Finding Predictors: Nearest Neighbor

- Modern Motivations: Be Lazy!
  - Classification
  - Regression
  - Choosing the right number of neighbors

- Some Optimizations

- Other types of lazy algorithms
Motivation: A Zoo

Given: Information about animals in the zoo
How can we classify new animals?
Motivation: A Zoo

Given: Information about animals in the zoo
How can we classify new animals?
Motivation: A Zoo

Given: Information about animals in the zoo
How can we classify new animals?
Motivation: A Zoo

Given: Information about animals in the zoo
How can we classify new animals?
Motivation: A Zoo

Given: Information about animals in the zoo
How can we classify new animals?
Unsupervised vs. Supervised Learning

- **Unsupervised**: No class information given. Goal: detect unknown patterns (e.g. clusters, association rules)
- **Supervised**: Class information exists/is provided by *supervisor*. Goal: learn class structure for future unclassified/unknown data.
• How do Expert Systems work?
Motivation: Expert/Legal Systems/Streber

- How do Expert Systems work?
  - Find most similar case(s).
Motivation: Expert/Legal Systems/Streber

- How do Expert Systems work?
  - Find most similar case(s).
- How does the American Justice System work?
Motivation: Expert/Legal Systems/Streber

- How do Expert Systems work?
  - Find most similar case(s).
- How does the American Justice System work?
  - Find most similar case(s).
Motivation: Expert/Legal Systems/Streber

- How do Expert Systems work?
  - Find most similar case(s).
- How does the American Justice System work?
  - Find most similar case(s).
- How does the nerd ("Streber") learn?
Motivation: Expert/Legal Systems/Streber

- How do Expert Systems work?
  - Find most similar case(s).
- How does the American Justice System work?
  - Find most similar case(s).
- How does the nerd ("Streber") learn?
  - He learns by heart.
Eager vs. Lazy Learners

- **Lazy:** Save all data from training, use it for classifying
  (The learner was lazy, classifier has to do the work)

- **Eager:** Builds a (compact) model/structure during training, use model for classification.
  (The learner was eager/worked harder, classifier has a simple life.)
Nearest neighbour predictors

Nearest neighbour predictors are special case of instance-based learning.
Nearest neighbour predictors are special case of instance-based learning.

Instead of constructing a model that generalizes beyond the training data, the training examples are merely stored.
Nearest neighbour predictors are special case of instance-based learning.

Instead of constructing a model that generalizes beyond the training data, the training examples are merely stored.

Predictions for new cases are derived directly from these stored examples and their (known) classes or target values.
Simple nearest neighbour predictor

For a new instance, use the target value of the closest neighbour in the training set.
Simple nearest neighbour predictor

For a new instance, use the target value of the closest neighbour in the training set.

Classification

Regression

©Michael R. Berthold, Christian Borgelt, Frank Höppner, Frank Klawonn and Iris Adä
Nearest Neighbour Predictor: Issues

Noisy Data is a problem:

How can we fix this?
Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the \( k \) \((k > 1)\) are taken into account, leading to the \( k \)-nearest neighbour predictor.

- Classification: Choose the majority class among the \( k \) nearest neighbours for prediction.
- Regression: Take the mean value of the \( k \) nearest neighbours for prediction.

All \( k \) nearest neighbours have the same influence on the prediction. Closer nearest neighbours should have higher influence on the prediction.
Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the $k$ ($k > 1$) are taken into account, leading to the $k$-nearest neighbour predictor.

Classification: Choose the majority class among the $k$ nearest neighbours for prediction.

Regression: Take the mean value of the $k$ nearest neighbours for prediction.

All $k$ nearest neighbours have the same influence on the prediction. Closer nearest neighbours should have higher influence on the prediction.
• Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the $k$ ($k > 1$) are taken into account, leading to the $k$-nearest neighbour predictor.

• Classification: Choose the majority class among the $k$ nearest neighbours for prediction.

• Regression: Take the mean value of the $k$ nearest neighbours for prediction.
Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the $k$ ($k > 1$) are taken into account, leading to the $k$-nearest neighbour predictor.

- Classification: Choose the majority class among the $k$ nearest neighbours for prediction.
- Regression: Take the mean value of the $k$ nearest neighbours for prediction.

Disadvantage:
Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the $k$ ($k > 1$) are taken into account, leading to the $k$-nearest neighbour predictor.

- **Classification:** Choose the majority class among the $k$ nearest neighbours for prediction.
- **Regression:** Take the mean value of the $k$ nearest neighbours for prediction.

**Disadvantage:**
- All $k$ nearest neighbours have the same influence on the prediction.
k-nearest neighbour predictor

- Instead of relying for the prediction on only one instance, the (single) nearest neighbour, usually the \( k \) \((k > 1)\) are taken into account, leading to the \( k\)-nearest neighbour predictor.

- Classification: Choose the majority class among the \( k\) nearest neighbours for prediction.

- Regression: Take the mean value of the \( k\) nearest neighbours for prediction.

Disadvantage:

- All \( k\) nearest neighbours have the same influence on the prediction.

- Closer nearest neighbours should have higher influence on the prediction.
**Distance Metric:**
The distance metric, together with a possible task-specific scaling or weighting of the attributes, determines which of the training examples are nearest to a query data point and thus selects the training example(s) used to produce a prediction.
Ingredients for the k-nearest neighbour predictor

- **Distance Metric:**
  The distance metric, together with a possible task-specific scaling or weighting of the attributes, determines which of the training examples are nearest to a query data point and thus selects the training example(s) used to produce a prediction.

- **Number of Neighbours:**
  The number of neighbours of the query point that are considered can range from only one (the basic nearest neighbour approach) through a few (like $k$-nearest neighbour approaches) to, in principle, all data points as an extreme case.
Ingredients for the k-nearest neighbour predictor

- **Distance Metric:**
  The distance metric, together with a possible task-specific scaling or weighting of the attributes, determines which of the training examples are nearest to a query data point and thus selects the training example(s) used to produce a prediction.

- **Number of Neighbours:**
  The number of neighbours of the query point that are considered can range from only one (the basic nearest neighbour approach) through a few (like $k$-nearest neighbour approaches) to, in principle, all data points as an extreme case (would that be a good idea?).
Ingredients for the k-nearest neighbour predictor

- **weighting function for the neighbours**
  Weighting function defined on the distance of a neighbour from the query point, which yields higher values for smaller distances.
Ingredients for the k-nearest neighbour predictor

- **weighting function for the neighbours**
  Weighting function defined on the distance of a neighbour from the query point, which yields higher values for smaller distances.

- **prediction function**
  For multiple neighbours, one needs a procedure to compute the prediction from the (generally differing) classes or target values of these neighbours, since they may differ and thus may not yield a unique prediction directly.
k Nearest neighbour predictor

Average (3 nearest neighbours)  Distance weighted (2 nearest neighbours)
Nearest neighbour predictor

Choosing the “ingredients”
Choosing the “ingredients”

- **distance metric**
  
  Problem dependent. Often Euclidean distance (after normalisation).
Nearest neighbour predictor

Choosing the “ingredients”

- **distance metric**
  Problem dependent. Often Euclidean distance (after normalisation).

- **number of neighbours**
  Very often chosen on the basis of cross-validation. Choose $k$ that leads to the best performance for cross-validation.
Nearest neighbour predictor

Choosing the “ingredients”

- **distance metric**
  Problem dependent. Often Euclidean distance (after normalisation).

- **number of neighbours**
  Very often chosen on the basis of cross-validation. Choose \( k \) that leads to the best performance for cross-validation.

- **weighting function for the neighbours**

  E.g. tricubic weighting function:
  \[
  w(s_i, q, k) = \left( 1 - \left( \frac{d(s_i, q)}{d_{\text{max}}(q, k)} \right)^3 \right)^3
  \]

  - \( q \) Query point
  - \( s_i \) (input vector of) the \( i \)-th nearest neighbour of \( q \) in the training data set
  - \( k \) number of considered neighbours
  - \( d \) employed distance function
  - \( d_{\text{max}}(q, k) \) maximum distance between any two nearest neighbours and the distances of the nearest neighbours to the query point
Choosing the “ingredients”

- **prediction function**

  - **Regression:**
    
    Compute the weighted average of the target values of the nearest neighbours.

  - **Classification:**
    
    - Sum up the weights for each class among the nearest neighbours.
    - Choose the class with the highest value (or incorporate a cost matrix and interpret the summed weights for the classes as likelihoods).
A $k$-nearest neighbour predictor with a weighting function can be interpreted as an $n$-nearest neighbour predictor with a modified weighting function where $n$ is the number of (training) data.
A \( k \)-nearest neighbour predictor with a weighting function can be interpreted as an \( n \)-nearest neighbour predictor with a modified weighting function where \( n \) is the number of (training) data.

The modified weighting function simply assigns the weight 0 to all instances that do not belong to the \( k \) nearest neighbours.
A \( k \)-nearest neighbour predictor with a weighting function can be interpreted as an \( n \)-nearest neighbour predictor with a modified weighting function where \( n \) is the number of (training) data.

The modified weighting function simply assigns the weight 0 to all instances that do not belong to the \( k \) nearest neighbours.

More general approach: Use a general kernel function that assigns a distance-dependent weight to all instances in the training data set.
Kernel functions

Such a kernel function $K$ assigning a weight to each data point that depends on its distance $d$ to the query point should satisfy the following properties:
Kernel functions

Such a kernel function $K$ assigning a weight to each data point that depends on its distance $d$ to the query point should satisfy the following properties:

- $K(d) \geq 0$
Kernel functions

Such a kernel function $K$ assigning a weight to each data point that depends on its distance $d$ to the query point should satisfy the following properties:

- $K(d) \geq 0$
- $K(0) = 1$ (or at least, $K$ has its mode at 0)
Kernel functions

Such a kernel function $K$ assigning a weight to each data point that depends on its distance $d$ to the query point should satisfy the following properties:

- $K(d) \geq 0$
- $K(0) = 1$ (or at least, $K$ has its mode at 0)
- $K(d)$ decreases monotonously with increasing $d$. 
Kernel functions

Typical examples for kernel functions ($\sigma > 0$ is a predefined constant):

- $K_{\text{rect}}(d) = \begin{cases} 1 & \text{if } d \leq \sigma, \\ 0 & \text{otherwise} \end{cases}$
Kernel functions

Typical examples for kernel functions ($\sigma > 0$ is a predefined constant):

- $K_{\text{rect}}(d) = \begin{cases} 1 & \text{if } d \leq \sigma, \\ 0 & \text{otherwise} \end{cases}$

- $K_{\text{triangle}}(d) = K_{\text{rect}}(d) \cdot (1 - d/\sigma)$
Kernel functions

Typical examples for kernel functions ($\sigma > 0$ is a predefined constant):

- $K_{\text{rect}}(d) = \begin{cases} 1 & \text{if } d \leq \sigma, \\ 0 & \text{otherwise} \end{cases}$

- $K_{\text{triangle}}(d) = K_{\text{rect}}(d) \cdot (1 - d/\sigma)$

- $K_{\text{tricubic}}(d) = K_{\text{rect}}(d) \cdot (1 - d^3/\sigma^3)^3$
Kernel functions

Typical examples for kernel functions ($\sigma > 0$ is a predefined constant):

- $K_{\text{rect}}(d) = \begin{cases} 
1 & \text{if } d \leq \sigma, \\
0 & \text{otherwise}
\end{cases}$

- $K_{\text{triangle}}(d) = K_{\text{rect}}(d) \cdot (1 - d/\sigma)$

- $K_{\text{tricubic}}(d) = K_{\text{rect}}(d) \cdot (1 - d^3/\sigma^3)^3$

- $K_{\text{gauss}}(d) = \exp \left( -\frac{d^2}{2\sigma^2} \right)$
Locally weighted (polynomial) regression

- For regression problems: So far, weighted averaging of the target values.

- Instead of a simple weighted average, one can also compute a (local) regression function at the query point taking the weights into account.
Locally weighted polynomial regression

Kernel weighted regression (left) vs. distance-weighted 4-nearest neighbour regression (tricubic weighting function, right) in one dimension.
Adjusting the distance function

- The choice of the distance function is crucial for the success of a nearest neighbour approach.
Adjusting the distance function

- The choice of the distance function is crucial for the success of a nearest neighbour approach.

- One can try to adapt the distance function.
Adjusting the distance function

- The choice of the distance function is crucial for the success of a nearest neighbour approach.

- One can try to adapt the distance function.

- One way to adapt the distance function is feature weights to put a stronger emphasis on those features that are more important.
The choice of the distance function is crucial for the success of a nearest neighbour approach.

One can try to adapt the distance function.

One way to adapt the distance function is feature weights to put a stronger emphasis on those features that are more important.

A configuration of feature weights can be evaluated based on cross-validation.
Adjusting the distance function

- The choice of the distance function is crucial for the success of a nearest neighbour approach.

- One can try to adapt the distance function.

- One way to adapt the distance function is feature weights to put a stronger emphasis on those features that are more important.

- A configuration of feature weights can be evaluated based on cross-validation.

- The optimisation of the feature weights can then be carried out based on some heuristic strategy like hill climbing, simulated annealing, evolutionary algorithms.
Data set reduction, prototype building

- Advantage of the nearest neighbour approach:
  No time for training is needed – at least when no feature weight adaptation is carried out or the number of nearest neighbours is fixed in advance.
Data set reduction, prototype building

- Advantage of the nearest neighbour approach:
  No time for training is needed – at least when no feature weight adaptation is carried out or the number of nearest neighbours is fixed in advance.

- Disadvantage:
  Calculation of the predicted class or value can take long when the data set is large.
Advantage of the nearest neighbour approach:
No time for training is needed – at least when no feature weight adaptation is carried out or the number of nearest neighbours is fixed in advance.

Disadvantage:
Calculation of the predicted class or value can take long when the data set is large.

Possible solutions:
Data set reduction, prototype building

- Advantage of the nearest neighbour approach:
  No time for training is needed – at least when no feature weight adaptation is carried out or the number of nearest neighbours is fixed in advance.

- Disadvantage:
  Calculation of the predicted class or value can take long when the data set is large.

Possible solutions:
- Finding a smaller subset of the training set for the nearest neighbour predictor.
Data set reduction, prototype building

- Advantage of the nearest neighbour approach:
  No time for training is needed – at least when no feature weight adaptation is carried out or the number of nearest neighbours is fixed in advance.

- Disadvantage:
  Calculation of the predicted class or value can take long when the data set is large.

Possible solutions:
- Finding a smaller subset of the training set for the nearest neighbour predictor.
- Building prototypes by merging (close) instances, for instance by averaging.
Data set reduction, prototype building

- Advantage of the nearest neighbour approach:
  No time for training is needed – at least when no feature weight adaptation is carried out or the number of nearest neighbours is fixed in advance.

- Disadvantage:
  Calculation of the predicted class or value can take long when the data set is large.

Possible solutions:
- Finding a smaller subset of the training set for the nearest neighbour predictor.
- Building prototypes by merging (close) instances, for instance by averaging.

Can be carried out based on cross-validation and using heuristic optimisation strategies.
Choice of parameter $k$

Linear classification problem (with some noise):
Choice of parameter k

1 nearest neighbour:
Choice of parameter k

2 nearest neighbour:
Choice of parameter $k$

5 nearest neighbour:
Choice of parameter $k$

50 nearest neighbour:
Choice of parameter $k$

470 nearest neighbour:
Choice of parameter $k$

480 nearest neighbour:
Choice of parameter $k$

500 nearest neighbour:
Choice of Parameter $k$

- $k = 1$ yields $y = \text{piecewise constant labeling}$
- "too small" $k$: very sensitive to outliers
- "too large" $k$: many objects from other clusters (classes) in the decision set
- $k = N$ predicts $y = \text{globally constant (majority) label}$

The selection of $k$ depends from various input "parameters":
- the size $n$ of the data set
- the quality of the data
  - ...

![Diagram showing decision sets for different values of $k$](image-url)
Choice of Parameter $k$: cont.

Simple classifier, $k = 1, 2, \ldots$
Choice of Parameter $k$: cont.

Simple data
Choice of Parameter $k$: cont.

Simple classifier, $k = 1$. Voronoi Tessellation of input space.
Choice of Parameter $k$: cont.

Simple classifier, $k = 1$. ...and classification.
Choice of Parameter k: cont.

Simple classifier, \( k = 1 \)

Concept, Images, and Analysis from Peter Flach.
Choice of Parameter $k$: cont.

Simple classifier, $k = 2$
Choice of Parameter $k$: cont.

Simple classifier, $k = 2$
Choice of Parameter $k$: cont.

Simple classifier, $k = 3$
Choice of Parameter \( k \)

- \( k = 1 \): highly localized classifier, perfectly fits separable training data
- \( k > 1 \):
  - the instance space partition refines
  - more segments are labelled with the same local models
Choice of Parameter $k$ - Cross Validation

$k$ is mostly determined manually or heuristically.

One heuristic: Cross Validation

1. Select a cross validation method (e.g. $q$-fold cross validation with $D = D_1 \cup \ldots \cup D_q$)
2. Select a range for $k$ (e.g. $1 < k \leq k_{\text{max}}$)
3. Select an evaluation measure (e.g. $E(k) = \sum_{i=1}^{q} \sum_{x \in D_i} p(x \text{ is correctly classified}\mid D \setminus D_i)$)
4. Use $k$ which results in minimal $k_{\text{best}} = \arg\min (E(k))$
Choice of Parameter k - Cross Validation

k is mostly determined manual or heuristic

One heuristic : Cross Validation

1. Select a cross validation method (e.g. q-fold cross validation with \( D = D_1 \cup \ldots \cup D_q \))

2. Select a range for k (e.g. \( 1 < k \leq k_{\text{max}} \))

3. Select an evaluation measure (e.g. 
\[
E(k) = \sum_{i=1}^{q} \sum_{x \in D_i} p(x \text{ is correct classified} | D \setminus D_i)
\]

4. Use \( k \) which results in minimal \( k_{\text{best}} = \arg \min (E(k)) \)

- Can we do this in KNIME?...
Instance Based Classifier: Remembers all training cases

Sensitive to neighborhood:

- Distance Function
- Neighborhood Weighting
- Prediction (Aggregation) Function
Food for Thought: 1-NN Classifier

- Bias of the Learning Algorithm?
- Model Bias?
- Hypothesis Space?
Food for Thought: 1-NN Classifier

- Bias of the Learning Algorithm?
  - No variations in search: simple store all examples
- Model Bias?
- Hypothesis Space?
Food for Thought: 1-NN Classifier

- Bias of the Learning Algorithm?
  - No variations in search: simple store all examples
- Model Bias?
  - Classification via Nearest Neighbor
- Hypothesis Space?
Food for Thought: 1-NN Classifier

- Bias of the Learning Algorithm?
  - No variations in search: simple store all examples
- Model Bias?
  - Classification via Nearest Neighbor
- Hypothesis Space?
  - One hypothesis only: Voronoi partitioning of space
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.
- Lazy Algorithms:

Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.
- Lazy Algorithms:
  - do nothing during training (just store examples)
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.
- Lazy Algorithms:
  - do nothing during training (just store examples)
  - Generate new hypothesis for each query ("class A!" in case of kNN!)
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.

Lazy Algorithms:
- do nothing during training (just store examples)
- Generate new hypothesis for each query ("class A!" in case of kNN!)

Eager Algorithms:
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.

Lazy Algorithms:
- do nothing during training (just store examples)
- Generate new hypothesis for each query ("class A!" in case of kNN!)

Eager Algorithms:
- do as much as possible during training (ideally: extract the one relevant rule!)
Again: Lazy vs. Eager Learners

- kNN learns a local model at query time
- Previous algorithms (k-means, ID3, ...) learn a global model before query time.
- Lazy Algorithms:
  - do nothing during training (just store examples)
  - Generate new hypothesis for each query ("class A!" in case of kNN!)
- Eager Algorithms:
  - do as much as possible during training (ideally: extract the one relevant rule!)
  - Generate one global hypothesis (or a set, see Candidate-Elimination) once.
Other Types of Lazy Learners
Lazy Decision Trees

- Can we use a Decision Tree Algorithm in a lazy mode?
Lazy Decision Trees

- Can we use a Decision Tree Algorithm in a lazy mode?
  - Sure: only create branch that contains test case.
Lazy Decision Trees

- Can we use a Decision Tree Algorithm in a lazy mode?
- Sure: only create branch that contains test case.
- Better: do beam search instead of greedy “branch” building!
Lazy Decision Trees

- Can we use a Decision Tree Algorithm in a lazy mode?
- Sure: only create branch that contains test case.
- Better: do beam search instead of greedy “branch” building!
- Works for essentially all model building algorithms (but makes sense for “partitioning”-style algorithms only...
Lazy(?) Neural Networks

Specht introduced Probabilistic Neural Networks in 1990.
Specht introduced Probabilistic Neural Networks in 1990.

- Special type of Neural Networks, optimized for classification tasks.
Specht introduced Probabilistic Neural Networks in 1990.

- Special type of Neural Networks, optimized for classification tasks.
- One Neuron per training instance
Lazy(?) Neural Networks

Specht introduced Probabilistic Neural Networks in 1990.

- Special type of Neural Networks, optimized for classification tasks.
- One Neuron per training instance
- Actually long known as “Parzen Window Classifier” in statistics.
Lazy(?) Neural Networks

Specht introduced Probabilistic Neural Networks in 1990.

- Special type of Neural Networks, optimized for classification tasks.
- One Neuron per training instance
- Actually long known as “Parzen Window Classifier” in statistics.

Parzen Window

The Parzen windows method is a non-parametric procedure that synthesizes an estimate of a probability density function (pdf) by superposition of a number of windows, replicas of a function (often the Gaussian).
Probabilistic Neural Networks

- Propagation function (input to hidden layer):

\[ d(\vec{w}, \vec{x}) = \sqrt{\sum_{i=1}^{m} (w_i - x_i)^2} \]

- Activation function (output of hidden layer):

\[ o_{\text{hidden}}^j = f(d, \sigma) = e^{-\frac{d^2}{2\sigma^2}} \]

- Propagation function (hidden to output layer):

\[ a_{k\text{out}}(\vec{x}) = \sum_{j=0}^{h} w_{j,k}^\text{out} \cdot o_{j\text{hidden}} \]

- Activation function (output layer):

\[ o_{k\text{out}} = \frac{a_{k\text{out}}}{\sum_{l=1}^{c} a_{l\text{out}}} \]

PNN vs. RBF:
- Output Layer not fully connected
- Normalized Outputs
Probabilistic Neural Networks

\[ o_{k}^{\text{out}} = p(\text{Klasse } k \mid \bar{x}) \]

\[ w_{j,k}^{\text{out}} = p(\text{Klasse } k \mid \text{Cluster } j) \]

\[ o_{j}^{\text{hidden}} = p(\text{Cluster } j \mid \bar{x}) \]

\[ d(\vec{w}_{j}^{\text{hidden}}, \bar{x}) \]
Probabilistic Neural Networks are powerful predictors (but $\sigma$-adjustment problematic)
Probabilistic Neural Networks are powerful predictors (but \( \sigma \)-adjustment problematic)

Efficient (and eager!) training algorithms exist that introduce more general neurons, covering more than just one training instance.
Probabilistic Neural Networks are powerful predictors (but $\sigma$-adjustment problematic)

Efficient (and eager!) training algorithms exist that introduce more general neurons, covering more than just one training instance.

Usual problems of distance based classifiers apply.