Business Analytics
Association Rules and Recommenders

Prof. Bichler

Decision Sciences & Systems

Department of Informatics

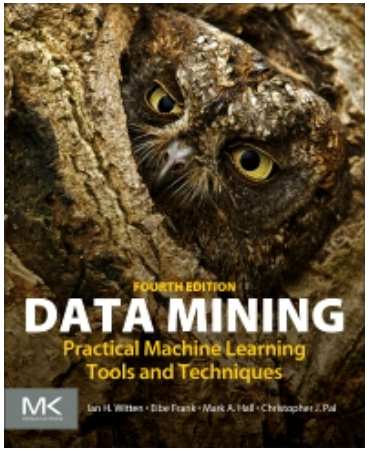
Technische Universität München

Course Content

- Introduction
- Regression Analysis
- Regression Diagnostics
- Logistic and Poisson Regression
- Naive Bayes and Bayesian Networks
- Decision Tree Classifiers
- Data Preparation and Causal Inference
- Model Selection and Learning Theory
- Ensemble Methods and Clustering
- High-Dimensional Problems
- **Association Rules and Recommenders**
- **Neural Networks**

Recommended Literature

- **Data Mining: Practical Machine Learning Tools and Techniques**
  - Ian H. Witten, Eibe Frank, Mark Hall, Christopher Pal
  - http://www.cs.waikato.ac.nz/ml/weka/book.html
  - Section: 4.5, 6.3



- https://en.wikipedia.org/wiki/Association_rule_learning

- Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009).

Unsupervised Learning

- Clustering
  - Unsupervised classification, that is, without the class attribute
  - Want to discover the classes

- Association Rule Discovery
  - Discover correlation among different attributes
  - Widely used (e.g., market basket analysis), later used for cross- and up-selling

Association Rule Discovery

- Aims to discover interesting correlations or other relationships in large databases

- Finds a rule of the form

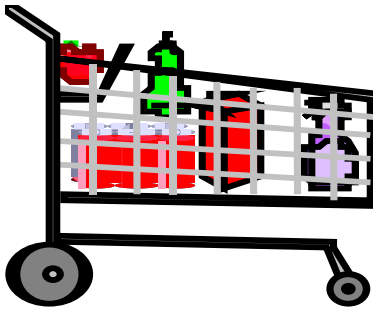
```
if A and B then C and D
```

- Which attributes will be included in the relation is unknown

Market Basket Analysis

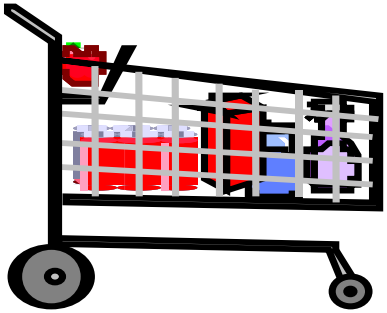
- Analyze customer buying habits by finding associations and correlations between the different items that customers place in their “shopping basket”

Milk, eggs, sugar, bread



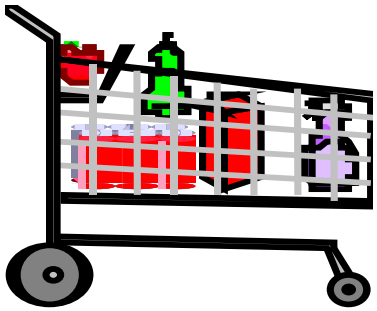
Customer1

Milk, eggs, cereal, bread



Customer2

Eggs, sugar



Customer3

Goal of MBA

- MBA applicable whenever a customer purchases multiple things in proximity
  - credit cards
  - services of telecommunication companies
  - banking services
  - medical treatments
  - Web usage mining

- Derive actionable information: can suggest
  - How do you boost the sales of a given product?
  - What other products do discontinuing a product impact?
  - Which products should be shelved together (store layout) / recommended (cross-selling)?

Transaction Database: Example

ITEMS:

A = milk
B= bread
C= cereal
D= sugar
E= eggs

TID	Products
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

Attributes converted to binary flags

TID	A	B	C	D	E
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

Instances = Transactions

Support and Frequent Itemsets

Support of an itemset $I = \{i_1, i_2, \ldots, i_n\}$

- $\text{supp}(I)$ = proportion of transactions t that support (i.e. contain) I

In example database:

$\text{supp}(\{A, B, E\}) = \frac{2}{9}$, $\text{supp}(\{B, C\}) = \frac{4}{9}$

- Frequent itemset I is one with at least the minimum support
- $\text{supp}(I) \geq minsupp$

TID	A	B	C	D	E
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

Terminology

Transaction:

Relational format	**Compact format**
<Tid, item>	**<Tid, itemset>**
<1, item1>	<1, {item1,item2}>
<1, item2>	<2, {item3}>
<2, item3>	

Item: single element**,** **Itemset**: set of items

Support of an itemset I: proportion of transactions containing I

Minimum Support $minsupp$: threshold for support

Frequent Itemset : with $supp \geq minsupp$

Frequent Itemsets represent sets of items which are positively correlated

Frequent Itemsets

Transaction ID	Items Bought
1	dairy,fruit
2	dairy,fruit, vegetable
3	dairy
4	fruit, cereals

$$\text{supp}(\{dairy\}) = \frac{3}{4}\ (75\%)$$

$$\text{supp}(\{fruit\}) = \frac{3}{4}\ (75\%)$$

$$\text{supp}(\{dairy, fruit\}) = \frac{2}{4}\ (50\%)$$

If $minsupp = 60\%$, then {dairy} and {fruit} are frequent while {dairy, fruit} is not.

From Frequent Itemsets to Association Rules

- *Q: Given frequent set {A,B,E}, what are possible association rules?*
  - $A \Rightarrow B, E$
  - $A, B \Rightarrow E$
  - $A, E \Rightarrow B$
  - $B \Rightarrow A, E$
  - $B, E \Rightarrow A$
  - $E \Rightarrow A, B$
  - $\_ \Rightarrow A, B, E\ (empty\ rule), or\ true \Rightarrow A, B, E$

Rule Support and Confidence

- *Suppose* $R: \ I \Rightarrow J$ *is an association rule*
  - $\text{supp}(R) = \text{supp}(I \cup J)$ is the support of rule R
    - support of itemset $I \cup J$
  - $\text{conf}(R) = \text{supp}(R) \ / \ \text{supp}(I)$ is the confidence of R
    - fraction of transactions with I that have $I \cup J$

- *Association rules with minimum support and confidence are sometimes called "***strong***" rules*

Association Rules Example

Q: Given frequent set {A,B,E}
($minsupp = 2/9$), what association rules
have $minconf = 50\%$ *?*

$$A,B \Rightarrow E : \text{conf} = 2/4 = 50\%$$
$$A,E \Rightarrow B : \text{conf} = 2/2 = 100\%$$
$$B,E \Rightarrow A : \text{conf} = 2/2 = 100\%$$
$$E \Rightarrow A,B : \text{conf} = 2/2 = 100\%$$

Don't qualify

$$A \Rightarrow B,E : \text{conf} = 2/6 = 33\% < 50\%$$
$$B \Rightarrow A,E : \text{conf} = 2/7 = 28\% < 50\%$$
$$\_ \Rightarrow A,B,E : \text{conf: } 2/9 = 22\% < 50\%$$

TID	List of items
1	**A, B, E**
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	**A, B, C, E**
9	A, B, C

Summary: Association Rule Mining

- STEP 1: Find all (frequent) item sets that meet minimum support
- STEP 2: Find all rules that meet minimum confidence
- STEP 3: Prune

Efficient Ways for Generating Item Sets

- How do we generate minimum support item sets in a scalable manner?
  - Total number of item sets is huge
  - Grows exponentially in the number of attributes

- Need an efficient algorithm:
  - Start by generating minimum support 1-item sets
  - Use those to generate 2-item sets, etc.

- Why does this work?

Justification

Item Set 1: {Humidity = high}

$$\text{supp}(1) = \textit{Proportion of records where humidity is high}$$

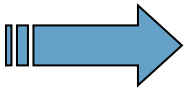
Item Set 2: {Windy = false}

$$\text{supp}(2) = \textit{Proportion of records where windy is false}$$

Item Set 3: {Humidity = high, Windy = false}

$$\text{supp}(3) = \textit{Proportion of records where humidity is high and windy is false}$$

$\text{supp}(3) \leq \text{supp}(1)$
$\text{supp}(3) \leq \text{supp}(2)$



If Item Set 1 and 2 do not both meet min. support Item Set 3 cannot either

Subset Property: Every subset of a frequent set is frequent!

Generating Item Sets

Start with all
3-item sets
that meet min.
support, where
A: {outlook=sunny}

{A, B, C}
{A, B, D}
{A, C, D}
{A, C, E}
{B, C, D}

Merge to
generate
4-item sets

{A, B, C, D}
{A, C, D, E}

Candidate 4-item sets with minimum support (must be checked)

There are only two 4-item sets that could possibly work

(Consider only
sets that start
with the same
two attributes, e.g. AB)

Apriori Algorithm: Basic Idea

- Apriori algorithm (Agrawal & Srikant): use one-item sets to generate two-item sets, two-item sets to generate three-item sets, …
  - If {A, B} is a frequent item set, then {A} and {B} have to be frequent item sets as well!
  - In general: if X is frequent k-item set, then all $(k-1)$-item subsets of X are also frequent

  $\Rightarrow$Compute k-item set by merging $(k-1)$-item sets

Algorithm for Generating Item Sets

- 1) Generating Item Sets
  - Build up from 1-item sets so that we only consider item sets that are found by merging two minimum support sets
  - Only consider sets that have all but one item in common

- 2) Generating rules from item sets
  - Can be computationally expensive for large attribute sets
  - Speed up by considering that rules with multiple consequences can only hold if simpler ones are true as well

Generating Rules

Meets min. support and conf.

If windy = false and play = no then outlook = sunny and humidity = high

⇓ only holds if

Meets min. support and conf.

If windy = false and play = no
then outlook = sunny

If windy = false and play = no
then humidity = high

Apriori Algorithm

- This is a simplified description of the Apriori algorithm

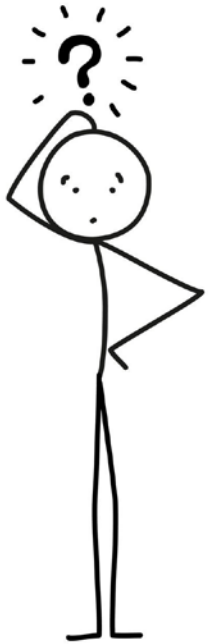
- Developed in early 90s by R. Agrawal (IBM Research, Almaden) and is the most commonly used approach
  - Alternatives are Eclat, FP-growth, and OPUS search

- New developments focus on
  - Generating item sets more efficiently
  - Generating rules from item sets more efficiently

Applications

- Market Basket Analysis
  - Store layout, client offers
  - Not always useful information
    - Wal-Mart knows that customers who buy Barbie dolls have a 60% likelihood of buying one of three types of candy bars
- Finding unusual events
  - What is Strange About Recent Events
- Applications to recommender systems

Explain why Apriori scales in spite of the exponential explosion of subsets?



Recommender Systems

- Systems for recommending items (e.g. books, movies, web pages, newsgroup messages) to users based on examples of their preferences.

- Many on-line stores provide recommendations
  – Amazon, eBay, Netflix, Last.fm, etc.

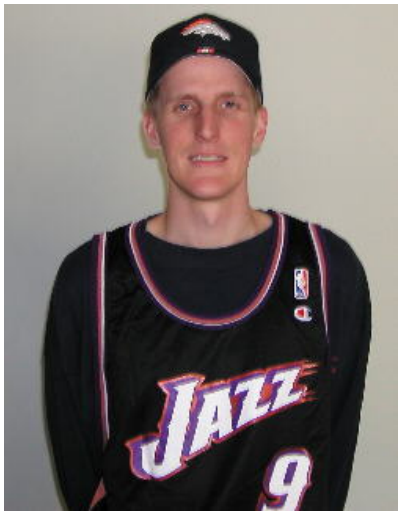
- Recommenders have been shown to substantially increase sales at on-line stores.
  – A large part of the rentals at NetFlix originate from recommendations.

- “Collaborative filtering” as wide-spread approach
  – based on similarities among users tastes

Recommendations

Customer A

- Buys Miles Davis CD
- Buys Dizzy Gillespie CD



Customer B

- searches for Miles Davis
- Sales staff presents Gillespie (knowing other customers preferences)

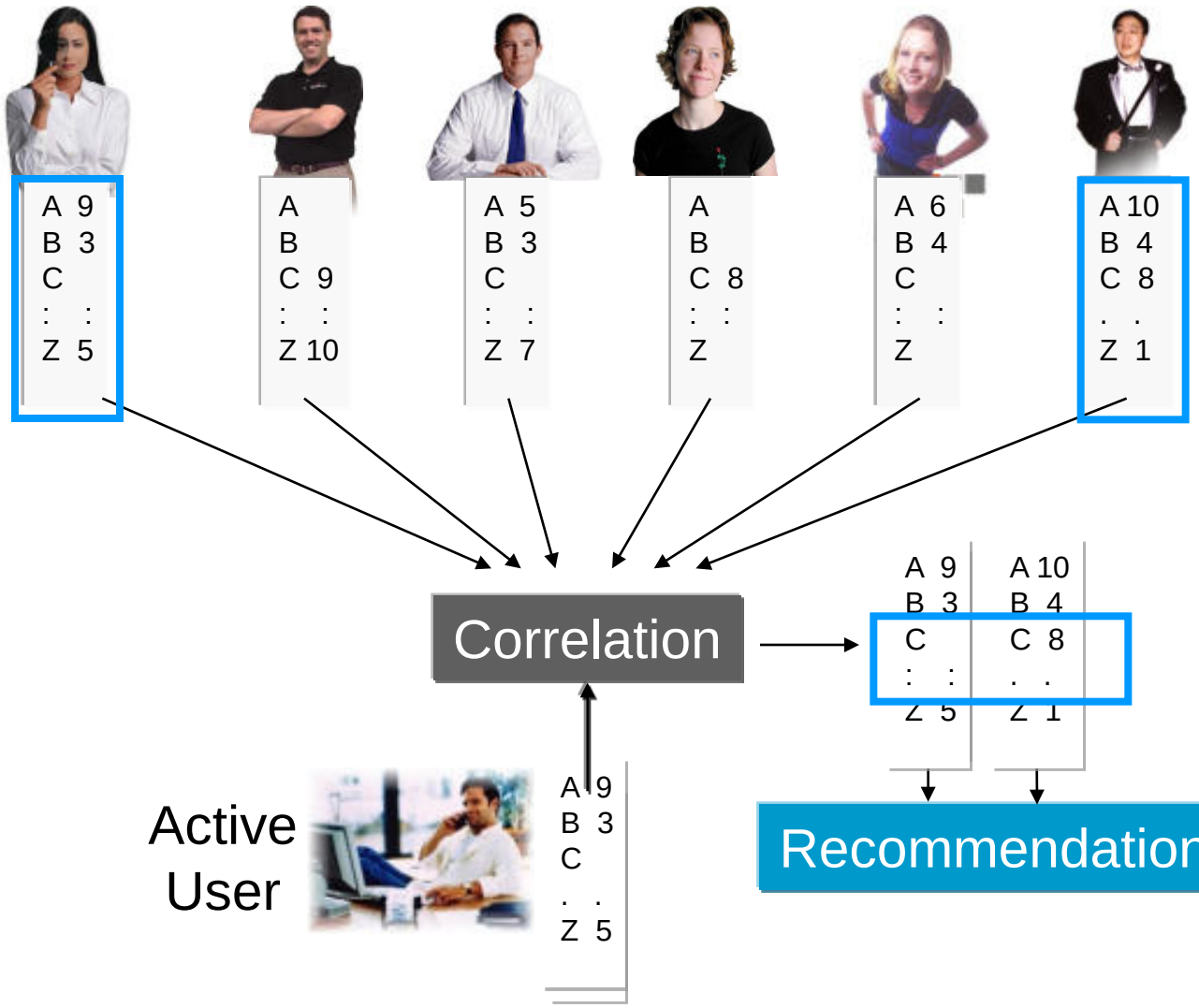
Association Rules

- Product associations
  - 90% of users who like product A and product B also like product C
  - A and B $\Rightarrow$ C (90%)
- User associations
  - 90% of products liked by user A and user B are also liked by user C
- Use combination of product and user associations

Collaborative Filtering

- Maintain a database of many users' ratings of a variety of items
- For a given user, find other similar users whose ratings strongly correlate with the current user
- Recommend items rated highly by these similar users, but not rated by the current user

Collaborative Filtering



Similarity Weighting

- Typically use Pearson correlation coefficient between ratings for active user, a, and another user, u

$$c_{a,u} = \frac{\mathrm{cov}(r_a, r_u)}{\sigma_{r_a} \sigma_{r_u}}$$

- r_a and r_u are the ratings vectors for the n items rated by **both** a and u

Significance Weighting

- Important not to trust correlations based on very few co-rated items
- Include significance weights, $s_{a,u}$, based on number of co-rated items, l

$$w_{a,u} = s_{a,u} c_{a,u}$$

$$s_{a,u} = \begin{cases} 1 \; if \; l > 50 \\ \frac{l}{50} if \; l \leq 50 \end{cases}$$

Rating Prediction

- Predict a rating, $p_{a,i}$, for each item i, for active user, a, by using the k selected neighbor users, $u \in \{1,2,\dots,k\}$

- To account for users different ratings levels, base predictions on differences from a user's average rating

- Weight users' ratings contribution by their similarity to the active user

$$p_{a,i} = \bar{r}_a + \sum_{u=1}^{k} \frac{w_{a,u}\left(r_{u,i} - \bar{r}_u\right)}{\sum_{u=1}^{k} \left|w_{a,u}\right|}$$

- Sparsity is an issue: often there are very few ratings per user.

SVD for Dining Recommendations

Let's consider 5 venues in a city and 8 people.

The people-location matrix ***A*** captures the number of times a person has visited the corresponding locations

$$A = \begin{array}{c} \begin{array}{ccccccc} \text{Starbucks} & \text{Costa cafe} & \text{Greek} & \text{Italian} & \text{Chinese} & \text{Japanese} & \text{McDonalds} \end{array} \\ \begin{pmatrix} 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 & 0 \end{pmatrix} \end{array}$$

SVD for Dining Recommendations

A

$$\begin{pmatrix} 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 & 0 \end{pmatrix} =$$

U (maps users to concepts)

$$\begin{pmatrix} -0.6155 & 0 & 0 & 0 \\ -0.4924 & 0 & 0 & 0 \\ -0.6155 & 0 & 0 & 0 \\ 0 & -0.6217 & 0 & -0.5035 \\ 0 & -0.4663 & 0 & -0.3776 \\ 0 & -0.6293 & 0 & 0.7771 \\ 0 & 0 & -0.7809 & 0 \\ 0 & 0 & -0.6247 & 0 \end{pmatrix}$$

S (strengths of concepts)

$$\cdot \begin{pmatrix} 11.4891 & 0 & 0 & 0 \\ 0 & 9.0771 & 0 & 0 \\ 0 & 0 & 9.0554 & 0 \\ 0 & 0 & 0 & 0.7790 \end{pmatrix}$$

Coffee lovers

Mediterranean food

Asian food

Fast food

$$\begin{pmatrix} -0.7071 & -0.7071 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.7054 & -0.7054 & 0 & 0 & -0.0693 \\ 0 & 0 & 0 & 0 & -0.7071 & -0.7071 & 0 \\ 0 & 0 & -0.0490 & -0.0490 & 0 & 0 & 0.9967 \end{pmatrix}$$

V^T (maps venues to concepts)

Low-rank Approximation

We can ignore the fourth concept since the singular value is small

$$\begin{pmatrix} 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 & 0 \end{pmatrix} \approx \begin{matrix} \text{cafe} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \text{mediter.} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \text{asian} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \text{misc} \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \mathbf{U}$$

$$\begin{pmatrix} -0.6155 & 0 & 0 & 0 \\ -0.4924 & 0 & 0 & 0 \\ -0.6155 & 0 & 0 & 0 \\ 0 & -0.6217 & 0 & 0 \\ 0 & -0.4663 & 0 & 0 \\ 0 & -0.6293 & 0 & 0 \\ 0 & 0 & -0.7809 & 0 \\ 0 & 0 & -0.6247 & 0 \end{pmatrix} \cdot \mathbf{S} \begin{pmatrix} 11.4891 & 0 & 0 & 0 \\ 0 & 9.0771 & 0 & 0 \\ 0 & 0 & 9.0554 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot$$

$$\mathbf{V^T} \begin{pmatrix} -0.7071 & -0.7071 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.7054 & -0.7054 & 0 & 0 & -0.0693 \\ 0 & 0 & 0 & 0 & -0.7071 & -0.7071 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Prediction

$$\begin{pmatrix} 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 5 & 5 & 0 \\ 0 & 0 & 0 & 0 & 4 & 4 & 0 \end{pmatrix} \approx \begin{pmatrix} -0.6155 & 0 & 0 & 0 \\ -0.4924 & 0 & 0 & 0 \\ -0.6155 & 0 & 0 & 0 \\ 0 & -0.6217 & 0 & 0 \\ 0 & -0.4663 & 0 & 0 \\ 0 & -0.6293 & 0 & 0 \\ 0 & 0 & -0.7809 & 0 \\ 0 & 0 & -0.6247 & 0 \end{pmatrix} \cdot \begin{pmatrix} 11.4891 & 0 & 0 & 0 \\ 0 & 9.0771 & 0 & 0 \\ 0 & 0 & 9.0554 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot$$

cafe mediter. asian misc — **U**

S

Naive SVD

$r_{ui} = U(User1) * S * V^t(Costa)$

$r_{1Costa} = 5$

$r_{1Chinese} = 0$

$$\begin{pmatrix} -0.7071 & -0.7071 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.7054 & -0.7054 & 0 & 0 & -0.0693 \\ 0 & 0 & 0 & 0 & -0.7071 & -0.7071 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Starb. Costa Greek Italian Chinese Japanese misc.

V^T

SVD-Based Recommendations

1. Start with a (sparse) user(u)-product(p) ratings matrix R.
2. Most values are missing in many applications! Naive approach: fill user-product ratings matrix using the average ratings for a product for example.
3. Normalize the matrix and subtract user average $\overline{r_u}$ from each rating resulting in matrix R‘.
4. Factor the matrix R‘ using SVD into U, S, and V and reduce S to a low dimension k (this parameter needs fine-tuning).
5. Use the matrices to compute the recommendation score for any user and product via the dot product of the u‘th row of U_k and the p‘th column of V_k with S_k and add the user average back.
   - Earlier example: $r_{ui} = \overline{r_u} + U(User1) * S * V^t(Costa)$

The matrix is sparse, but we need values for all entries! Rather than step 2 recommenders use convex optimization to find values for U and V^t such that the known values in R are best approximated.

Problems with Collaborative Filtering

- Cold Start: There needs to be enough other users already in the system to find a match

- Sparsity: If there are many items to be recommended, even if there are many users, the user/ratings matrix is sparse, and it is hard to find users that have rated the same items

- First Rater: Cannot recommend an item that has not been previously rated
  - New items
  - Esoteric items

- Popularity Bias: Cannot recommend items to someone with unique tastes.
  - Tends to recommend popular items

Content-Based Filtering

- Recommendations are based on information on the content of items rather than on other users' opinions
  – e.g. content-based recommender for books using information about titles

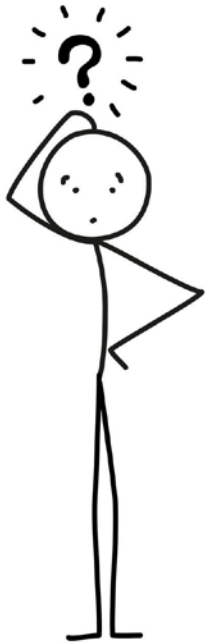
- Uses a machine learning algorithm (classification) to induce a profile of the users preferences from examples based on a featural description of content

- Able to recommend to users with unique tastes.

- Able to recommend new and unpopular items.

- Often combined with collaborative filtering.

What are the advantages of SVD in applications to recommender systems?



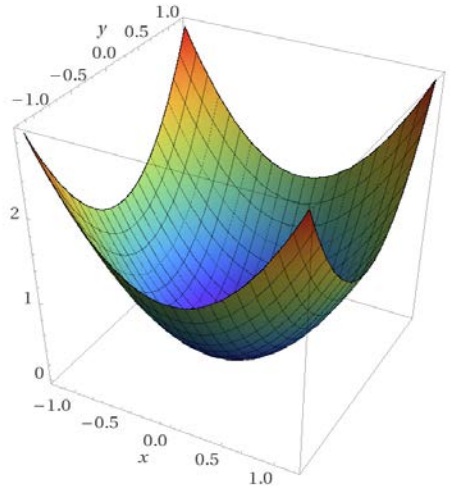
Course Content

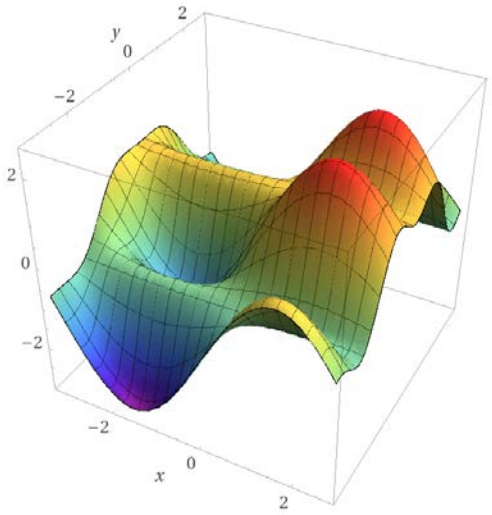
- Introduction
- Regression Analysis
- Regression Diagnostics
- Logistic and Poisson Regression
- Naive Bayes and Bayesian Networks
- Decision Tree Classifiers
- Data Preparation and Causal Inference
- Model Selection and Learning Theory
- Ensemble Methods and Clustering
- High-Dimensional Problems
- Association Rules and Recommenders
- **Neural Networks (Intro)**

Linear Models vs. Non-Linear Neural Networks

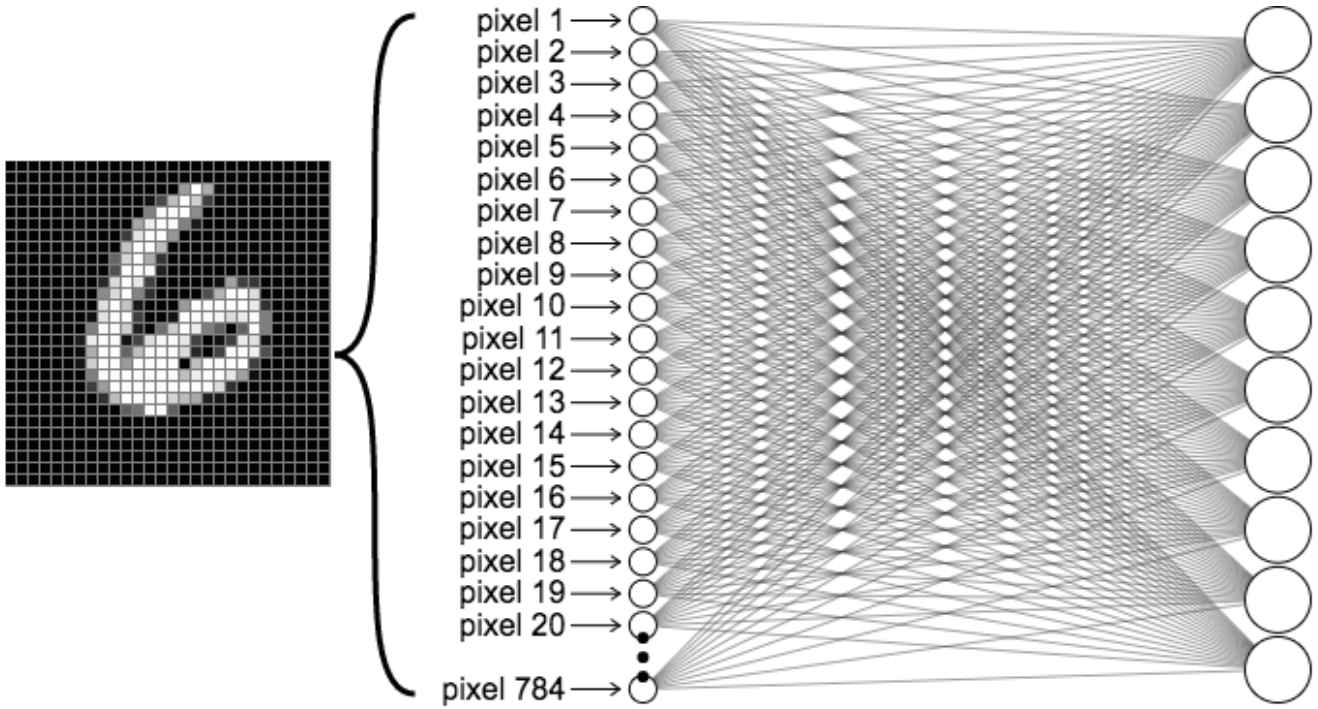
- Convex optimization
- Good for CPUs
- Reliable convergence
- Few hyperparameters
- Simple to interpret
- Requires feature engineering

- Non-convex optimization of the loss fctn.
- Often difficult to train
- Many hyperparameters
- Hard to interpret
- Automatically explores feature-engineering for the user

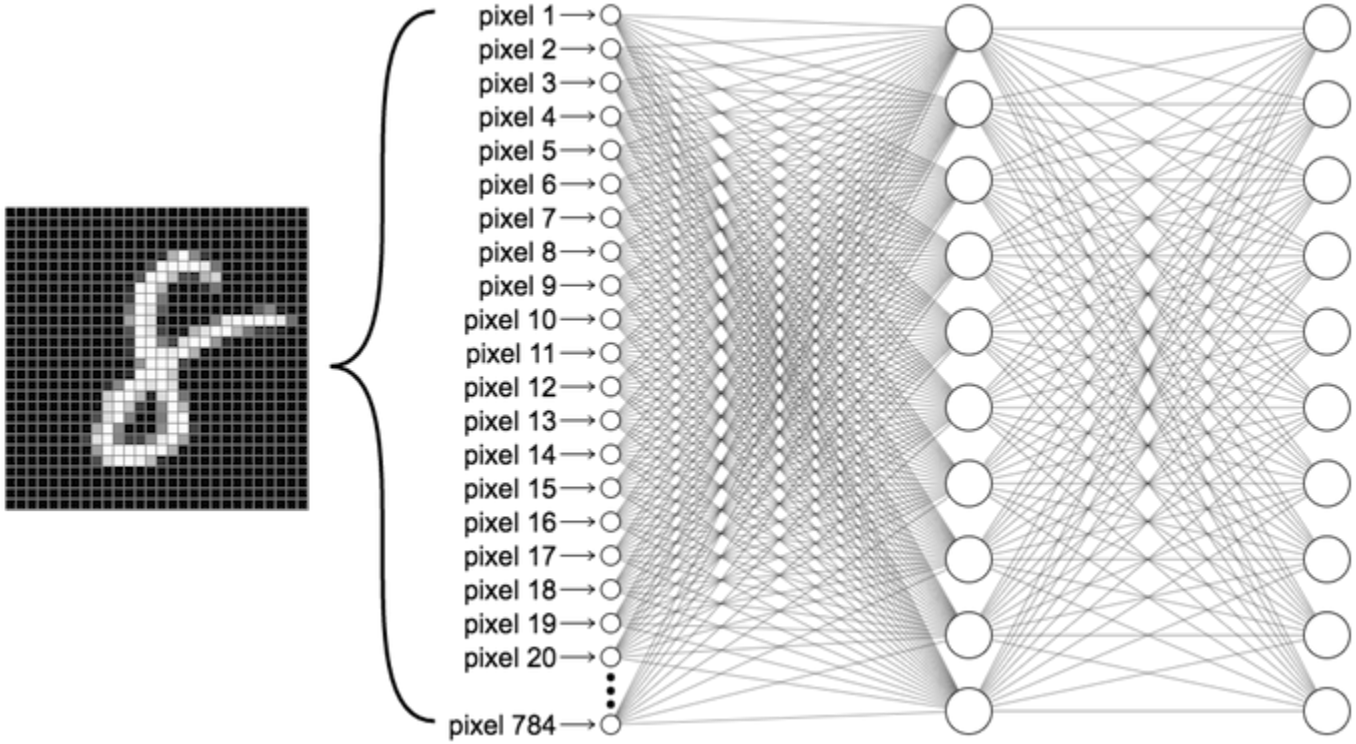




Classification via Neural Networks



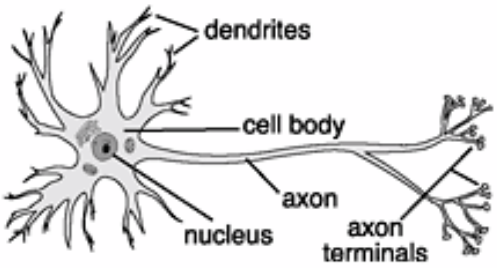
Classification via Neural Networks

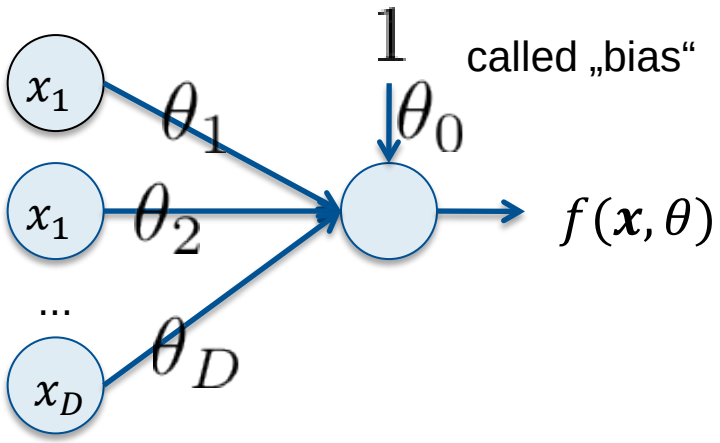


Neuron vs. Regression

- Edges multiply the signal (x_i) by some weight (θ_i) as a simple model
- Nodes sum inputs
- Equivalent to *linear regression*
- The bias can be threshold when the neuron fires

$$f(x, \theta) = \sum_{d=1}^{D} \theta_d x^d + \theta_0$$

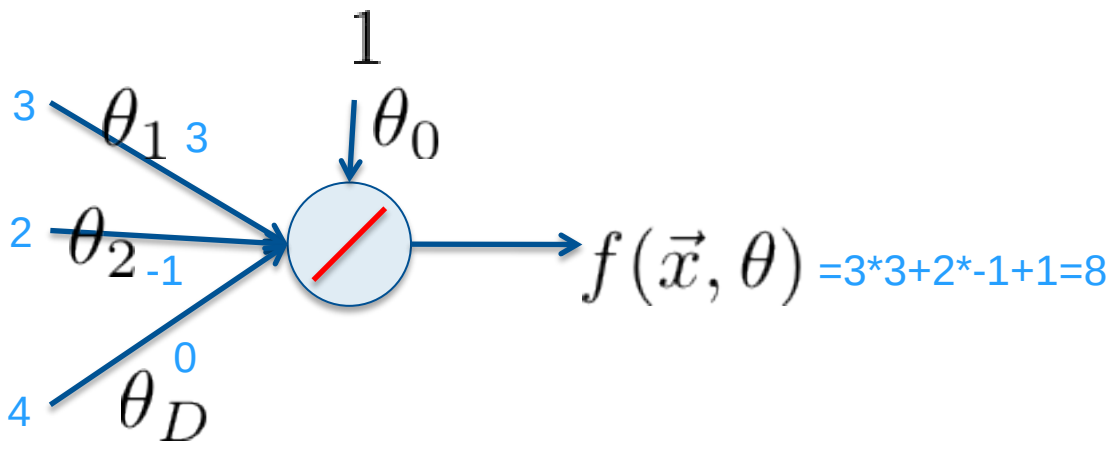




Linear Activation Function

Each neuron has its own bias and weights:

$$f(\vec{x},\theta) = \sum_{n=0}^{N-1} \sum_{d=1}^{D} \theta_d \phi_d(x_n) + \theta_0 \qquad f(\vec{x},\theta) = \theta^T \vec{x}$$

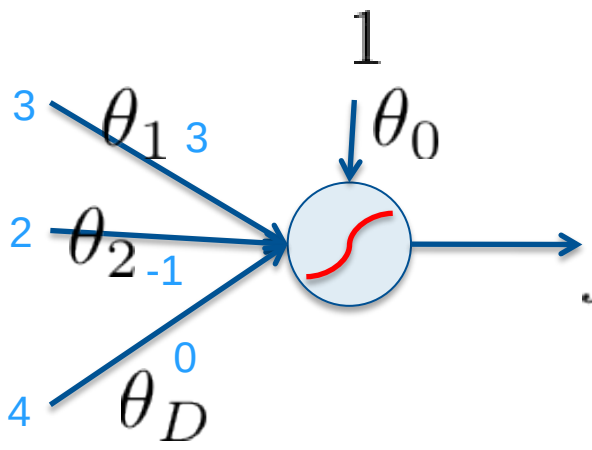


Linear neuron

Sigmoid Activation Function

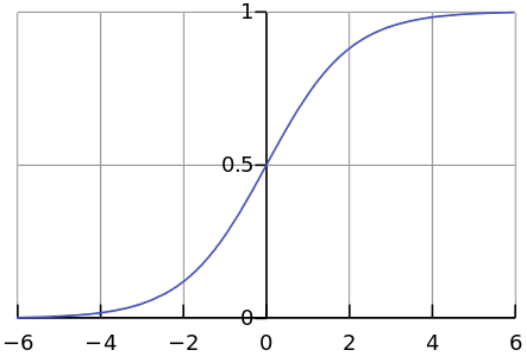
Sigmoid function (special case logistic function)

$$f(\vec{x},\theta) = g(\theta^T \vec{x}) \qquad g(z) = (1 + exp(-z))^{-1}$$



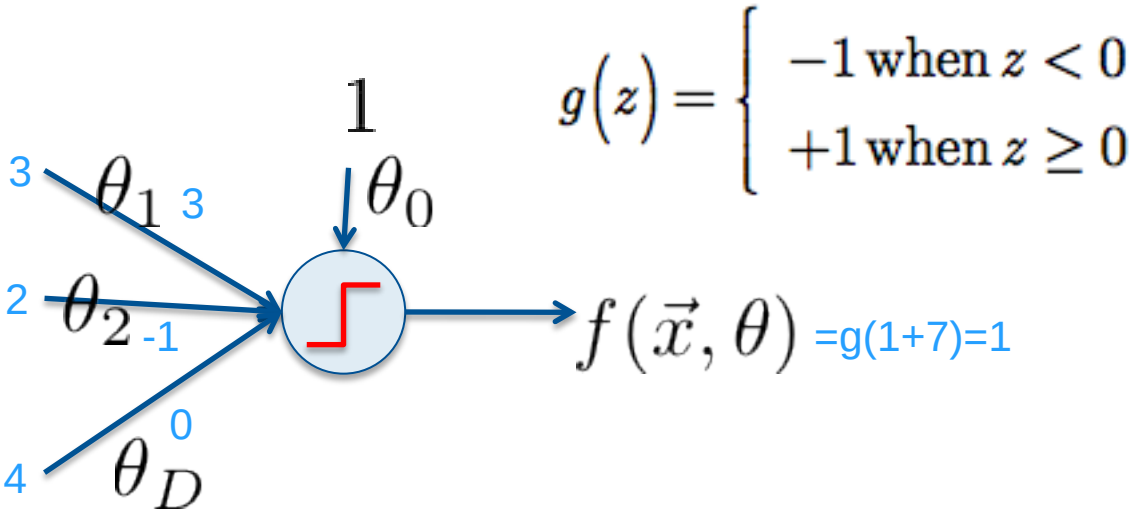
=(1+exp(-7))$^{-1}$=0.999

Logistic Neuron

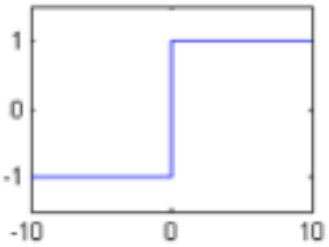


Perceptron

Binary classification via single-layer neural network



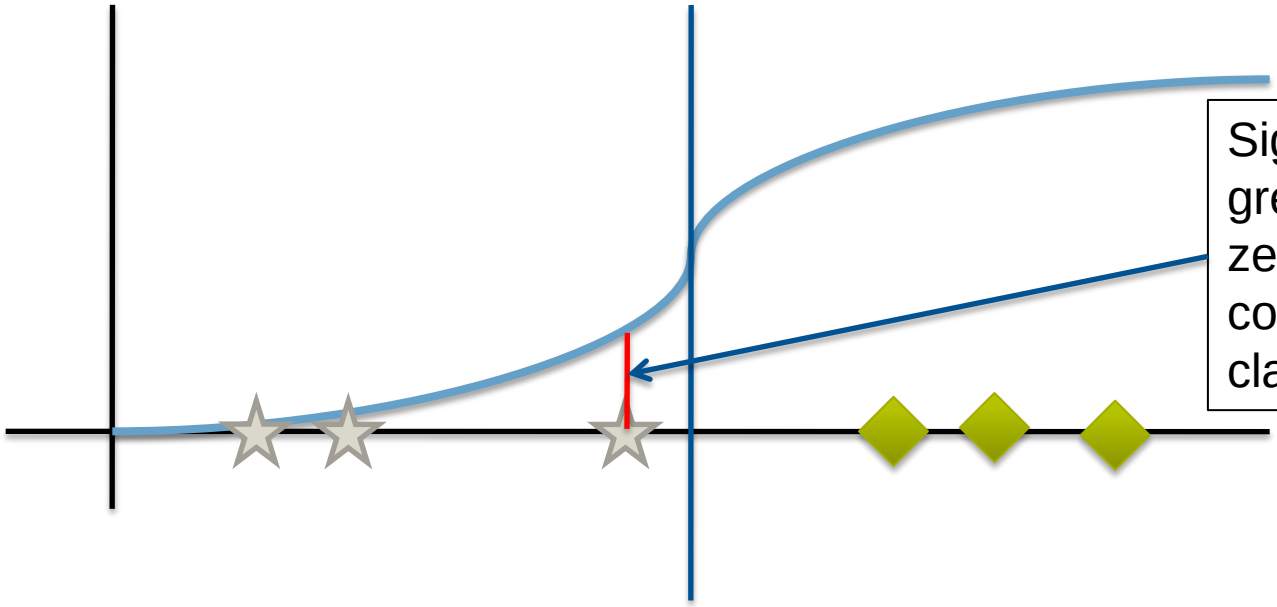
Classification error



Classification Error

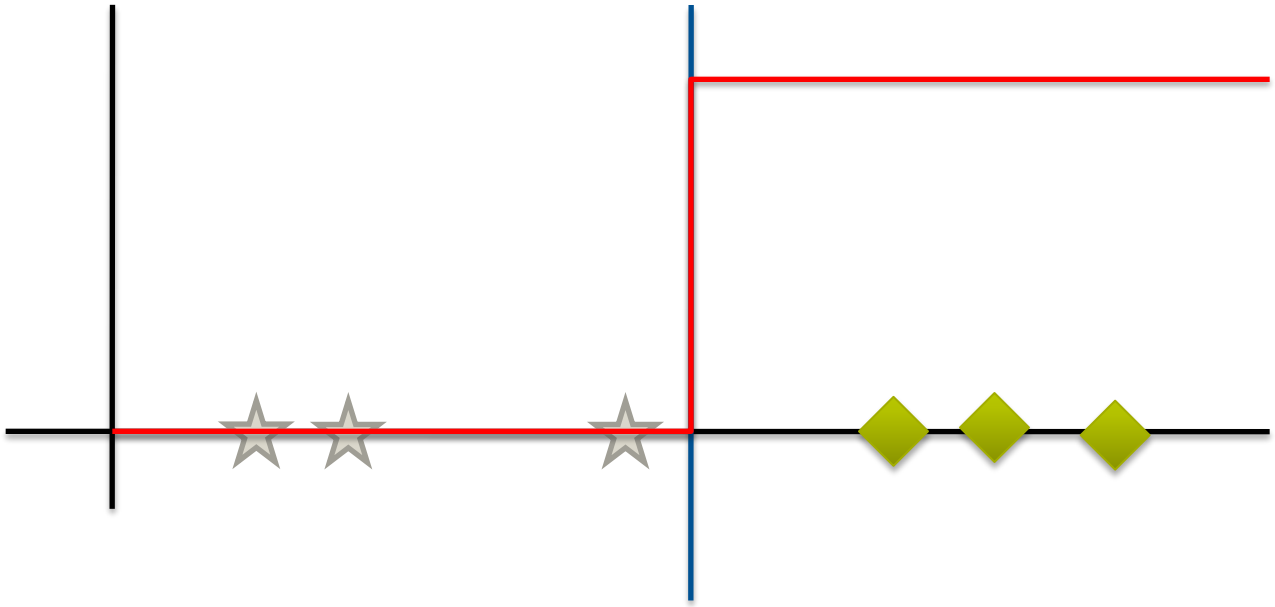
Only count errors when a classification is incorrect.

$$R(\theta) = \frac{1}{N}\sum_n L(y_n, f(x_n)) = \frac{1}{2N}\sum_n (y_n - g(\theta^T x_n))^2$$



Perceptron Error

We can’t do gradient descent (see next class) with step functions, as it is non-differentiable at one point and otherwise $\nabla_\theta R = 0$.

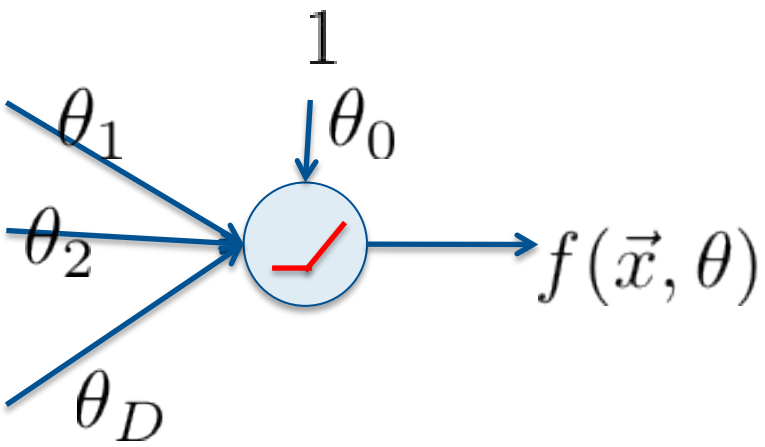


ReLU Activation Function

ReLU activation (is now often used in DNNs)

(see https://en.wikipedia.org/wiki/Activation_function)

$$f(x) = \max(0, x)$$



ReLU Neuron

Example

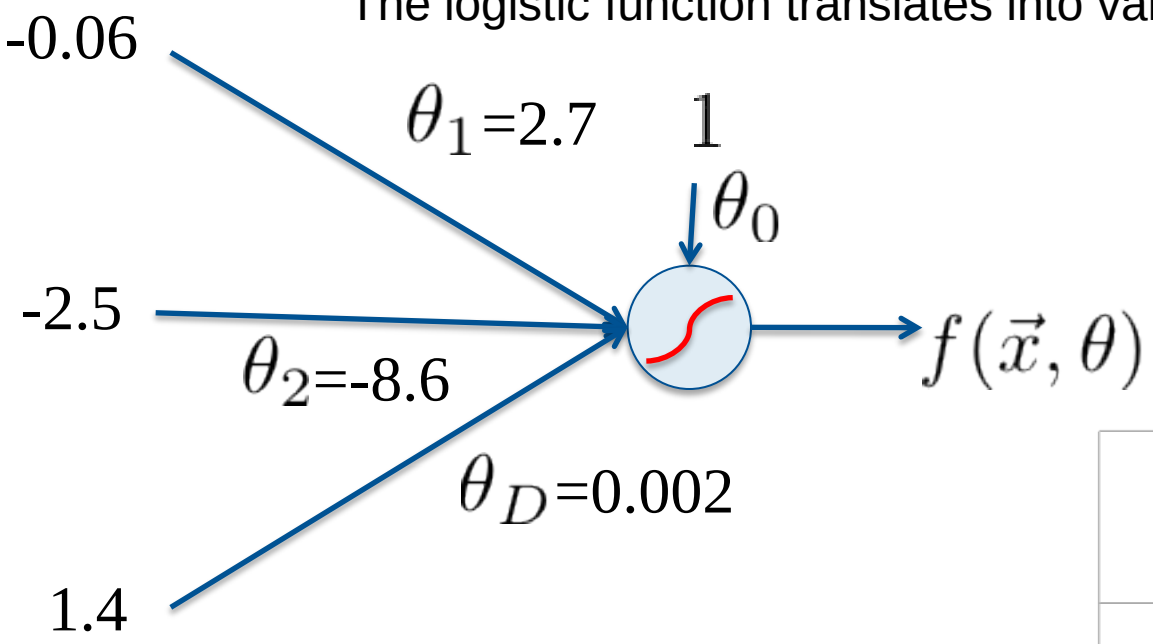
Activation function:

$$f(\vec{x}, \theta) = g(\theta^T \vec{x})$$

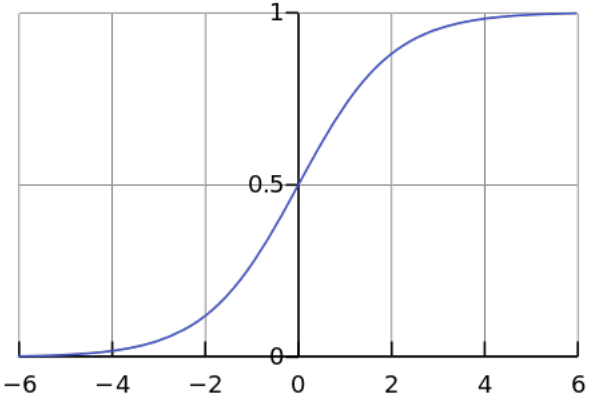
bias

$$\theta^T \vec{x} = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 + 4 = 25.34$$

The logistic function translates into values 0 to 1



$$g(z) = (1 + exp(-z))^{-1}$$

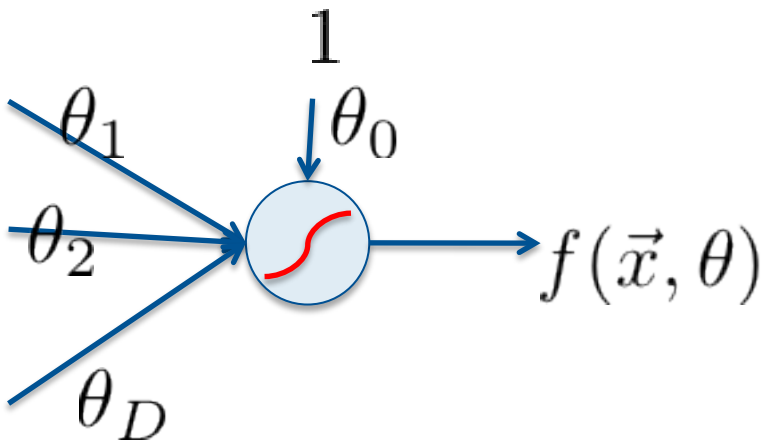


Example

Initialize with random weights

Training data set

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

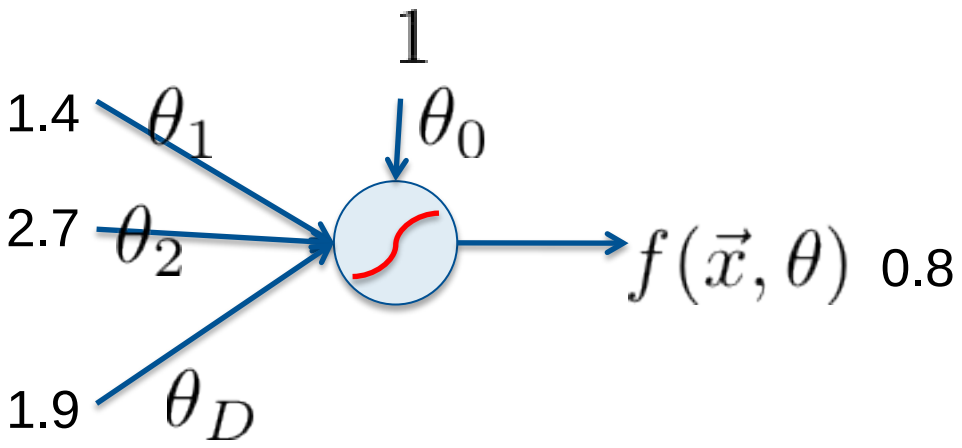


Example

Initialize with random weights
Present a training pattern
Feed it through to get output

Training data set

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

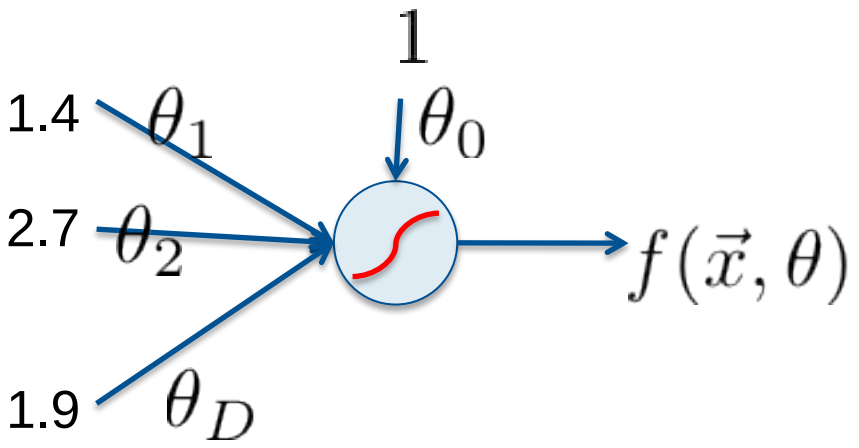


Example

Initialize with random weights
Present a training pattern
Feed it through to get output

Training data set

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



0.8

Correct 0

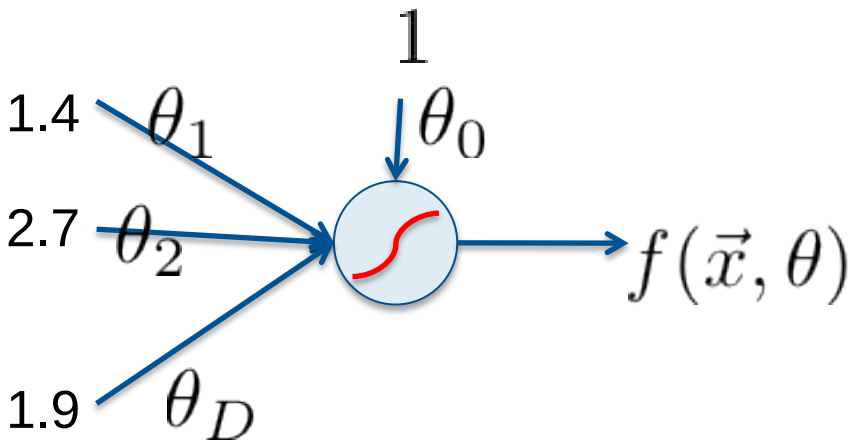
Error 0.8

Example

Initialize with random weights
Present a training pattern
Feed it through to get output
Adjust weights θ (see next class)

Training data set

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



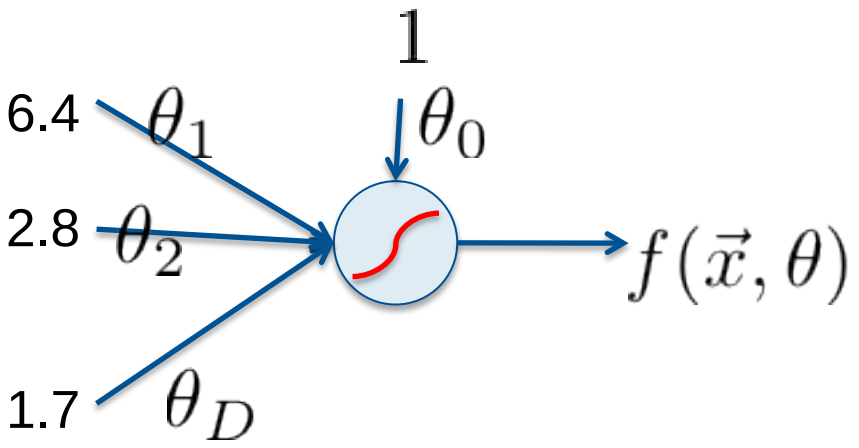
Example

Present another training pattern
Feed it through to get output
Adjust weights

Training data set

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

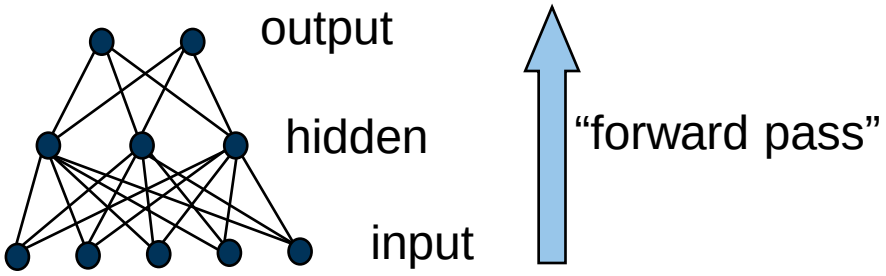
etc …



- Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Multi-Layer Feed-Forward Networks

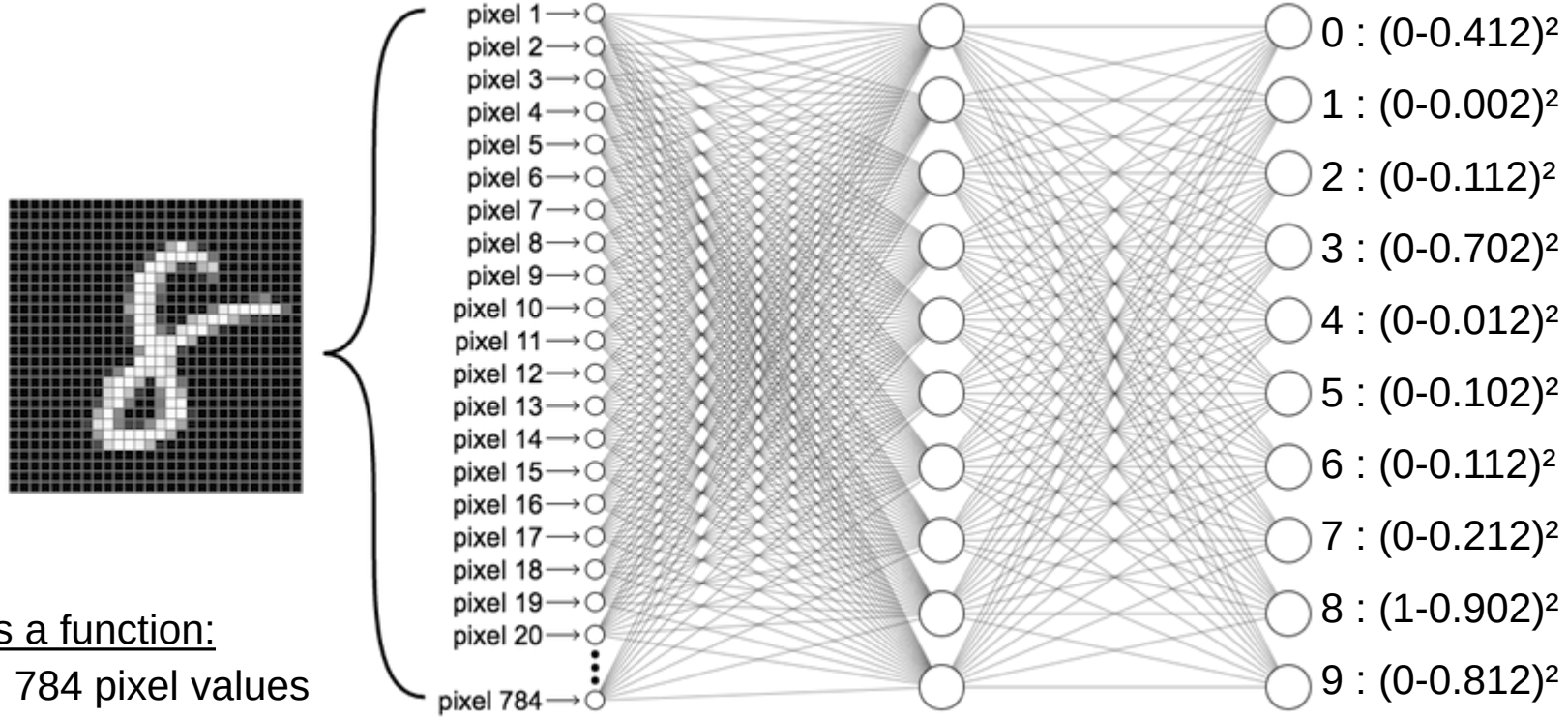
- Multi-layer networks can represent arbitrary non-linear functions.
- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.



- Multi-layer networks have thousands of weights and biases that need to be adapted. The weights determine the overall function computed.
- Weights are updated via backpropagation (i.e. gradient decent) (see next class)

Loss Function

Use a loss function (aka. cost function) to compute the average (misclassification) cost for all training data, i.e. for a particular set of weights and biases it computes a single number (average cost).



<u>NN as a function:</u>

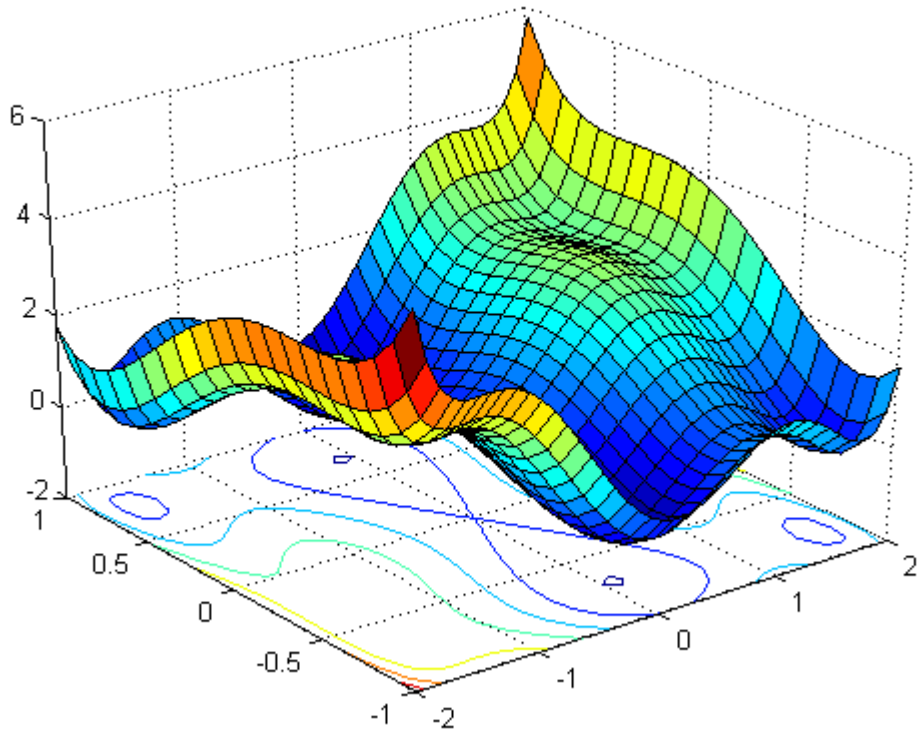
Input: 784 pixel values

Output: 10 numbers

Parameters: 784*10+10*10=7940 weights plus 20 biases

Learning as Minimizing a Loss Function

Next week we use the gradient descent algorithm to adapt the weights such that the loss function is minimized. For this we use the direction of steepest descent (negative gradient) of a function.



Cost function:
Input: 7960 parameters (weights + biases)
Output: scalar (loss)