Lecture 3
Machine Learning
K Nearest Neighbor Classifier
Today

• kNN classifier - the simplest classifier on earth
• matlab implementation of kNN
k-Nearest Neighbors

• classify an unknown example with the most common class among \( k \) closest examples
  • “tell me who your neighbors are, and I’ll tell you who you are”

• Example:
  • \( k = 3 \)
  • 2 sea bass, 1 salmon
  • Classify as sea bass
kNN: Multiple Classes

- Easy to implement for multiple classes
- Example for $k = 5$
  - 3 fish species: salmon, sea bass, eel
  - 3 sea bass, 1 eel, 1 salmon $\Rightarrow$ classify as sea bass
kNN: How to Choose k?

• In theory, if infinite number of samples available, the larger is $k$, the better is classification

• But the caveat is that all $k$ neighbors have to be close
  • Possible when infinite # samples available
  • Impossible in practice since # samples is finite
kNN: How to Choose k?

- Rule of thumb is $k = \sqrt{n}$, $n$ is number of examples

- interesting theoretical properties

- In practice, $k = 1$ is often used for efficiency, but can be sensitive to “noise”

In the diagram:
- **1 NN**: every example in the blue shaded area will be misclassified as the blue class.
- **3 NN**: every example in the blue shaded area will be classified correctly as the red class.
• Larger $k$ gives smoother boundaries, better for generalization
  • But only if locality is preserved. Locality is not preserved if end up looking at samples too far away, not from the same class.
• Interesting theoretical properties if $k < \sqrt{n}$, $n$ is # of examples
• Can choose $k$ through cross-validation (study soon)
kNN: How Well does it Work?

• kNN is simple and intuitive, but does it work?
• Theoretically, the best error rate is the Bayes rate $E^*$
  • Bayes error rate is the best (smallest) error rate a classifier can have, for a given problem, but we do not study it in this course
• Assume we have an unlimited number of samples
• kNN leads to an error rate greater than $E^*$
• But even for $k = 1$, as $n \to \infty$, it can be shown that kNN error rate is smaller than $2E^*$
• As we increase $k$, the upper bound on the error gets better, that is the error rate (as $n \to \infty$) for the kNN rule is smaller than $cE^*$, with smaller $c$ for larger $k$
• If we have lots of samples, kNN works well
• Voronoi tessellation is useful for visualization
kNN Selection of Distance

• So far we assumed we use Euclidian Distance to find the nearest neighbor:

\[ D(a, b) = \sqrt{\sum_{k} (a_k - b_k)^2} \]

• Euclidean distance treats each feature as equally important
• However some features (dimensions) may be much more discriminative than other features
kNN Distance Selection: Extreme Example

• feature 1 gives the correct class: 1 or 2
• feature 2 gives irrelevant number from 100 to 200
• dataset: \[ \begin{bmatrix} 1 & 150 \\ 2 & 110 \end{bmatrix} \]
• classify \[ \begin{bmatrix} 1 & 100 \end{bmatrix} \]

\[ D(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 1 \\ 150 \end{bmatrix}) = \sqrt{(1-1)^2 + (100-150)^2} = 50 \]

\[ D(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 2 \\ 110 \end{bmatrix}) = \sqrt{(1-2)^2 + (100-110)^2} = 10.5 \]

• \[ \begin{bmatrix} 1 & 100 \end{bmatrix} \] is misclassified!
• The denser the samples, the less of this problem
• But we rarely have samples dense enough
• Decision boundary is in red, and is really wrong because
  • feature 1 is discriminative, but it’s scale is small
  • feature 2 gives no class information but its scale is large, it dominates distance calculation
kNN: Feature Normalization

• Notice that 2 features are on different scales:
  • First feature takes values between 1 or 2
  • Second feature takes values between 100 to 200
• Idea: normalize features to be on the same scale
• Different normalization approaches
• Linearly scale the range of each feature to be, say, in range $[0,1]$

\[
f_{new} = \frac{f_{old} - f_{old}}{f_{max} - f_{min}}
\]
kNN: Feature Normalization

- Linearly scale to 0 mean variance 1:
- If \( Z \) is a random variable of mean \( \mu \) and variance \( \sigma^2 \), then \( (Z - \mu)/\sigma \) has mean 0 and variance 1
- For each feature \( f \) let the new rescaled feature be

\[
 f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}
\]

- \( C \) is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

\[
 C_{\text{new}} = (C - \text{repmat(mean}(C), \text{size}(C,1), 1))*\text{diag}(1./\text{std}(C))
\]
**kNN: Feature Normalization**

- $\mathbf{C}$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously.

$$f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}$$

$$\mathbf{C}_{\text{new}} = (\mathbf{C} - \text{repmat(mean}(\mathbf{C}), \text{size}(\mathbf{C}, 1), 1)) \ast \text{diag}(1./\text{std}(\mathbf{C}))$$

$$\mathbf{C} = \begin{bmatrix}
1 & 180 \\
1 & 100 \\
1 & 160 \\
2 & 120 \\
2 & 150 \\
2 & 170
\end{bmatrix}$$

$\text{size}(\mathbf{C}, 1) = 6$
• \( C \) is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

\[
C_{\text{new}} = (C - \text{repmat}(\text{mean}(C), 6, 1)) \cdot \text{diag}(1./\text{std}(C))
\]

\[
f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}
\]
• $C$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

$$C_{new} = (C - \text{repmat(mean}(C), 6, 1)) \times \text{diag}(1./\text{std}(C))$$

$$[1.5 \quad 146.7]$$

$$C = \begin{bmatrix}
1 & 180 \\
1 & 100 \\
1 & 160 \\
2 & 120 \\
2 & 150 \\
2 & 170
\end{bmatrix}$$

$$f_{new} = \frac{f_{old} - \mu}{\sigma}$$
\[ \text{repmat}\left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, 2, 3\right) = \begin{bmatrix} 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \end{bmatrix} \]
kNN: Feature Normalization

- \( C \) is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

\[
f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}
\]

\[
C_{\text{new}} = (C - \text{repmat}(\text{mean}(C), 6, 1)) \ast \text{diag}(1./\text{std}(C))
\]

\[
\begin{bmatrix}
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
\end{bmatrix}
\]

mean\((C) = [1.5 \ 146.7]\)
kNN: Feature Normalization

- $\mathbf{C}$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

$$f_{new} = \frac{f_{old} - \mu}{\sigma}$$

$$\mathbf{C}_{new} = (\mathbf{C} - \text{repmat}(\text{mean}(\mathbf{C}), 6, 1)) * \text{diag}(1./\text{std}(\mathbf{C}))$$

$$\begin{bmatrix}
1 & 180 \\
1 & 100 \\
1 & 160 \\
2 & 120 \\
2 & 150 \\
2 & 170 \\
\end{bmatrix} - \begin{bmatrix}
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
1.5 & 146.7 \\
\end{bmatrix} = \begin{bmatrix}
-0.5 & 33.3 \\
-0.5 & -46.7 \\
-0.5 & 13.3 \\
-0.5 & -26.7 \\
-0.5 & 3.3 \\
1.5 & 23.3 \\
\end{bmatrix}$$
kNN: Feature Normalization

- \( C \) is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

\[
C_{\text{new}} = (C - \text{repmat(\text{mean}(C), 6, 1))) \ast \text{diag}(1./\text{std}(C))
\]

\[
\text{std}(C) = [0.55 \quad 30.8]
\]

\[
C = \begin{bmatrix}
1 & 180 \\
1 & 100 \\
1 & 160 \\
2 & 120 \\
2 & 150 \\
2 & 170
\end{bmatrix}
\]

\[
f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}
\]
kNN: Feature Normalization

- $\mathbf{C}$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously
  
  $\mathbf{C}_{\text{new}} = (\mathbf{C} - \text{repmat(\text{mean}(\mathbf{C}), 6, 1)}) \ast \text{diag}(1./\text{std}(\mathbf{C}))$

  \[
  f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}
  \]

  $1./\text{std}(\mathbf{C}) = [1.83 \ 0.03]$

  $\text{std}(\mathbf{C}) = [0.55 \ 30.8]$
kNN: Feature Normalization

- $C$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

$$C_{\text{new}} = (C - \text{repmat(mean}(C), 6, 1)) \ast \text{diag}(1./\text{std}(C))$$

$$f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}$$

\[
\begin{bmatrix}
1.83 & 0 \\
0 & 0.03
\end{bmatrix}
\]

$1./\text{std}(C) = [1.83 \ 0.03]$
kNN: Feature Normalization

- $\mathbf{C}$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

$$
\mathbf{C}_{\text{new}} = (\mathbf{C} - \text{repmat}(\text{mean}(\mathbf{C}), 6, 1)) \times \text{diag}(1./\text{std}(\mathbf{C}))
$$

$$
f_{\text{new}} = \frac{f_{\text{old}} - \mu}{\sigma}
$$

\[
\begin{pmatrix}
-0.5 & 33.3 \\
-0.5 & -46.7 \\
-0.5 & 13.3 \\
0.5 & -26.7 \\
0.5 & 3.3 \\
1.5 & 23.3 \\
\end{pmatrix}
\begin{pmatrix}
1.83 & 0 \\
0 & 0.03 \\
\end{pmatrix}
\begin{pmatrix}
-0.9 & 1.08 \\
-0.9 & -1.21 \\
-0.9 & 0.43 \\
0.9 & -0.87 \\
0.9 & 0.11 \\
0.9 & 0.76 \\
\end{pmatrix}
\]
kNN: Feature Normalization

- $C$ is a matrix with all samples stored as rows, in Matlab can normalize all features simultaneously

$$C_{new} = (C - \text{repmat(mean}(C), 6, 1)) \ast \text{diag}(1./\text{std}(C))$$

$$f_{new} = \frac{f_{old} - \mu}{\sigma}$$
kNN: Selection of Distance

- Feature normalization does not help in high dimensional spaces if most features are irrelevant

\[ D(a, b) = \sqrt{\sum_{k} (a_k - b_k)^2} = \sqrt{\sum_{i} (a_i - b_i)^2 + \sum_{j} (a_j - b_j)^2} \]

- If the number of useful features is smaller than the number of noisy features, Euclidean distance is dominated by noise
kNN: Feature Weighting

• Scale each feature by its importance for classification

\[ D(a, b) = \sqrt{\sum_k w_k (a_k - b_k)^2} \]

• Can use our prior knowledge about which features are more important

• Can learn the weights \( w_k \) using cross-validation (to be covered later)
Basic kNN algorithm stores all examples
Suppose we have $n$ examples each of dimension $d$
$O(d)$ to compute distance to one example
$O(nd)$ to find distances to all examples
$O(knd)$ to find $k$ closest examples examples
$O(nd) + O(kn)$ if careful
Thus total complexity is $O(knd)$
Very expensive for a large number of samples
But we need a large number of samples for kNN to work well!
Reducing Complexity: editing 1NN

- If all Voronoi neighbors have the same class, a sample is useless, remove it
- Number of samples decreases
- Decision boundary does not change
Reducing Complexity: Partial Distance

- Have current $k$ closes samples
- Abort distance computation if partial distance is already greater than the full distance to the current $k$ closest samples

Advantages:
- complexity decreases
- we are guaranteed to find closes neighbor(s)

Disadvantages:
- how much complexity decreases depends on our luck and data layout
kNN in Matlab

$\text{class1} = \begin{bmatrix} 3 & 8 \\ 5 & 9 \\ 7 & 10 \\ 6 & 8 \end{bmatrix}$

$\text{class2} = \begin{bmatrix} 2 & 4 \\ 3 & 7 \\ 5 & 4 \end{bmatrix}$

- Want to classify $\text{newSample} = [4 \ 7]$
numClass1 = size(Class1,1);
numClass2 = size(Class2,1);
totalSamples = numClass1+numClass2;
combinedSamples = [Class1;Class2];
trueClass = [zeros(numClass1,1)+1;zeros(numClass2,1)+2];
testMatrix = repmat(newSample,totalSamples,1);
absDiff = abs(combinedSamples-testMatrix);
absDiff = absDiff.^2;
dist = sum(absDiff,2);
[Y,I] = sort(dist);
neighborsInd = I(1:k);
neighbors = trueClass(neighborsInd);

class1 = find(neighbors == 1);
class2 = find(neighbors == 2);
joint = [size(class1,1);size(class2,1)];

[value class] = max(joint);
kNN in Matlab

numClass1    = size(Class1,1);
numClass2    = size(Class2,1);
totalSamples = numClass1+numClass2;
combinedSamples = [Class1;Class2];
trueClass = [zeros(numClass1,1)+1;zeros(numClass2,1)+2];

\[
\begin{bmatrix}
3 & 8 \\
5 & 9 \\
7 & 10 \\
6 & 8 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
2 & 4 \\
3 & 7 \\
5 & 4 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 \\
1 \\
1 \\
2 \\
2 \\
2 \\
\end{bmatrix}
\]
```matlab
% kNN in Matlab

testMatrix = repmat(newSample,totalSamples,1);
absDiff = abs(combinedSamples-testMatrix);
absDiff = absDiff.^2;
dist = sum(absDiff,2);

% Array Definitions
newSample = [4 7];
combinedSamples = [3 8; 5 9; 7 10; 6 8; 2 4; 3 7; 5 4];
trueClass = [1; 1; 1; 1; 2; 2; 2];

testMatrix = [4 7; 4 7; 4 7; 4 7; 4 7; 4 7; 4 7];
absDiff = [1 1; 1 2; 3 3; 2 1; 2 3; 1 0; 1 3];
dist = [2; 5; 18; 5; 13; 1; 10];
```
\[
[Y,I] = \text{sort}(\text{dist});
\]
\[
\text{neighborsInd} = I(1:k);
\]
\[
\text{neighbors} = \text{trueClass}(\text{neighborsInd});
\]
\[
\begin{bmatrix}
1 \\
2 \\
5 \\
5 \\
10 \\
13 \\
18
\end{bmatrix}
\]
\[
\begin{bmatrix}
6 \\
1 \\
2 \\
4 \\
7 \\
5 \\
3
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
2 \\
2 \\
2
\end{bmatrix}
\]
\[
dist = \begin{bmatrix}
2 \\
5 \\
18 \\
5 \\
13 \\
1 \\
10
\end{bmatrix}
\]
\[
\text{trueClass} = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
2 \\
2 \\
2
\end{bmatrix}
\]
\[
k = 3
\]
class1 = find(neighbors == 1);
class2 = find(neighbors == 2);
joint = [size(class1,1);size(class2,1)];

[value class] = max(joint);

\[
\begin{bmatrix}
2 \\
3 \\
\end{bmatrix}
\]

class1

\[
\begin{bmatrix}
1 \\
\end{bmatrix}
\]

class2

\[
\begin{bmatrix}
2 \\
1 \\
\end{bmatrix}
\]

joint

\[
\begin{bmatrix}
1 \\
1 \\
\end{bmatrix}
\]

class

\[
\begin{bmatrix}
2
\end{bmatrix}
\]

neighbors

\[
\begin{bmatrix}
1 \\
1 \\
\end{bmatrix}
\]
kNN in Matlab

class1 = find(neighbors == 1);
class2 = find(neighbors == 2);
joint = [size(class1,1);size(class2,1)];

[value class] = max(joint);

Also can use class = mode(neighbors) instead

neighbors = \[
\begin{bmatrix}
2 \\
1 \\
1 
\end{bmatrix}
\]

class1 = \[
\begin{bmatrix}
2 \\
3
\end{bmatrix}
\]

class2 = [1]

joint = \[
\begin{bmatrix}
2 \\
1
\end{bmatrix}
\]

class = 1
kNN in Matlab
Video

• http://videolectures.net/aaai07_bosch_knnn/
kNN Summary

• Advantages
  • Can be applied to the data from any distribution
    • for example, data does not have to be separable with a linear boundary
  • Very simple and intuitive
  • Good classification if the number of samples is large enough

• Disadvantages
  • Choosing $k$ may be tricky
  • Test stage is computationally expensive
    • No training stage, all the work is done during the test stage
    • This is actually the opposite of what we want. Usually we can afford training step to take a long time, but we want fast test step
  • Need large number of samples for accuracy