Nearest neighbor methods
Lecture 11

David Sontag
New York University

Slides adapted from Vibhav Gogate, Carlos Guestrin, Mehryar Mohri, & Luke Zettlemoyer
Nearest Neighbor Algorithm

• Learning Algorithm:
  – Store training examples

• Prediction Algorithm:
  – To classify a new example \( \mathbf{x} \) by finding the training example \((\mathbf{x}^i, y^i)\) that is nearest to \( \mathbf{x} \)
  – Guess the class \( y = y^i \)
K-Nearest Neighbor Methods

• To classify a new input vector \( x \), examine the \( k \)-closest training data points to \( x \) and assign the object to the most frequently occurring class.

Common values for \( k \): 3, 5
Decision Boundaries

- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.

\[ 1-\text{NN Decision Surface} \]

- The more examples that are stored, the more complex the decision boundaries can become.
Example results for k-NN

[Figures from Hastie and Tibshirani, Chapter 13]
Nearest Neighbor

When to Consider
- Instance map to points in $R^n$
- Less than 20 attributes per instance
- Lots of training data

Advantages
- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages
- Slow at query time
- Easily fooled by irrelevant attributes
Issues

• Distance measure
  – Most common: Euclidean

• 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.
Distance

• Notation: object with $p$ features

$$x^i = (x^i_1, x^i_2, \ldots, x^i_p)$$

• Most common distance metric is *Euclidean* distance:

$$d_E(x^i, x^j) = \left( \sum_{k=1}^{p} (x^i_k - x^j_k)^2 \right)^{1/2}$$

• ED makes sense when different features are commensurate; each is variable measured in the same units.

• If the features are different, say length and weight, it is not clear.
Normalization of features

Can divide features by them by the standard deviation, making them all equally important.

The estimate for the standard deviation of feature \( k \):

\[
\hat{\sigma}_k = \left( \frac{1}{n} \sum_{i=1}^{n} \left( x_k^i - \bar{x}_k \right)^2 \right)^{\frac{1}{2}}
\]

where \( \bar{x}_k \) is the sample mean:

\[
\bar{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_k^i
\]
Weighted Euclidean distance

Finally, if we have some idea of the relative importance of each variable, we can weight them:

\[ d_{WE}(i, j) = \left( \sum_{k=1}^{p} w_k (x^i_k - x^j_k)^2 \right)^{\frac{1}{2}} \]
$k$-NN and irrelevant features
$k$-NN and irrelevant features
Nearest neighbor problem

Problem: given sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$, find the nearest neighbor of test point $x$.

- general problem extensively studied in computer science.
- exact vs. approximate algorithms.
- dimensionality $N$ crucial.
- better algorithms for small intrinsic dimension (e.g., limited doubling dimension).

[Slides from Mehryar Mohri]
Efficient Indexing: N=2

Algorithm:

- compute Voronoi diagram in $O(m \log m)$.
- **point location** data structure to determine NN.
- complexity: $O(m)$ space, $O(\log m)$ time.

[Slides from Mehtyar Mohri]
Efficient Indexing: N>2

- Voronoi diagram: size in $O(m^{\lceil N/2 \rceil})$.

- Linear algorithm (no pre-processing):
  - compute distance $\|x - x_i\|$ for all $i \in [1, m]$.
  - complexity of distance computation: $\Omega(Nm)$.
  - no additional space needed.

- Tree-based data structures: pre-processing.
  - often used in applications: $k$-d trees ($k$-dimensional trees).

[Slides from Mehryar Mohri]
Efficient Indexing for \( N>2 \): KD trees

- **Algorithm**: for each non-leaf node,
  - choose dimension (e.g., longest of hyperrectangle).
  - choose pivot (median).
  - split node according to (pivot, dimension).

→ balanced tree, binary space partitioning.

[Slides from Mehryar Mohri]
Efficient Indexing for N>2: KD trees

Algorithm:
- find region containing $x$ (starting from root node, move to child node based on node test).
- save region point $x_0$ as current best.
- move up tree and recursively search regions intersecting hypersphere $S(x, \|x - x_0\|)$.

[Slides from Mehryar Mohri]
Weighted \( k \)-NN

- Consider the following generalization of the \( k \)-NN algorithm (specialized to binary classification):

\[
\hat{y}(\bar{x}) \leftarrow \text{sign} \left( \sum_{i=1}^{k} y_i d(\bar{x}_i, \bar{x}) \right) \quad \text{with} \quad d(\bar{x}_i, \bar{x}) = \frac{1}{||\bar{x}_i - \bar{x}||_2^2} \quad \text{or} \quad d(\bar{x}_i, \bar{x}) = \exp \left(-\frac{||\bar{x}_i - \bar{x}||_2^2}{2\sigma^2} \right)
\]

- Weighs the \( i' \)th training point’s label by how far \( x_i \) is from \( \bar{x} \)
$k$-NN is similar to SVM with Gaussian kernel!

- Consider the following generalization of the $k$-NN algorithm (specialized to binary classification):

$$
\hat{y}(\bar{x}) \leftarrow \text{sign} \left( \sum_{i=1}^{N} y_i d(\bar{x}_i, \bar{x}) \right) \quad \text{with} \quad d(\bar{x}_i, \bar{x}) = \frac{1}{||\bar{x}_i - \bar{x}||^2} \quad \text{or...} \quad d(\bar{x}_i, \bar{x}) = \exp \left( -\frac{||\bar{x}_i - \bar{x}||^2}{2\sigma^2} \right)
$$

- Looks at all training points (i.e., $k=\text{N}$), but weighs the $i$'th training point’s label by how far $\mathbf{x}_i$ is from $\mathbf{x}$

- Now compare this to classification with SVM and a Gaussian kernel:

$$
\hat{y}(\bar{x}) \leftarrow \text{sign} \left( \sum_{i=1}^{N} \alpha_i y_i K(\bar{x}_i, \bar{x}) \right) \quad \text{with} \quad K(u, v) = \exp \left( -\frac{||u - v||^2}{2\sigma^2} \right) \quad 0 \leq \alpha_i \leq C
$$

- The discriminant functions are nearly identical! The SVM has parameters $\alpha_i$ that can be learned
KNN Advantages

• Easy to program
• No optimization or training required
• Classification accuracy can be very good; can outperform more complex models