K-Nearest Neighbour (Continued)

Dr. Xiaowei Huang
https://cgi.csc.liv.ac.uk/~xiaowei/
Up to now,

• Recap basic knowledge
• Decision tree learning
• k-NN classification
  • What is k-nearest-neighbor classification
  • How can we determine similarity/distance
  • Standardizing numeric features (leave this to you)
Today’s Topics

• Definition

• **Speeding up k-NN**
  • edited nearest neighbour
  • k-d trees for nearest neighbour identification

• Variants of k-NN
  • K-NN regression
  • Distance-weighted nearest neighbor
  • Locally weighted regression to handle irrelevant features

• Discussions
  • Strengths and limitation of instance-based learning
  • Inductive bias
Definition
k-nearest-neighbor classification

• classification task
  • **given**: an instance $x^{(q)}$ to classify
  • find the $k$ training-set instances $(x^{(1)}, y^{(1)})$... $(x^{(k)}, y^{(k)})$ that are the **most similar** to $x^{(q)}$
  • return the class value

$$
\hat{y} \leftarrow \text{argmax} \sum_{v \in \text{values}(Y)}^{k} \delta(v, y^{(i)})
$$

$$
\delta(a, b) = \begin{cases} 
1 & \text{if } a = b \\
0 & \text{otherwise}
\end{cases}
$$

• (i.e. return the class that have the most number of instances in the $k$ training instances)
How can we determine similarity/distance

• suppose all features are discrete
  • Hamming distance (or $L^0$ norm): count the number of features for which two instances differ

• Example: $X = \text{(Weekday, Happy?, Weather)}$  $Y = \text{AttendLecture?}$
  • $D$: in the table
  • New instance: $<\text{Friday, No, Rain}>$
  • Distances = $\{2, 3, 1, 2\}$
  • For 1-nn, which instances should be selected?
  • For 2-nn, which instances should be selected?
  • For 3-nn, which instances should be selected?

<table>
<thead>
<tr>
<th></th>
<th>$v1$</th>
<th>$v2$</th>
<th>$v3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed</td>
<td>Yes</td>
<td>Rain</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Wed</td>
<td>Yes</td>
<td>Sunny</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Thu</td>
<td>No</td>
<td>Rain</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Fri</td>
<td>Yes</td>
<td>Sunny</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

New datum

Fri   No   Rain
How can we determine similarity/distance

• Example: \( X = (\text{Weekday}, \text{Happy?}, \text{Weather}) \) \( Y = \text{AttendLecture?} \)
  • New instance: \(<\text{Friday}, \text{No}, \text{Rain}>\>
  • For 3-\( \text{nn} \), selected instances: \{\(<\text{Wed}, \text{Yes}, \text{Rain}>\), \text{No}\}, \(<\text{Thu}, \text{No}, \text{Rain}>\), \text{Yes}\), \(<\text{Fri}, \text{Yes}, \text{Sunny}>\), \text{No}\}\}

• Classification:

\[
\hat{y} \leftarrow \underset{v \in \text{values}(Y)}{\text{argmax}} \sum_{i=1}^{k} \delta(v, y^{(i)})
\]

• \( v = \text{Yes} \). \[
\sum_{i=1}^{k} \delta(v, y^{(i)}) = 0 + 1 + 0 = 1
\]

• \( v = \text{No} \). \[
\sum_{i=1}^{k} \delta(v, y^{(i)}) = 1 + 0 + 1 = 2
\]

So, which class this new instance should be in?
How can we determine similarity/distance

- suppose all features are continuous
  - Euclidean distance:
    \[ d(x^{(i)}, x^{(j)}) = \sqrt{\sum_{f} (x_f^{(i)} - x_f^{(j)})^2} \]
  - Manhattan distance:
    \[ d(x^{(i)}, x^{(j)}) = \sum_{f} |x_f^{(i)} - x_f^{(j)}| \]

where \( x_f^{(i)} \) represents the \( f \)-th feature of \( x^{(i)} \)

Recall the difference and similarity with L^p norm
How can we determine similarity/distance

• Example: $X = (\text{Height}, \text{Weight}, \text{RunningSpeed})$  $Y = \text{SoccerPlayer}$?
  • $D$: in the table
  • New instance: $<185, 91, 13.0>$
  • Suppose that Euclidean distance is used.
  • Is this person a soccer player?

<table>
<thead>
<tr>
<th>$v1$</th>
<th>$v2$</th>
<th>$v3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>182</td>
<td>87</td>
<td>11.3</td>
<td>No</td>
</tr>
<tr>
<td>189</td>
<td>92</td>
<td>12.3</td>
<td>Yes</td>
</tr>
<tr>
<td>178</td>
<td>79</td>
<td>10.6</td>
<td>Yes</td>
</tr>
<tr>
<td>183</td>
<td>90</td>
<td>12.7</td>
<td>No</td>
</tr>
</tbody>
</table>

New datum: $<185, 91, 13.0>$
How can we determine similarity/distance

• if we have a mix of discrete/continuous features:

\[ d(x^{(i)}, x^{(j)}) = \sum_f \left\{ \begin{array}{ll} |x_f^{(i)} - x_f^{(j)}| & \text{if } f \text{ is continuous} \\ 1 - \delta(x_f^{(i)}, x_f^{(i)}) & \text{if } f \text{ is discrete} \end{array} \right. \]

• typically want to apply to continuous features some type of normalization (values range 0 to 1) or standardization (values distributed according to standard normal)

• many other possible distance functions we could use ...
Standardizing numeric features

• given the training set D, determine the mean and stddev for feature $x_i$

$$
\mu_i = \frac{1}{|D|} \sum_{d=1}^{|D|} x_i^{(d)} \quad \sigma_i = \sqrt{\frac{1}{|D|} \sum_{d=1}^{|D|} (x_i^{(d)} - \mu_i)^2}
$$

• standardize each value of feature $x_i$ as follows

$$
\hat{x}_i^{(d)} = \frac{x_i^{(d)} - \mu_i}{\sigma_i}
$$

• do the same for test instances, using the same $\mu$ and $\sigma$ derived from the training data
Speeding up k-NN
Issues

• Choosing k
  • Increasing k reduces variance, increases bias

• For high-dimensional space, problem that the nearest neighbor may not be very close at all!

• Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.
Nearest neighbour problem

• Given sample $S = ((x_1,y_1),...,(x_m,y_m))$ and a test point $x$,
• it is to find the nearest k neighbours of $x$.

• Note: for the algorithms, dimensionality $N$, i.e., number of features, is crucial.
Efficient Indexing: N=2

- Algorithm
  - compute Voronoi diagram in $O(m \log m)$
    - See algorithm in https://en.wikipedia.org/wiki/Fortune's_algorithm
  - use point location data structure to determine nearest neighbours
  - complexity: $O(m)$ space, $O(\log m)$ time.
Efficient Indexing: N>2

• Voronoi diagram: size in $O(m^{N/2})$
• Linear algorithm (no pre-processing):
  • compute distance $||x - x_i||$ for all $i \in [1, m]$.
  • complexity of distance computation: $\Omega(N m)$.
  • no additional space needed.

k-NN is a “lazy” learning algorithm – does virtually nothing at training time but classification/prediction time can be costly when the training set is large
Efficient Indexing: N>2

- two general strategies for alleviating this weakness
  - don’t retain every training instance (edited nearest neighbor)
  - pre-processing. Use a smart data structure to look up nearest neighbors (e.g. a k-d tree)
Edited instance-based learning

• select a subset of the instances that still provide accurate classifications

• *incremental deletion*
  
  start with all training instances in memory
  for each training instance \((x^{(i)}, y^{(i)})\)
  if other training instances provide correct classification for \((x^{(i)}, y^{(i)})\)
  delete it from the memory

• *incremental growth*
  
  start with an empty memory
  for each training instance \((x^{(i)}, y^{(i)})\)
  if other training instances in memory don’t correctly classify \((x^{(i)}, y^{(i)})\)
  add it to the memory

Q1: Does ordering matter?

Q2: If following the optimal ordering, do the two approaches produce the same subset of instances?
**k-d trees**

- A *k-d tree* is similar to a decision tree except that each internal node
  - Stores one instance
  - Splits on the median value of the feature having the highest variance
Construction of k-d tree

median value of the feature having the highest variance?
-- point f, $x_1 = 6$
Construction of k-d tree

median value of the feature having the highest variance?
-- point f, $x_1 = 6$
-- point c, $x_2 = 10$ and point h, $x_2 = 5$
Construction of k-d tree

There can be other methods of constructing k-d trees, see e.g., https://en.wikipedia.org/wiki/K-d_tree#Nearest_neighbour_search
Finding nearest neighbors with a k-d tree

- use branch-and-bound search
- priority queue stores
  - nodes considered
  - lower bound on their distance to query instance
- lower bound given by distance using a single feature
- average case: $O(\log_2 m)$
- worst case: $O(m)$ where $m$ is the size of the training-set
Intuitively, for a pair \((\text{node}, \text{value})\), \text{value} represents the smallest guaranteed distance, i.e., greatest lower bound up to now, from the instance \(x^{(q)}\) to the set of instances over which \text{node} is the selected one to split.

For example, the set of instances where \text{root} is the selected one to split over is the whole training set. (root,0) means that at the beginning, the guaranteed smallest distance to the training set is 0.
k-d tree example (Manhattan distance)
k-d tree example (Manhattan distance)

Given query \( x^{(q)} = (2, 3) \)

```
(node, bound) = PQ.pop();
if (bound ≥ best_dist)
    return best_node.instance
dist = distance(x^{(q)}, node.instance)
if (dist < best_dist)
    best_dist = dist
    best_node = node
if (q[node.feature] − node.threshold > 0)
    PQ.push(node.left, x^{(q)}[node.feature])
    PQ.push(node.right, 0)
else
    PQ.push(node.left, 0)
    PQ.push(node.right, node.threshold)
```
k-d tree example (Manhattan distance)

Given query: $x(q) = (2, 3)$

```
(node, bound) = PQ.pop();
if (bound ≥ best_dist)
    return best_node.instance

dist = distance($x(q)$, node.instance)
if (dist < best_dist)
    best_dist = dist
    best_node = node

if (q[node.feature] − node.threshold > 0)
    PQ.push(node.left, $x(q)[node.feature]$
    PQ.push(node.right, 0)
else
    PQ.push(node.left, 0)
    PQ.push(node.right, node.threshold)
```

---

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>∞</td>
<td></td>
<td></td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>
k-d tree example (Manhattan distance)

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(f, 0)</td>
<td>PQ.push(node.left, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0)</td>
</tr>
</tbody>
</table>
| | | | }
k-d tree example (Manhattan distance)

Given query: \(x(q) = (2, 3)\)

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop f</td>
<td>(\infty)</td>
<td></td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0) (h, 4)</td>
</tr>
</tbody>
</table>

```
(node, bound) = PQ.pop();
if (bound ≥ best_dist)
    return best_node.instance
dist = distance(x(q), node.instance)
if (dist < best_dist)
    best_dist = dist
    best_node = node
if (q[node.feature] − node.threshold > 0)
PQ.push(node.left, x(q)[node.feature])
PQ.push(node.right, 0)
else
    PQ.push(node.left, 0)
```
k-d tree example (Manhattan distance)

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0) (h, 4)</td>
</tr>
<tr>
<td>10.0</td>
<td>4.0</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

pop f

pop c
k-d tree example (Manhattan distance)

- **Given query:** \( x(q) = (2, 3) \)

- **Tree Structure:**
  - **Root:** \( x_1 > 6 \)
  - **Left Child:** \( x_2 > 10 \)
  - **Right Child:** \( x_2 > 5 \)

- **Node Operations:**
  - (node, bound) = PQ.pop();
  - if (bound ≥ best_dist);
  - return best_node.instance
  - dist = distance(\( x(q) \), node.instance)
  - if (dist < best_dist);
  - best_dist = dist
  - best_node = node
  - if (\( q[\text{node.feature}] - \text{node.threshold} > 0 \));
    - PQ.push(node.left, \( x(q)[\text{node.feature}] \))
  - else;
    - PQ.push(node.right, node.threshold)

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>pop f</td>
<td>( \infty )</td>
<td>f</td>
<td>(f, 0)</td>
</tr>
<tr>
<td>pop c</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0) (h, 4)</td>
</tr>
<tr>
<td>10.0</td>
<td>4.0</td>
<td>f</td>
<td>(e, 0) (h, 4) (b, 7)</td>
</tr>
</tbody>
</table>
k-d tree example (Manhattan distance)

Given query $x^{(q)} = (2, 3)$

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td></td>
<td></td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0) (h, 4)</td>
</tr>
<tr>
<td>10.0</td>
<td>4.0</td>
<td>f</td>
<td>(e, 0) (h, 4)</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>e</td>
<td></td>
</tr>
</tbody>
</table>
k-d tree example (Manhattan distance)

Given query: \( x(q) = (2, 3) \)

<table>
<thead>
<tr>
<th>distance</th>
<th>best distance</th>
<th>best node</th>
<th>priority queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( f )</td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>( f )</td>
<td>(c, 0) (h, 4)</td>
</tr>
<tr>
<td>10.0</td>
<td>4.0</td>
<td>( f' )</td>
<td>(e, 0) (h, 4) (b, 7)</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>( e )</td>
<td>(d, 1) (h, 4) (b, 7)</td>
</tr>
</tbody>
</table>

(node, bound) = PQ.pop();
if (bound ≥ best_dist)
    return best_node.instance
dist = distance(x(q), node.instance)
if (dist < best_dist)
    best_dist = dist
    best_node = node
if (q[node.feature] − node.threshold > 0)
    PQ.push(node.left, x(q[node.feature])
else
    PQ.push(node.right, node.threshold)
**k-d tree example (Manhattan distance)**

**Diagram:**
- Given query: \( x(q) = (2, 3) \)
- Nodes in the tree: \( a, b, c, d, e, f, g, h, i, j \)
- Distances:
  - \( d \rightarrow f: 4.0 \)
  - \( c \rightarrow f: 10.0 \)
  - \( e \rightarrow e: 1.0 \)

**Algorithm:**
- Initialize priority queue (PQ) with initial node
- For each node \( n \):
  - if \( \text{dist} \leq \text{best_dist} \):
    - update best_distance and best_node
    - if \( \text{dist} < \text{best_dist} \):
      - return best_node
- If no match found, return any node

**Priority Queue Operations:**
- \( \text{PQ.pop()} \)
- \( \text{PQ.push(node, threshold)} \)

**Table:**
<table>
<thead>
<tr>
<th>Distance</th>
<th>Best Distance</th>
<th>Best Node</th>
<th>Priority Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \infty )</td>
<td>( \infty )</td>
<td></td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0) (h, 4)</td>
</tr>
<tr>
<td>10.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0) (h, 4) (b, 7)</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>e</td>
<td>(d, 1) (h, 4) (b, 7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Extended Materials: Voronoi Diagram Generation

- [https://courses.cs.washington.edu/courses/cse326/00wi/projects/voronoi.html](https://courses.cs.washington.edu/courses/cse326/00wi/projects/voronoi.html)
Variants of k-NN
k-nearest-neighbor *regression*

- learning stage
  - given a training set \((x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})\), do nothing
    - (it’s sometimes called a *lazy learner*)

- classification stage
  - **given**: an instance \(x^{(q)}\) to classify
  - find the \(k\) training-set instances \((x^{(1)}, y^{(1)}) \ldots (x^{(k)}, y^{(k)})\) that are most similar to \(x^{(q)}\)
  - return the value

\[
\hat{y} \leftarrow \frac{1}{k} \sum_{i=1}^{k} y^{(i)}
\]

Average over neighbours’ values
Distance-weighted nearest neighbor

• We can have instances contribute to a prediction according to their distance from $x^{(q)}$

• classification:

$$\hat{y} \leftarrow \arg\max_{v \in \text{values}(Y)} \sum_{i=1}^{k} w_i \delta(v, y^{(i)})$$

$$w_i = \frac{1}{d(x^{(q)}, x^{(i)})^2}$$

Intuition: instances closer to the current one is more important.

• regression:

$$\hat{y} \leftarrow \frac{\sum_{i=1}^{k} w_i y^{(i)}}{\sum_{i=1}^{k} w_i}$$

reciprocal of the distance
Irrelevant features in instance-based learning

here’s a case in which there
is one relevant feature \( x_1 \) and a 1-NN
rule classifies each instance correctly

consider the effect of an
irrelevant feature \( x_2 \) on distances
and nearest neighbors

Can you find a point \((a,b)\) which is red,
if classified only according to feature
\( x_1 \), but is green, if classified according
to both features?
Locally weighted regression

• one way around this limitation is to weight features differently

• *locally weighted regression* is one nearest-neighbor variant that does this

• prediction task
  • **given**: an instance $x^{(q)}$ to make a prediction for
  • find the k training-set instances $(x^{(1)}, y^{(1)})$ ... $(x^{(k)}, y^{(k)})$ that are most similar to $x^{(q)}$
  • return the value $f(x^{(q)})$

What’s function f?
Locally weighted regression

- Determining function $f$
  - Assume that $f$ is a linear function over the features, i.e.,
    \[
    f(x^{(i)}) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_n x_n^{(i)}
    \]
  - find the weights $w_i$ for each $x^{(q)}$ by
    \[
    \arg \min_{w_0, w_1, \ldots, w_n} \sum_{i=1}^{k} (f(x^{(i)}) - y^{(i)})^2
    \]
  - After obtaining weights, for $x^{(q)}$, we have
    \[
    f(x^{(q)}) = w_0 + w_1 x_1^{(q)} + w_2 x_2^{(q)} + \ldots + w_n x_n^{(q)}
    \]
  can do this using gradient descent (to be covered soon)
Discussions
Strengths of instance-based learning

• simple to implement
• “training” is very efficient
• adapts well to on-line learning
• robust to noisy training data (when k > 1)
• often works well in practice
Limitations of instance-based learning

- sensitive to range of feature values
- sensitive to irrelevant and correlated features, although ...
  - there are variants (such as locally weighted regression) that learn weights for different features
- classification/prediction can be inefficient, although ...
  - edited methods and k-d trees can help alleviate this weakness
- doesn’t provide much insight into problem domain because there is no explicit model
Inductive bias

• *inductive bias* is the set of assumptions a learner uses to be able to predict $y_i$ for a previously unseen instance $x_i$

• two components
  • *hypothesis space bias*: determines the models that can be represented
  • *preference bias*: specifies a preference ordering within the space of models

• in order to *generalize* (i.e. make predictions for previously unseen instances) a learning algorithm must have an inductive bias
Consider the inductive bias of DT and k-NN learners

<table>
<thead>
<tr>
<th>learner</th>
<th>hypothesis space bias</th>
<th>preference bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID3 decision tree</td>
<td>trees with single-feature, axis-parallel splits</td>
<td>small trees identified by greedy search</td>
</tr>
<tr>
<td>k-NN</td>
<td>Voronoi decomposition determined by nearest neighbors</td>
<td>instances in neighborhood belong to same class</td>
</tr>
</tbody>
</table>