False negative (FN)
	False positive (FP)	True negative (TN)

Extra: Measure Model quality (2)

Fact In binary classification we have two choices: predict 0 or 1
$\longrightarrow$ 2 possible wrong predictions and 2 possible correct predictions

Visualization Confusion matrix

	Predicted value	
True value	True positive (TP)	False negative (FN)
	False positive (FP)	True negative (TN)

Accuracy $Acc = \frac{TP+TN}{N}$

Big Remark The accuracy only tells us something about the data $\mathcal{D}$ we know! There are no guarantees for new data

Extra: Measure Model quality (3)

Obviously The best model has $Acc = 1$, the worst has $Acc = 0$
Observation If we store all the data for look-up, then $Acc = 1$

Extra: Measure Model quality (3)

Obviously The best model has $Acc = 1$, the worst has $Acc = 0$
Observation If we store all the data for look-up, then $Acc = 1$

Question Is that what we want?
Clear This is just memorizing the training data, no real learning!
Question How well deals our model with new, yet unseen data?

Extra: Measure Model quality (3)

Obviously The best model has $Acc = 1$, the worst has $Acc = 0$
Observation If we store all the data for look-up, then $Acc = 1$

Question Is that what we want?
Clear This is just memorizing the training data, no real learning!
Question How well deals our model with new, yet unseen data?

Idea Split data into training $\mathcal{D}_{Train}$ and test data $\mathcal{D}_{Test}$
Then $\mathcal{D}_{Test}$ is new to the model f
Question How to split $\mathcal{D}$?

Extra: Measure Model quality (4)

1) Test/Train Split Split $\mathcal{D}$ by size, e.g. 80% training and 20% test data
$\rightarrow$ Fast and easy to compute, but sensitive for “bad” splits.
$\rightarrow$ Model quality might be over- or under-estimated

Extra: Measure Model quality (4)

1) Test/Train Split Split $\mathcal{D}$ by size, e.g. 80% training and 20% test data
$\rightarrow$ Fast and easy to compute, but sensitive for “bad” splits.
$\rightarrow$ Model quality might be over- or under-estimated

2) Leave-One-Out Use every example once for testing and train model on the remaining data. Average results.
$\rightarrow$ N models are computed, but insensitive for “bad” splits.
$\rightarrow$ Usually impractical

Extra: Measure Model quality (4)

1) Test/Train Split Split $\mathcal{D}$ by size, e.g. 80% training and 20% test data
$\rightarrow$ Fast and easy to compute, but sensitive for "bad" splits.
$\rightarrow$ Model quality might be over- or under-estimated

2) Leave-One-Out Use every example once for testing and train model on the remaining data. Average results.
$\rightarrow$ N models are computed, but insensitive for "bad" splits.
$\rightarrow$ Usually impractical

3) K-fold cross validation Split data into k buckets. Use every bucket once for testing / train model on the rest. Average results.
$\rightarrow$ Insensitive for "bad" splits and practical. Usually $k \in \{5, 10\}$.

Extra: K nearest neighbor (K-NN) method

Goal Solve supervised problem

Thus We want a prediction method $\widehat{f}(\vec{x})$
Observation Examples $\vec{x}_i$ and $\vec{x}_j$ which are similar probably have the same label $y_i = y_j$

Extra: K nearest neighbor (K-NN) method

Goal Solve supervised problem

Thus We want a prediction method $\widehat{f}(\vec{x})$
Observation Examples $\vec{x}_i$ and $\vec{x}_j$ which are similar probably have the same label $y_i = y_j$

Idea Given new and unseen observation $\vec{x}$

- use distance function $dist \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- calculate $d(\vec{x}, \vec{x}_i)$ for all $i = 1, \ldots, N$
- find k nearest neighbors of $\vec{x}$ $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_k, y_k)\}$
- predict most common label in S

Extra: K nearest neighbor (K-NN) method

Goal Solve supervised problem

Thus We want a prediction method $\widehat{f}(\vec{x})$
Observation Examples $\vec{x}_i$ and $\vec{x}_j$ which are similar probably have the same label $y_i = y_j$

Idea Given new and unseen observation $\vec{x}$

- use distance function *dist*: $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- calculate $d(\vec{x}, \vec{x}_i)$ for all $i = 1, \ldots, N$
- find k nearest neighbors of $\vec{x}$ $S = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_k, y_k)\}$
- predict most common label in S

Note If S has equal number of positive and negative examples, take a random class

Extra: K-NN (Some Notes)

Note K-NN is a simple and large models. It simply stores $\mathcal{D}$.

Extra: K-NN (Some Notes)

Note K-NN is a simple and large models. It simply stores $\mathcal{D}$.

K-NN has two parameters

- *dist* Models the distance of neighbors. This must fit the data given! Usually euclidean norm is a good start:
$$dist(\vec{x}_i, \vec{x}_j) = \sqrt{(\vec{x}_i - \vec{x}_j)^T \cdot (\vec{x}_i - \vec{x}_j)}$$
- K Models the number of neighbors we want to look at.

Extra: K-NN (Some Notes)

Note K-NN is a simple and large models. It simply stores $\mathcal{D}$.

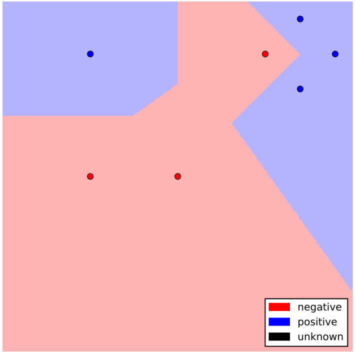
K-NN has two parameters

- *dist* Models the distance of neighbors. This must fit the data given! Usually euclidean norm is a good start:
$$dist(\vec{x}_i, \vec{x}_j) = \sqrt{(\vec{x}_i - \vec{x}_j)^T \cdot (\vec{x}_i - \vec{x}_j)}$$
- K Models the number of neighbors we want to look at.

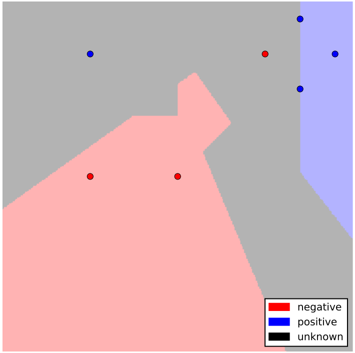
Note 2 K-NN can be used for regression as well. Just average the labels in S :

$$\widehat{f}(\vec{x}) = \frac{1}{k} \sum_{y \in S} y$$

Extra: K-NN Examples

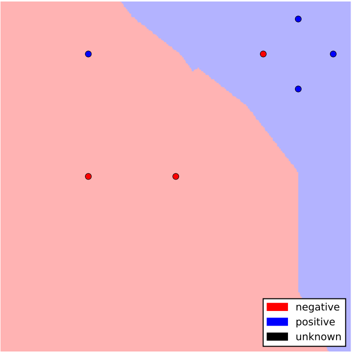


$k = 1$

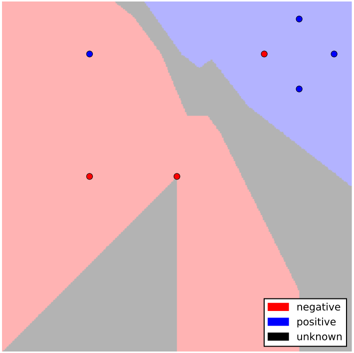


$k = 2$

Extra: K-NN More examples



$k = 3$



$k = 4$

Practical Data Science: Python

Data Scientist use `Python`

- Dynamically scripting language
- Usually interpreted
- Only syntax is checked before execution (fails late)
- Indentation is a part of syntax (e.g. tabs instead of '{')

Practical Data Science: Python

Data Scientist use `Python`

- Dynamically scripting language
- Usually interpreted
- Only syntax is checked before execution (fails late)
- Indentation is a part of syntax (e.g. tabs instead of '{')

Why Python?

- **Fast** Important algorithm have fast C/GPU backends
- **Flexible** Easy to test and try new things
- **Extensible** There are a lot of packages

Practical Data Science: Packages

Important Packages

- `numpy` Contains linear algebra functions
- `scipy` Contains common functions for scientific computing
- `matplotlib` Produces nice plots
- `pandas` Reads and handles files efficiently

Practical Data Science: Packages

Important Packages

- `numpy` Contains linear algebra functions
- `scipy` Contains common functions for scientific computing
- `matplotlib` Produces nice plots
- `pandas` Reads and handles files efficiently

Important tools

- `Python` I recommend version ≥ 3.5
- `pip/anaconda` Install/Manage new packages
- `IPython` (Enhanced) Interactive python shell
- `Jupyter` Browser notebook for 'story' driven programming
- `plotly` Interactive plots on the web

BUT Your editor and raw python is your best tool