Machine Learning

Zusammenfassung / Joel von Rotz / Quelldateien

Begriffe

Artificial Intelligence Software, die intelligentes Verhalten imitiert
$\hookrightarrow$ **Machine Learning** System, das von Erfahrung/Daten lernt
$\hookrightarrow$ **Deep Learning** basiert auf geschichteten neuralen Netzwerken

Supervised Learning Lernt anhand labeled Daten $\longrightarrow$ predicts unseen data, *Dimensionality Reduction*
Unsupervised Learning unlabeled Daten $\longrightarrow$ Daten-Struktur lernen und Kunden in Zielgruppen teilen (*Recommender Systems*)
Semi-Supervised Learning Mix Sup.+Unsup. $\longrightarrow$ wenig *Labeled* & viele *Unlabeled*
Reinforcement Learning Belohnungsfunktion $\longrightarrow$ sucht ideales Verhalten für maximale Belohnung

Regression Problem Label (vorherzusagendes Attribut) ist kontinuierlich
Classification Problem Label ist kategorial (binary -> Bsp. Ja, nein)
Hyperparameters vom Programmierer ausgewählte Parameter (Bsp. K bei K-NN)

Data Quality Assessment

● Identify data sources and trustworthiness ● Check data ranges (e.g. negative salary) ● Interpret statistical key figures ● Validate plausibility of variable correlations ● Measure data redundancy ● Check for anomalies in sytax and semantics ● Explore Null values and duplicate data records

Data Cleaning

Data quality can be improved (although document changes, inform,...):

● identify & remove duplicate data records ● replace null values (e.g. median or mean) or delete them (requires large data) ● investigate the origins of quality issues

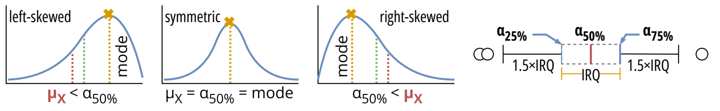
Feat. Eng. - Vector Space Model

Data set consists **only** of numerical data (e.g. color $\longrightarrow$ black, red, ...). Categorical data is transformed into numerical data

Statistics

mean μ easy calculate ; **median** robust against outliers ; Var$_{\text{smpl.}}$ $\frac{1}{n-1}$; Var$_{\text{pop.}}$ $\frac{1}{n}$

Var $\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_X)^2$; $\text{Cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_X)(y_i-\mu_Y)$



Pearson Correlation $\rho(X,Y) = \frac{\text{Cov}(X,Y)}{\sigma_X\sigma_Y}$ normalizes ranges: -1=anti-corr., +1=corr

Similariy Measures

Euclidean Distance $\sqrt{\sum_{i=1}^{n}(x_i-y_i)}$ (distance)

Cosine Distance 1 − CosineSimil. $= 1 - \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$ (distance + angle)

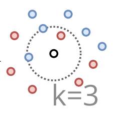
Manhattan Distance $\sum_i |x_i - y_i|$ (along a grid)

Levenshtein/Edit Distance for strings: +1 on delete, add ; +2 on change char

Jacard Similarity: $\frac{|X\cap^{and}Y|}{|X\cup^{or}Y|} \longrightarrow [0,1]$; 1=same

k-Nearest-Neighbor Classification

Data takes value of majority of k (hyperparameter) neighbors + (optional) weight based on $1/d$ (distance d).
− very slow with lots of data and most comput. is done in classification stage + least data hungry algo, good baseline for other classifier



Normalization

Min-Max: $\frac{x-\min x}{\max x - \min x} \longrightarrow [0,1]$ + interpretation in %, no negatives − no supervised

Z-Score: $\frac{x-\mu_X}{\sigma_X} \longrightarrow \begin{matrix}\mu_{X'}=0,\\ \sigma_{X'}=1\end{matrix}$ + for (un)supervised − no direct interpr., negatives

Regression Hypothesis & Evaluation Metrics

Goal is to fit a line or curve-function to a number of points. y_i ground truth, f_i prediction ; MAE (Mean Absolute Error), MAPE (Mean Absolute Percentage Error) human readable, MSE (Mean Squared Error) optimized for processing

$$\frac{1}{m}\left[\overset{\text{MAE}}{\sum_{i=1}^{m}|y_i-f_i|} \quad \overset{\text{MAPE}}{\sum_{i=1}^{m}\left|\frac{y_i-f_i}{y_i}\right|} \quad \overset{\text{MSE}}{\sum_{i=1}^{m}(y_i-f_i)^2}\right] \Bigg| \; \overset{R^2=\rho^2}{1-\frac{\sum_{i=1}^{m}(y_i-f_i)^2}{\sum_{i=1}^{m}(y_i-\mu_Y)^2}}$$

$R^2 \leq 1$; $R^2 < 0$ worse fit than horizontal hyperplane (mean) ; $R^2 > 1$ model performs worse than mean the fraction becomes

Model Evaluation Workflow

Workflow: (1) Mix data (2) Split into training & test data (3) train (4) evaluate! NEVER reuse test set!

Model Selection: (1) 60% Training, 20% Validation, 20% Test (2) Train different models (3) Check via Validation and select model (4) Use selection on test data!

K-Fold Cross Valid.: not enough data $\longrightarrow$ (1) Split 80% data into k parts (2) Round 1 uses first part as **validation** (3) average all accuracy to get final accuracy

Fitting Data

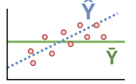
● Underfitting (too loose), Just Right (not perfect fit, but distribution is learnt), Overfitting (too perfect, no distribution)

Regression: X regressor, ε error $y_{real} - y_{lin}$: $Y = X\theta + \varepsilon$ ($y_j = \theta_0 + \theta_1 \cdot x_j + \varepsilon_j$)

Cost Function: used to minimize MSE ; $J(\theta_0,\theta_1) = \frac{1}{N}\sum_{j=N}^{N}\varepsilon_j^2$

Ordinary Least Squares (OLS): $\frac{\partial J(\theta)}{\partial\theta} = 0 \longrightarrow \theta_{opt} = (X^TX)^{-1}X^Ty$ (**Gradient Descent** iteratively: (1) random θ (2) small steps down cost surface ● direction = negative gradient ; $\theta_{new} = \theta_{old} - \alpha^{\partial J(\theta)/\partial\theta}$)

Regression Performance: $R^2 = 1 - \frac{\sum_j \varepsilon_j^2}{\sum_j \varepsilon_\delta^2} = 1 - \frac{\sum_j (y_j-\hat{Y})^2}{\sum_j (y_j-\bar{Y})^2}$
Plot data, as R^2 does not show the distribution (*Anscome's quartet*)!



To penalize large parameters and stop overfitting, **LASSO** and **Ridge** can be used.

$$J(\theta) = \underbrace{\frac{1}{N}\sum_{j=1}^{N}(y_j-\hat{y}_j)^2}_{\text{OLS}} + \underbrace{\lambda\sum_{k=1}^{M}\theta_k^2}_{\text{Ridge}} \Big/ \underbrace{\lambda\sum_{k=1}^{M}|\theta_k|}_{\text{LASSO}}$$

Polynomial Regression: $h(\theta,X) = \theta_0+\theta_1X+\theta_2X^2+\cdots \Rightarrow \theta_0+\theta_1X_1+\theta_2X_2+\cdots$

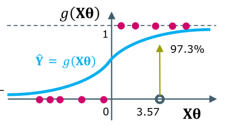
Classification

Classification predicts categories (*decision boundry*) *vs.* **Regression** predicts values (*data relationship*)

Logistic Regression

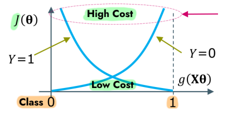
$$\hat{Y} = P(Y=1|X) = h_\theta(X) = g(X\theta) = \frac{1}{1+e^{-X\theta}}$$

Instead of either exactly 1 or 0, a percentage [0, 1] is given. No OLS-form possible, good parameters calculated via *Gradient Descent*.



Cross Entropy Cost Function

Classifying for one class **must** penalize the other!



One Versus The Rest

Train model based on a specific class, against others. It is either in that class (1) or not (0) $\longrightarrow$ **One vs. All** classification.

Performance Analysis

$$\text{Sensitivity / Recall} = \frac{TP}{TP+FN} \qquad \text{Specifity} = \frac{TN}{TN+FP}$$

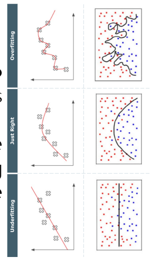
$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{F1 Score} = \frac{2\cdot\text{prec}\cdot\text{reca}}{\text{prec}+\text{reca}}$$

Conf. Matrix		Predict	
		No	Yes
Actual	No	TN	FP
	Yes	FN	TP

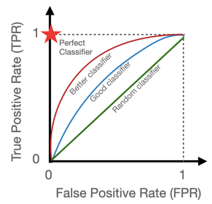
Accuracy $= {}^{(TP+TN)}/_{Total}$; lower F1 indicates some kind of skewed data (unbalance).

Dealing with Bias & Variance

Dealing with bias is easier (underfitting): ● larger set of feature ● different features ● feature engineering ● more complex algorithms (more data is not necessarily the answer)
Dealing with variance (overfitting): ● reduce features ● get more training data ● stop training early to avoid overfitting ● data cleaning
Bias means we are systematically off by a fixed amount. **Variance** means we are systematically off in a scattered way

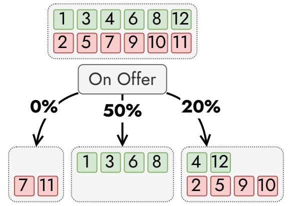


Receiver Operating Characterstic (ROC):
- to compare different models
- largest area under curve is best



Decision Trees

Gini Impurity measures the likelihood of an incorrect classification of a random data record. *Impure* features produce no significant information. **Cont. Var.** (1) Sort Table (2) Calculate avg. of adjacent values ($10 \longrightarrow 15 \longleftarrow 20$) (3) Averages are splitting criteria (histogramm binning reduces splits)



Label	p(No)	p(Yes)	$1\text{-}p(No)^2\text{-}p(Yes)^2$
0%	1	0	0
20%	2/3	1/3	4/9
50%	0	1	0

Gini Impurity of entire feature:
$2/12 \times 0 + 6/12 \times 4/9 + 4/12 \times 0 = 2/9$

Regression Trees ● Are grown like decision trees for classification but with different splitting criteria ● Instead of Gini impurity, **minimize variance of values in the same subset** ● Finally, predict average value of all instances in a leaf node

Tree Construction Rules

(1) **Only** positive or negative nodes left $\longrightarrow$ stop branch & assign corresponding decision as leaf node (2) **Some** positive and negative remain $\longrightarrow$ apply another feature (3) **No** instances left $\longrightarrow$ look at parent node and decide according the more frequent label (4) Instances left but **no** features $\longrightarrow$ training data is no good or contains hidden features ; decide on the more frequent label

Pros & Cons Decision & Regression Trees

+ readable, can both candle numeric & categorial, (almost) no data preparation, perform well on larger datasets − not as accurate, danger of over-complex trees, very sensitive to data quality

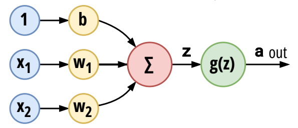
Random Forests (collection of decision trees): Utilize **random** (row)sampling with replacement and **random** (column) feature selection (without replacement) to be less sensitive & no overfitting.

Ensembles complex ML model built from simpler models. *Building Bricks* $\longrightarrow$ **Bragging** computes weighted average of votes from simple models, **Boosting** improves simple model by correcting its errors ($F_m(x) + h_m(x)$, e.g. Gradient Boosting).

Neural Networks

Input x_i from previous 'axons', contain some kind of information / Weights b, w_i define the inputs' influence / Sum $\sum$ sum of the weighted inputs / Activation $g(z)$ can be **hard threshold** ($a = g(z) = 1$ if $\sum \geq 0$ else 0), **logistic regression** ($g(z) = \frac{1}{1+\exp(-z)}$)

Perceptron ($z = b + \sum_i x_i \cdot w_i$)



Feed Forward data moves one way (left $\longrightarrow$ right)

Hidden Layers: allows for new features a to be added

Layer Count $c_L = c_{hidden} + c_{output}$ (input is not counted)

Feed Forward →

- A neural net is the connection of many artificial neurons in parallel and series
- Each layer of a feed-forward ANN is defined by its weights **W** and biases **b**
- Given data with labels **y** and features **x**, we feed **x** forward through the ANN

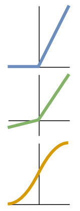
Back Propagation ←

- At the output of the ANN we get the estimate $\hat{\mathbf{y}}$ of **y**
- Because we start with random **W** and **b**, the estimate $\hat{\mathbf{y}}$ is poor
- Define the error J as a measure of distance between $\hat{\mathbf{y}}$ and **y**
- We seek the best **W** & **b** to minimize J. This occurs when the gradient of J is small or zero
- So we need the gradient of J
- Compute the gradient iteratively at each layer: more efficient!
- We introduce deltas δ at each layer to simplify computation

Gradient Descent ↓

- Back-propagation gives the gradient of J with respect to weights **W** and biases **b**
- The gradient is a vector which points steeply up the error surface
- So take a small step α in the opposite direction, reducing J
- "Take a step" means making small changes to **W** & **b**
- Changes are defined exactly by $-\alpha$ times the gradient
- This is gradient descent by feed forward & back-prop using labelled data

Activation Function



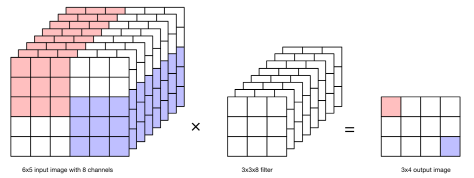
ReLU (rectified linear unit ;good for hidden) $\max(0, x)$ brings better perfomance and alleviates the vanishing gradients problem (flat line activation) $\longrightarrow$ introduces **dying units** (bad for gradient training)

Leaky ReLU $\max(0.1x, x)$ introduces non-zero for negative values and fixes vanishing gradients + dying unit $\longrightarrow$ results aren't always consistent

Soft-Max (good for output) $\text{softmax}_j = \frac{\exp z_j}{\sum_{k=1}^{K} \exp(z_k)}$ is a generalization of the logistic function to **multiple dimensions** and can be interpreted as probabilites (normalization).

Convolution Neural Network

Features are extracted from images using **convolution** (Faltung). **Weights** are used as Masks/Filters on these images to extract certain features such as edges or colour. The final layer mostly uses **softmax**!



Pooling applies operations such as max, min or avg to a pixel neighbour and the neighbourhood size is determined by the pool size parameter.

Parameter Counting Example on 20×20–**RGB**–Image (L1) *$\underline{2}$ Filters of 3×3* ($(3 \cdot 3 \cdot 3 + 1) \cdot \underline{2} = 56$ results $18 \times 18 \times 2$ Layers) (L2) *$\underline{3}$ Filters of 3×3 with 2 channels each* ($(3 \cdot 3 \cdot 2 + 1) \cdot \underline{3} = 57$ results $16 \times 16 \times 3$ Layers) (L3) *Flatten result to a <u>single</u> column of neurons* ($\underline{1} \cdot 16 \cdot 16 \cdot 3 = 1 \times 768$) (L4) *Fully connected (aka. dense) layer and 2 output channels* ($\#\text{param} = 768 \cdot 2 + 2 = 1538$ (2 biases! & 2 neurons/output shape))

Natural Language Processing

Syntax compare spelling or words / **Similarity** Levenshtein Distance

Word Relatedness Words are *related*… • when they appear in the **same document** (synonyms not related!!) • when they relate to the **same topic** (requires a lot of space) • when they can be replaced by each other

Word Similarity Words are *similar* when they appear in the **same context**

Term Frequency (Inverse Doucment Frequency) $\text{TF-IDF}(x, y) = \text{TF}(x, y) \cdot \log\left(\frac{N}{\text{DF}(x)}\right)$ / N number of documents in the corpus, **TF** counts the number of occurences of x in y, **DF** counts the number of documents that contain x. *QUETZALCOATLUS* has high score, due to being used a lot in a **single** document, compared to the low scoring word *AND* (used everywhere).

Recurrent Neural Nets (RNN)

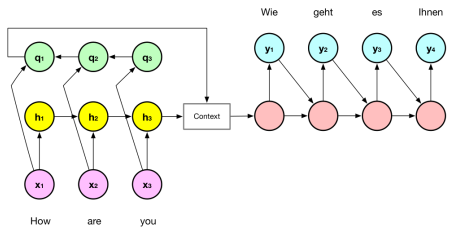
$$h_t = (W_{xh} \cdot x_t + W_{hh} \cdot h_{t-1} + b_h) \qquad y_t = \varphi(W_{hy} \cdot h_t + b_y)$$

RNN are made for the supervised analysis of arbitrary time-series data (instead of fixed length, various sized strings can be analyzed) and **do not parallelize**! LSTM & GRU help with alleviating vanishing gradients to remember more of the context (Training an RNN by multiplying gradients backwards in time).

GRU ($\cancel{h_t}$) use hidden states as memory (faster) & **Long Short-Term Memory Model** have an additional memory state (better performance)

Bidirectional RNN reads left and right $\longrightarrow$ produces together output

Encoder-Decoder Architecture: usually RNNs have one input and output, which makes it unusuable for i.e. translators. To allow in- and outputs of various lengths, en- & decoder architecture is used.



Attention Mechanism

Mimics the behaviour of knowing which page the content can be found in. Words are assigned an *importance* value which is used by the next generation for the output words.

In ⇨ Encoder ▷ Attention ▷ Decoder ⇨ **Out**

k-Means Clustering

(1) Input k Clusters and data points x_n (2) Randomly choose k clusters centers μ_k (3) Repeat until convergence (stability): (a) assign data points to nearest cluster center (b) update cluster center with mean of assigned datapoints

Two data points can meet in a cluster and separate again in the next step! $\longrightarrow$ pattern always changes

Clustering Distortion

Total distortion sum of squared distances of points and cluster center • **Average Distortion** average of total distortion (additional $^1/_n$) and allows for comparison with other data sets

k-Means only approximates optimal solutions and converges either to global **OR** local minimum

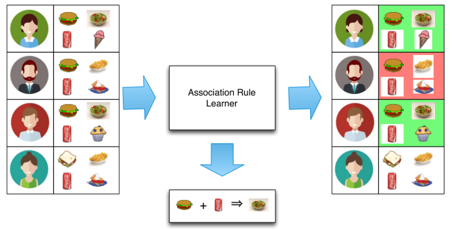
Elbow Methods increasing the amount of clusters leads to an 'elbow' in the distance-nCluster diagram $\longrightarrow$ use cluster count at the 'elbow'

Agglomerative Clustering

Outputs an hierarchy (*dendrogram*) that are informative and human interpretable. It does not scale with larger data, but <u>cluster stays together forever!</u> (1) Initially each datapoint is a separate cluster (2) While stop condit. (on no change) is not met: (a) calculate distance between pairs of cluster (b) merge pairs with shortest distance

Association Rules

Implication ($X \longrightarrow Y$): the higher confidence, the more likely Y is also present in transaction with X



Support (Interestingness): how frequently an item set occurs in the data $\text{support}(\{i_1, \ldots, i_n\}) = \frac{\#\text{purchases of } \{i_1, \ldots, i_n\}}{\#\text{transactions}}$

Support of an Association $\text{support}(X \longrightarrow Y) = \text{support}(Y \longrightarrow X) = \text{support}(X \cup Y)$

Confidence (Trustworthiness): how frequently items in Y appear in transactions that contain X $\text{confidence}(X \longrightarrow Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)}$

BOTH Support and **Confidence** need to be considered. − Only Antecedent X and co-occurence $X \cup Y$ is taken into account, not Y ; sup. & conf. cannot filter coincidental occurrences out.

Lift (Association Strength): how many times more often X and Y show up together than expected $\text{lift}(X \longrightarrow Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X) \cdot \text{support}(Y)}$ ($= 1$ independent; < 1 X & Y appear less often together ; > 1 X & Y appear often together)

Apriori Algorithm

(1) Generate frequent item sets $\longrightarrow$ (1) frequent items leads subsets being frequent too (2) infrequent leads supersets infrequent too. (shape **lattice of subsets**: first layer single node, second double connection,…)

(2) Extract rules from frequent item satisfying the confidence threshold $\longrightarrow$ (1) when an item violates the confidence bound all subsets can be eliminated. (shape **lattice of rules**)

Recommender System - Non-Personalized

+ simple, fast & cheap ; Associations are context aware ; Associations can be tailored to specific businesses (who viewed, who bought…) ; No new user cold start problem − non-personalized

Recommender System - Personalized

$$P_{uj} = \frac{\sum_{i \in N_j} \text{similar}(j, i) \cdot r_{ui}}{\sum_{i \in N_j} \text{similar}(j, i)} \qquad P_{uj} = \bar{r}_u + \frac{\sum_{i \in N_u} \text{similar}(u, i) \cdot (r_{ij} - \bar{r}_i)}{\sum_{i \in N_u} \text{similar}(u, i)}$$

N rated items (hyperparameter) ; P_{uj} how much does user u like item j ; N_j Neighbours of item j ; r_{ui} rating that user u gave item i ; N_u Neighbors of user u ; r_{ij} from neighbor i to item j

Personalized Content-Based Recommendations (left equ.): + user independance, no new item cold start problem − limited content analysis, overspecialization (bubble esque), severe new user cold start problem

User-to-User Collaborative Filtering (right equ.): + cross-category recommendations − many participants required, system must first learn user + item rating (cold start problem), gray sheep problem (unusual taste results in low accuracy)

Hybridization

Combination of different recommender systems! • **Weighted Approach** item score is weighted sum score from different recommenders • **Mixed Approach** results of different recommender are presented • **Cascade Approach** one recommender refines the result of other method • **Switching Approach** use different methods ta different places (online shops)