**Overview**

- Nearest neighbour method
  - classification and regression
  - practical issues: k, distance, ties, missing values
  - optimality and assumptions
- Making kNN fast:
  - K-D trees
  - inverted indices
  - fingerprinting
- References: W&F sections 4.7 and 6.4

**Intuition for kNN**

- set of points \((x, y)\)
  - two classes
  - is the box red or blue
  - how did you do it
    - use Bayes rule?
    - a decision tree?
    - fit a hyperplane?
  - nearby points are red
  - use this as a basis for a learning algorithm

**Nearest neighbour classification**

- Use the intuition to classify a new point \(x^*\):
  - find the most similar training example \(x^*\)
  - predict its class \(y^*\)
- Voronoi tessellation
  - partitions space into regions
  - boundary: points at same distance from two different training examples
- Classification boundary
  - non-linear, reflects classes well
  - compare to NB, DT, logistic
  - impressive for simple method

**Nearest neighbour: outliers**

- Algorithm is sensitive to outliers
  - single mislabeled example dramatically changes boundary
- No confidence \(P(y|x)\)
- Insensitive to class prior
- Idea:
  - use more than one nearest neighbor to make decision
  - count class labels in \(k\) most similar training examples
    - many “triangles” will outweigh single “circle” outlier

**kNN classification algorithm**

- Given:
  - training examples \(\{x_i, y_i\}\)
  - \(x_i\): attribute-value representation of examples
  - \(y_i\): class label (ham, spam), digit \((0,1,\ldots,9)\) etc.
  - testing point \(x\) that we want to classify
- Algorithm:
  - compute distance \(D(x, x_i)\) to every training example \(x_i\)
  - select \(k\) closest instances \(x_1, \ldots, x_k\) and their labels \(y_1, \ldots, y_k\)
  - output the class \(y^*\) which is most frequent in \(y_1 \ldots y_k\)

**Example: handwritten digits**

- 16x16 bitmaps
- 8-bit grayscale
- Euclidean distance
  - over raw pixels
  - \(D(A,B) = \sqrt{\sum \sum (A_i - B_i)^2}\)
- Accuracy:
  - 7-NN = 95.2%
  - SVM = 95.8%
  - humans = 97.5%

**kNN regression algorithm**

- Given:
  - training examples \(\{x_i, y_i\}\)
  - \(x_i\): attribute-value representation of examples
  - \(y_i\): real-valued target (profit, rating on YouTube, etc.)
  - testing point \(x\) that we want to predict the target
- Algorithm:
  - compute distance \(D(x, x_i)\) to every training example \(x_i\)
  - select \(k\) closest instances \(x_1, \ldots, x_k\) and their labels \(y_1, \ldots, y_k\)
  - output the mean of \(y_1 \ldots y_k\):
    \[ \hat{y} = f(x) = \frac{1}{k} \sum_{i=1}^{k} y_i \]

**Example: kNN regression in 1-d**
Choosing the value of k

- Value of k has strong effect on kNN performance
  - large k → everything classified as the most probable class: PV(y)
  - small k → highly variable, unstable decision boundaries
  - small changes to training set → large changes in classification
  - affects “smoothness” of the boundary
- Selecting the value of k
  - set aside a portion of the training data (validation set)
  - vary k, observe training & validation error
  - pick k that gives best generalization performance

kNN: practical issues

- Resolving ties:
  - equal number of positive/negative neighbours
  - use odd k (doesn’t solve multi-class)
- Breaking ties:
  - random: flip a coin to decide positive / negative
  - prior: pick class with greater prior
  - nearest: use 1nn classifier to decide
- Missing values:
  - have to “fill in”, otherwise can’t compute distance
  - key concern: should distance be as little as possible
  - reasonable choice: average value across entire dataset

kNN, Parzen Windows and Kernels

- Parzen window: average of the training data
- Kernel function: smooths the training data
- Parzen window: average of the training data
- Kernel function: smooths the training data

Distance measures

- Key component of the kNN algorithm
  - defines which examples are similar & which aren’t
  - can have strong effect on performance
- Euclidean (numeric attributes): $D(x, x') = \sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2}$
  - symmetric, spherical, treats all dimensions equally
  - sensitive to extreme differences in single attribute
  - behaves like a “soft” logical OR
- Hamming (categorical attributes): $D(x, x') = \sum_{i} I(x_i \neq x'_i)$
  - number of attributes where $x, x'$ differ

Custom distance measures (BM25 for text)

kNN pros and cons

- Almost no assumptions about the data
  - smoothness: nearby regions of space → same class
  - assumptions implied by distance function (only locally)
- Non-parametric approach: “let the data speak for itself”
  - nothing to infer from the data, except it and possible DJ
- Easy to update in online setting: just add new item to training set
- Need to handle missing data: fill-in or create a special distance
- Sensitive to class-outliers (mislabeled training instances)
- Sensitive to lots of irrelevant attributes (affect distance)
- Computational expensive:
  - space: need to store all training examples
  - time: need to compute distance to all examples: O(nd)
  - n → number of training examples, d → cost of computing distance
  - require system & data parallel to avoid slow performance
  - naively testing, not taking time into account

Summary: kNN

- Key idea: nearby points → same class
  - important to select good distance function
- Can be used for classification and regression
- Simple, non-linear, asymptotically optimal
  - does not make assumptions about the data
  - “let the data speak for itself”
- Select k by optimizing error on held-out set
- Naive implementations slow for big datasets
  - use KD trees (low-d) or inverted lists (high-d)

Why is kNN slow?

- What you see
  - Training set:
    - $\{1.3, 3.2, 4.3, 5.3, 6.4, 7.2, 8.0, 7.0, 9.0\}$
  - Testing instance:
    - $\{7.4\}$
  - Nearest neighbors:
    - compare one-by-one to each training instance
  - n comparisons
  - each takes d operations

- What algorithm sees
  - Training set:
    - $\{1.3, 3.2, 4.3, 5.3, 6.4, 7.2, 8.0, 7.0, 9.0\}$
  - Testing instance:
    - $\{7.4\}$
  - Nearest neighbors:
    - compare one-by-one to each training instance
  - n comparisons
  - each takes d operations

Making kNN fast

- Training: O(d), but testing: O(nd)
- Reduce d: dimensionality reduction
  - simple feature selection, other methods O(d^2)
- Reduce n: don’t compare to all training examples
  - idea: quickly identify m<n potential near neighbors
    - compare only to these, pick k nearest neighbors: O(mn) time
  - K-D trees: low-dimensional, real-valued data
    - $O(d \log n)$ only works when $d < n$, may miss neighbors
  - Inverted lists: high-dimensional, discrete data
    - $O(d \log n)$ only works when $d < n$, may miss neighbors
  - Locality-sensitive hashing: high-d, discrete or real-valued
    - $O(d \log n)$, $k << n$, small fingerprint, may miss neighbors

Distance measures (2)

- Minkowski distance (p-norm): $D(x, x') = \left(\sum_{i=1}^{n}(x_i - x'_i)^p\right)^{1/p}$
  - p=2: Euclidean
  - p=1: Manhattan
  - p=0: Hamming → logical AND
  - p=-\infty: $max_{i} |x_i - x'_i|$ → logical OR
- Kullback-Leibler (KL) divergence:
  - for histograms $\sum_{i} x_i > 0, \sum_{i} y_i = 1$:
  - $D(x, x') = -\sum_{i} x_i \log \frac{x_i}{y_i}$
  - asymmetric, excess bits to encode with x with x'}
K-D tree example

- Building a K-D tree from training data:
  - pick random dimension, find median, split data, repeat
- Find NNs for new point (7,4)
  - find region containing (7,4)
  - compare to all points in region

Locality-Sensitive Hashing (LSH)

- Random hyper-planes \( h_1, \ldots, h_k \)
  - space sliced into \( 2^k \) regions (polytopes)
  - compare \( x \) only to training points in the same region \( R \)
- Complexity: \( O(kd + 2n/k) \)
  - \( O(kd) \) to find region \( R \), \( k \ll n \)
  - dot-product \( x \) with \( h_i \)
  - compare to \( n/2^k \) points in \( R \)
- Inexact: missed neighbors
  - repeat with different \( h_1, \ldots, h_k \)
- Why not K-D tree?

Inverted list example

- Data structure used by search engines (Google, etc)
  - list all training examples that contain particular attribute
  - assumption: most attribute values are zero (sparseness)
- Given a new testing example:
  - merge inverted lists for attributes present in new example
  - \( O(n) \): \( d \) non-zero attributes, \( n \) avg. length of inverted list

<table>
<thead>
<tr>
<th>Example</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1: “send your password”</td>
<td>spam</td>
</tr>
<tr>
<td>D2: “send us review”</td>
<td>ham</td>
</tr>
<tr>
<td>D3: “send us password”</td>
<td>spam</td>
</tr>
<tr>
<td>D4: “send us details”</td>
<td>ham</td>
</tr>
<tr>
<td>D5: “send your password”</td>
<td>spam</td>
</tr>
<tr>
<td>D6: “review your account”</td>
<td>spam</td>
</tr>
<tr>
<td>D7: “new email”</td>
<td>account</td>
</tr>
</tbody>
</table>

Note: Examples are from various online sources.