Machine Learning and Pervasive Computing

Stephan Sigg

Georg-August-University Goettingen, Computer Networks

17.12.2014
Overview and Structure

22.10.2014 Organisation
22.10.3014 Introduction  (Def.: Machine learning, Supervised/Unsupervised, Examples)
29.10.2014 Machine Learning Basics  (Toolchain, Features, Metrics, Rule-based)
05.11.2014 A simple Supervised learning algorithm
12.11.2014 Excursion: Avoiding local optima with random search
19.11.2014 –
26.11.2014 Bayesian learner
03.12.2014 –
10.12.2014 Decision tree learner
17.12.2014 k-nearest neighbour
07.01.2015 Support Vector Machines
14.01.2015 Artificial Neural networks and Self Organizing Maps
21.01.2015 Hidden Markov models and Conditional random fields
28.01.2015 High dimensional data, Unsupervised learning
04.02.2015 Anomaly detection, Online learning, Recom. systems
Outline

Histogram methods

Parzen Estimator methods

Nearest neighbour techniques
Histogram methods

Alternative approach to function estimation: histogram methods

In general, the probability density of an event is estimated by dividing the range of $N$ values into bins of size $\Delta_i$.

Then, count the number of observations that fall inside bin $\Delta_i$. This is expressed as a normalised probability density

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

Accuracy of the estimation is dependent on the width of the bins.

Approach well suited for big data since the data items can be discarded once the histogram is created.
Histogram methods

**Curse of dimensionality**

Due to the edges of the bins, the modelled distribution is characterised by discontinuities not present in the underlying distribution observed.

The method does not scale well with increasing dimension.
The curse of dimensionality

Issues related to high dimensional input data:

**Exponential growth** When dividing the space into bins, the number of bins grows exponentially with dimension
The curse of dimensionality

Issues related to high dimensional input data:

**Exponential growth** When dividing the space into bins, the number of bins grows exponentially with dimension.

**Counter-intuitive properties** Higher dimensional spaces can have counter-intuitive properties (see example on next slides)
The curse of dimensionality

Example – Volume of a sphere

Consider a sphere of radius $r = 1$ in a $D$-dimensional space
The curse of dimensionality

Example – Volume of a sphere

Consider a sphere of radius $r = 1$ in a $D$-dimensional space. What is the fraction of the volume of the sphere that lies between radius $r = 1$ and $r' = 1 - \varepsilon$?
The curse of dimensionality

Example – Volume of a sphere

Consider a sphere of radius $r = 1$ in a $D$-dimensional space.
What is the fraction of the volume of the sphere that lies between radius $r = 1$ and $r' = 1 - \varepsilon$?

We can estimate the volume of a sphere with radius $r$ as

$$V_D(r) = \delta_D r^D$$

for appropriate $\delta$. 

The curse of dimensionality

Example – Volume of a sphere

We can estimate the volume of a sphere with radius $r$ as

$$V_D(r) = \delta_D r^D$$

for appropriate $\delta$
The curse of dimensionality

Example – Volume of a sphere

We can estimate the volume of a sphere with radius \( r \) as

\[
V_D(r) = \delta_D r^D
\]

for appropriate \( \delta \)

The required fraction is given by

\[
\frac{V_D(1) - V_D(1 - \varepsilon)}{V_D(1)} = 1 - (1 - \varepsilon)^D
\]
The curse of dimensionality

Example – Volume of a sphere

The required fraction is given by

\[
\frac{V_D(1) - V_D(1 - \varepsilon)}{V_D(1)} = 1 - (1 - \varepsilon)^D
\]
The curse of dimensionality

Example – Volume of a sphere

The required fraction is given by

\[
\frac{V_D(1) - V_D(1 - \varepsilon)}{V_D(1)} = 1 - (1 - \varepsilon)^D
\]

For large D, this fraction tends to 1

In high dimensional spaces, most of the volume of a sphere is concentrated near the surface
The curse of dimensionality
The curse of dimensionality

Example – Gaussian distribution

The probability mass of the gaussian distribution is concentrated in a thin shell (here plotted as distance from the origin in a polar coordinate system)
The curse of dimensionality

Discussion
While the curse of dimensionality induces severe problems, we will investigate effective techniques applicable to high-dimensional spaces.
Outline

Histogram methods

Parzen Estimator methods

Nearest neighbour techniques
Parzen estimator methods

Assume an unknown probability density $\mathcal{P}(\cdot)$

We want to estimate the probability density $\mathcal{P}(\overrightarrow{x})$ of $\overrightarrow{x}$ in a $\mathcal{D}$-dimensional Euclidean space

Consider a small region $\mathcal{R}$ around $\overrightarrow{x}$:

$$P = \int_{\mathcal{R}} \mathcal{P}(\overrightarrow{x}) d\overrightarrow{x}$$
Parzen estimator methods

We utilise a data set of $N$ observations.

Each observation has a probability of $P$ to fall inside $\mathcal{R}$.

With the binomial distribution we can calculate the count $K$ of points falling into $\mathcal{R}$:

$$\text{Bin}(K|N, P) = \frac{N!}{K!(N - K)!} P^K (1 - P)^{N-K}$$
Parzen estimator methods

We utilise a data set of $N$ observations

Each observation has a probability of $P$ to fall inside $\mathcal{R}$

With the binomial distribution we can calculate the count $K$ of points falling into $\mathcal{R}$:

$$
\text{Bin}(K|N, P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{N-K}
$$

For large $N$ we can show

$$
K \approx NP
$$

With sufficiently small $\mathcal{R}$ we can also show for the volume $V$ of $\mathcal{R}$

$$
P \approx \mathcal{P}(\vec{x}) V
$$

Therefore, we can estimate the density as

$$
\mathcal{P}(\vec{x}) = \frac{K}{NV}
$$
Parzen estimator methods

We assume that $\mathcal{R}$ is a small hypercube

In order to count the number $K$ of points that fall inside $\mathcal{R}$ we define

$$k(\overrightarrow{u}) = \begin{cases} 
1, & |u_i| \leq \frac{1}{2}, \quad i = 1, \ldots, D, \\
0, & \text{otherwise}
\end{cases}$$

This represents a unit cube centred around the origin

This function is an example of a kernel-function or Parzen window
Parzen estimator methods

\[
k(\vec{u}) = \begin{cases} 
   1, & |u_i| \leq \frac{1}{2}, \quad i = 1, \ldots, D, \\
   0, & \text{otherwise}
\end{cases}
\]

When the measured data point \( \vec{x}_n \) lies inside a cube of side \( h \) centred around \( \vec{x} \), we have

\[
k \left( \frac{\vec{x} - \vec{x}_n}{h} \right) = 1
\]

The total count \( K \) of points that fall inside this cube is

\[
K = \sum_{n=1}^{N} k \left( \frac{\vec{x} - \vec{x}_n}{h} \right)
\]
Parzen estimator methods

The total count $K$ of points that fall inside this cube is

$$K = \sum_{n=1}^{N} k \left( \frac{\vec{x} - \vec{x}_n}{h} \right)$$

When we substitute this in the density estimate derived above

$$P(\vec{x}) = \frac{K}{NV}$$

with volume $V = h^D$ we obtain the overall density estimate as

$$P(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} \left( \frac{\vec{x} - \vec{x}_n}{h} \right)$$
Parzen estimator methods

\[ P(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} \left( \frac{\vec{x} - \vec{x}_n}{h} \right) \]

Again, this density estimator suffers from artificial discontinuities
(Due to the fixed boundaries of the cubes)

Problem can be overcome by choosing a smoother kernel function
(A common choice is a Gaussian kernel with a standard deviation \(\sigma\))

\[ P(\vec{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} e^{-\frac{||\vec{x} - \vec{x}_n||^2}{2\sigma^2}} \]
Parzen estimator methods

Density estimation for various values of $\sigma$

\(\sigma = 0.005\)

\(\sigma = 0.07\)

\(\sigma = 0.2\)
Outline

Histogram methods

Parzen Estimator methods

Nearest neighbour techniques
Nearest neighbour methods

A problem with Parzen estimator methods is that the parameter governing the kernel width ($h$ or $\sigma$) is fixed for all values $x$.

In regions with

...high density, a wide kernel might lead to over-smoothing
...low density, the same width may lead to noisy estimates
Nearest neighbour methods

NN-methods address this by adapting width to data density.

Parzen estimator methods fix $V$ and determine $K$ from the data.
Nearest neighbour methods fix $K$ and choose $V$ accordingly.

Again, we consider a point $\vec{x}$ and estimate the density $P(\vec{x})$.

The radius of the sphere is increased until $K$ data points (the nearest neighbours) are covered.
Nearest neighbour methods

The value $K$ then controls the amount of smoothing.

Again, an optimum value for $K$ exists.
Nearest neighbour methods

**Classification**: Apply KNN-density estimation for each class

Assume data set of $N$ points with $N_k$ points in class $C_k$

To classify sample $\overrightarrow{x}$, draw a sphere containing $K$ points around $\overrightarrow{x}$

Sphere can contain other points regardless of their class

Assume sphere has volume $V$ and contains $K_k$ points from $C_k$
**Nearest neighbour methods**

**Assume:** Sphere of volume $V$ contains $K_k$ points from class $C_k$

We estimate the density of class $C_k$ as

$$P(\vec{x}|C_k) = \frac{K_k}{N_k V}$$

The unconditional density is given as

$$P(\vec{x}) = \frac{K}{N V}$$

The probability to experience a class $C_k$ is given as

$$P(C_k) = \frac{N_k}{N}$$

With Bayes theorem we can combine this to achieve

$$P(C_k|\vec{x}) = \frac{P(\vec{x}|C_k)P(C_k)}{P(\vec{x})} = \frac{K_k}{K}$$
Nearest neighbour methods

\[ P(C_k | \vec{x}) = \frac{P(\vec{x} | C_k) P(C_k)}{P(\vec{x})} = \frac{K_k}{K} \]

To minimise the probability of misclassification, assign \( \vec{x} \) to class with the largest probability

This corresponds to the largest value of

\[ \frac{K_k}{K} \]
Nearest neighbour methods

To classify a point, we identify the $K$ nearest points
And assign the point to the class having most representatives in this set

Choice $K = 1$ is called nearest neighbour rule

For this choice, the error rate is never more than twice the minimum achievable error rate of an optimum classifier

---

Nearest neighbour methods

Classification of points by the K-nearest neighbour classifier
Nearest neighbour methods

Classification of points by the K-nearest neighbour classifier
Nearest neighbour methods

Instance-based learning

In instance-based learning, classification is not derived from rules but from the instances themselves.
Nearest neighbour methods

Instance-based learning

In instance-based learning, classification is not derived from rules but from the instances themselves.

Nearest neighbour classification

New instances are compared with existing ones and nearest neighbour is used to predict a class.
Nearest neighbour methods

**Instance-based learning**

In instance-based learning, classification is not derived from rules but from the instances themselves.

**Nearest neighbour classification** New instances are compared with existing ones and nearest neighbour is used to predict a class.

**k-nearest neighbour** Majority vote among $k$ nearest neighbours.
Nearest neighbour methods

**Storage demands**  KNN and Parzen-method are not well suited for large data sets since they require the entire data set to be stored
Nearest neighbour methods

**Storage demands** KNN and Parzen-method are not well suited for large data sets since they require the entire data set to be stored.

**Instance-based learning** Distance function to determine which member of a training set is closest to an unknown test instance.

⇒ How to calculate the distance?
Nearest neighbour methods

Storage demands  KNN and Parzen-method are not well suited for large data sets since they require the entire data set to be stored

Instance-based learning  Distance function to determine which member of a training set is closest to an unknown test instance

⇒ How to calculate the distance?

Low Classification speed  The intuitive way to find nearest neighbours involves linear comparison to all training examples

⇒ Can we store and process data more efficiently?
Nearest neighbour methods

Storage demands  KNN and Parzen-method are not well suited for large data sets since they require the entire data set to be stored

Instance-based learning  Distance function to determine which member of a training set is closest to an unknown test instance

⇒ How to calculate the distance?

Low Classification speed  The intuitive way to find nearest neighbours involves linear comparison to all training examples

⇒ Can we store and process data more efficiently?
Nearest neighbour methods

Distance function

Most instance-based learners utilise Euclidean distance:

\[ \sqrt{(v_1 - v'_1)^2 + (v_2 - v'_2)^2 + \cdots + (v_k - v'_k)^2} \]
Nearest neighbour methods

Distance function

Most instance-based learners utilise Euclidean distance:

$$\sqrt{(v_1 - v'_1)^2 + (v_2 - v'_2)^2 + \cdots + (v_k - v'_k)^2}$$

Alternatives:  
- Manhattan Distance

$$(v_1 - v'_1) + (v_2 - v'_2) + \cdots + (v_k - v'_k)$$
Nearest neighbour methods

Distance function

Most instance-based learners utilise Euclidean distance:

$$\sqrt{(v_1 - v_1')^2 + (v_2 - v_2')^2 + \cdots + (v_k - v_k')^2}$$

Alternatives:

- **Manhattan Distance**

  $$(v_1 - v_1') + (v_2 - v_2') + \cdots + (v_k - v_k')$$

- **Powers higher than square**

  Increase the influence of large differences at the expense of small differences

It is important to think of actual instances and what it means for them to be separated by a certain distance.
Nearest neighbour methods

Different scales

Different features often follow different scales.

Normalize! It is usually a good idea to normalize all features first

\[ \text{feature}_i = \frac{v_i - \min(v_i)}{\max(v_i) - \min(v_i)} \]
Nearest neighbour methods
Nominal features

Nominal features that take symbolic rather than numeric values, have to be handled differently
Nearest neighbour methods
Nominal features

Nominal features that take symbolic rather than numeric values, have to be handled differently

Common solution:

Features with identical value  Distance = 0
Features with different value  Distance = 1

More expressive metric:  e.g. hue in color space for colors
Nearest neighbour methods

Missing feature values

For missing feature values, commonly the distance is chosen as large as possible

(if both are missing, Distance= 1, if only one is missing)
Nearest neighbour methods

Missing feature values

For missing feature values, commonly the distance is chosen as large as possible

(if both are missing, Distance = 1, if only one is missing)

**Nominal features**  Distance = 1

**Numeric features**  Distance = $\max(v', 1 - v')$ (where $v'$ is the (normalized) value to compare to)
Nearest neighbour methods
Finding nearest neighbours efficiently

Instance-based learning is often slow
Intuitive way to find nearest neighbour is to iteratively compare to all training examples
Nearest neighbour methods
Finding nearest neighbours efficiently

Instance-based learning is often slow
  Intuitive way to find nearest neighbour is to iteratively compare to all training examples

More efficient search for nearest neighbour possible by kD-tree
  Binary tree that stores points from k-dimensional space
Nearest neighbour methods

**kD-Trees**

- Each region contains 0-1 points
- Every point in the training set corresponds to a single node
- Up to half the nodes are at the leaves of the tree
Nearest neighbour methods

Find nearest Neighbour in a kD-Tree:

1. Traverse the nodes of the tree to locate the region containing the point.
2. Check parent nodes and other siblings of the parent for their distance to the point. It might be necessary to recursively repeat for parent node.
Nearest neighbour methods

Find nearest Neighbour in a kD-Tree:

1. Traverse the nodes of the tree to locate the region containing the point
2. Check parent nodes and other siblings of the parent for their distance to the point. It might be necessary to recursively repeat for parent node.
Nearest neighbour methods

Creating good kD-Trees

Unbalanced trees lead to low efficiency gain
Backtracking cost is lowest for approximately square regions
Nearest neighbour methods

Creating good kD-Trees

1. How to select good first instance to split on?
2. How to determine dimension of split?

- Find dimension for a split
  - Calculate variance of data points along each axis individually
  - Select axis with greatest variance and create hyperplane perpendicular to it
  - Split perpendicular to direction of greatest spread

- Find good first instance for a split
  - Calculate median along that axis and select the corresponding point
  - Half of the points lie on either side of the split
Nearest neighbour methods

Creating good kD-Trees

1. How to select good first instance to split on?
2. How to determine dimension of split?

Find dimension for a split

Calculate variance of data points along each axis individually
Select axis with greatest variance and create hyperplane perpendicular to it
Split perpendicular to direction of greatest spread
Nearest neighbour methods

Creating good kD-Trees

1. How to select good first instance to split on?
2. How to determine dimension of split?

Find dimension for a split

- Calculate variance of data points along each axis individually
- Select axis with greatest variance and create hyperplane perpendicular to it
  
  Split perpendicular to direction of greatest spread

Find good first instance for a split

- Calculate median along that axis and select the corresponding point
  
  Half of the points lie on either side of the split
Nearest neighbour methods

kD-trees – online updates

- On-line learning with kD-trees is possible by appending new samples to an existing tree
Nearest neighbour methods
kD-trees – online updates

- On-line learning with kD-trees is possible by appending new samples to an existing tree
  - Traverse each new sample down to a leaf of an existing tree to find its hyperrectangle
  - If empty, place new point there
  - else, Split hyperrectangle along its longest dimension
Nearest neighbour methods

kD-trees – high dimensions

A problem with kD-trees

- Especially in higher dimensions:
- Corners of rectangles may contain points farther to the center of the enclosing rectangle than to other rectangles
- Then, unnecessarily many branches of the tree are considered which reduces efficiency
Nearest neighbour methods

Ball trees

- Use hyperspheres not hyperrectangles
- Binary tree
- Each node defines the center and radius of the smallest hypersphere that contains all nodes of its sub-trees

**Recursive construction:** Split data into two sets along the dimension with greatest spread (split at median point)
Nearest neighbour methods

Ball trees
Outline

Histogram methods

Parzen Estimator methods

Nearest neighbour techniques
Questions?

Stephan Sigg
stephan.sigg@cs.uni-goettingen.de
Literature