Instance-Based Learning

CS 760@UW-Madison
Goals for the lecture

you should understand the following concepts

- $k$-NN classification
- $k$-NN regression
- edited nearest neighbor
- $k$-d trees for nearest neighbor identification (optional)
- locally weighted regression
- inductive bias (hypothesis space bias, preference bias)
Nearest-neighbor classification

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 training-set instance \(x^{(i)}\) that is most similar to \(x^{(q)}\)
• return the class value \(y^{(i)}\)
The decision regions

Voronoi diagram: each polyhedron indicates the region of feature space that is in the nearest neighborhood of each training instance
\( k \)-nearest-neighbor classification

classification task

- **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 class value

\[
\hat{y} \leftarrow \arg \max_{v \in \text{values}(Y)} \sum_{i=1}^{k} \delta(v, y^{(i)}) \\
\delta(a, b) = \begin{cases} 
1 & \text{if } a = b \\
0 & \text{otherwise}
\end{cases}
\]

(i.e. return the class that have the most instances)
How can we determine distance

suppose all features are discrete
• Hamming distance: count the number of features for which two instances differ

suppose all features are continuous
• Euclidean distance:

\[ d(x^{(i)}, x^{(j)}) = \sqrt{\sum_f \left( x_f^{(i)} - x_f^{(j)} \right)^2} \]

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

• Manhattan distance:

\[ d(x^{(i)}, x^{(j)}) = \sum_f |x_f^{(i)} - x_f^{(j)}| \]
How can we determine distance

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

\[ d(x^{(i)}, x^{(j)}) = \sum_{f} \begin{cases} 
| x_f^{(i)} - x_f^{(j)} | & \text{if } f \text{ is continuous} \\
1 - \delta (x_f^{(i)}, x_f^{(j)}) & \text{if } f \text{ is discrete}
\end{cases} \]

- 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 $x_i$

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

- 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 $\mu_i$ and $\sigma_i$ derived from the training data
Variants
**k-nearest-neighbor regression**

**learning stage**
- given a training set \((x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\), do nothing

**prediction stage**
- **given**: an instance \(x^{(q)}\) to make a prediction for
- 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

\[
\hat{y} \leftarrow \frac{1}{k} \sum_{i=1}^{k} y^{(i)}
\]
Distance-weighted nearest neighbor

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

classification:

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

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

regression:

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

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.
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 \((x^{(1)}, y^{(1)}), \ldots, (x^{(k)}, y^{(k)})\) that are most similar to \( x^{(q)} \)
- return the value

\[
f(x^{(q)}) = w_0 + w_1 x_1^{(q)} + w_2 x_2^{(q)} + \ldots + w_n x_n^{(q)}
\]
Locally weighted regression

drop prediction/learning task
• find the weights $w_i$ for each $x^{(q)}$ by minimizing

$$E(x^{(q)}) = \sum_{i=1}^{k} (f(x^{(i)}) - y^{(i)})^2$$

• this is done at prediction time, specifically for $x^{(q)}$
• can do this using gradient descent (to be covered soon)
Speeding up $k$-NN

- $k$-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

- two general strategies for alleviating this weakness
  - don’t retain every training instance (edited nearest neighbor)
  - 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
Strength and Limitations
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
  • later we’ll talk about feature selection methods

• 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 $\kappa$-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>$\kappa$-NN</td>
<td>Voronoi decomposition determined by nearest neighbors</td>
<td>instances in neighborhood belong to same class</td>
</tr>
</tbody>
</table>
k-d Tree: Data Structure for Finding Nearest Neighbors
**$k$-$d$ 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
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
Finding nearest neighbors in a k-d tree

NearestNeighbor(instance $x^{(q)}$)

PQ = { } // minimizing priority queue
best_dist = $\infty$ // smallest distance seen so far
PQ.push(root, 0)
while PQ is not empty
    (node, bound) = PQ.pop();
    if (bound ≥ best_dist)
        return best_node.instance // nearest neighbor found
    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, $x^{(q)}[node.feature] – node.threshold$)
        PQ.push(node.right, 0)
    else
        PQ.push(node.left, 0)
        PQ.push(node.right, node.threshold - $x^{(q)}[node.feature]$)
return best_node.instance
k-d tree example (Manhattan distance)

given query
\(x^{(q)} = (2, 3)\)
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>
</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></td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td></td>
</tr>
</tbody>
</table>

pop f
### k-d tree example (Manhattan distance)

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

---

**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></td>
<td></td>
<td>(f, 0)</td>
</tr>
<tr>
<td>4.0</td>
<td>4.0</td>
<td>f</td>
<td>(c, 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Pop f

---

Diagram showing the k-d tree with nodes and distances.
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, 0)</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>

pop f
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, 0)</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></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></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>
</tbody>
</table>

*pop f*

*pop c*
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></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>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>

**Diagram:**
- **q** is the given query point.
- The k-d tree structure is shown with nodes and distances.
- The priority queue contains nodes according to their distances.

**Steps:**
1. **pop f**
2. **pop c**
3. **pop e**
k-d tree example (Manhattan distance)

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

distance
\begin{align*}
\text{distance} & \quad \text{best distance} & \text{best node} & \text{priority queue} \\
\infty & \quad & & (f, 0) \\
4.0 & \quad 4.0 & f & (c, 0) (h, 4) \\
10.0 & \quad 4.0 & f & (e, 0) (h, 4) (b, 7) \\
1.0 & \quad 1.0 & e & (d, 1) (h, 4) (b, 7) \\
\end{align*}

\text{pop f}
\text{pop c}
\text{pop e}
\text{pop d} \quad \text{return } e
THANK YOU

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, and Pedro Domingos.