CS4442/9542b: Artificial Intelligence II
Prof. Olga Veksler

Lecture 3: Machine Learning

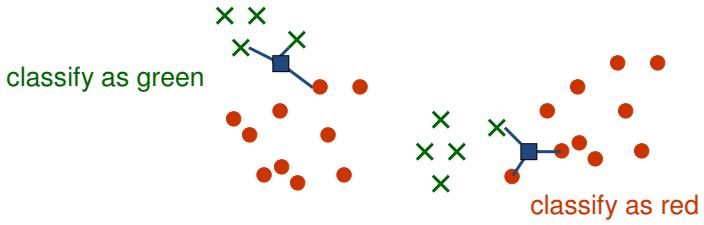
k Nearest Neighbors Classifier

Today

kNN classifier -- the simplest classifier on earth

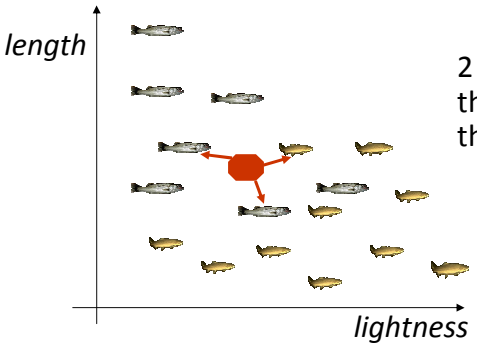
k-Nearest Neighbors

- classify an unknown example with the most common class "around" this example
  - "around" means ***k*** closest example (or ***k*** nearest "neighbors")
  - *"tell me who your neighbors are, and I'll tell you who you are"*



k-Nearest Neighbor: Example

- Back to fish sorting
  - 2 features (length and lightness)
  - Let ***k*** = 3



2 sea bass, 1 salmon are the 3 nearest neighbors, thus classify as sea bass

kNN: How Well Does it Work?

kNN rule is certainly simple and intuitive, but does it work?

Assume we have an unlimited number of samples

Theoretically, the best possible 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

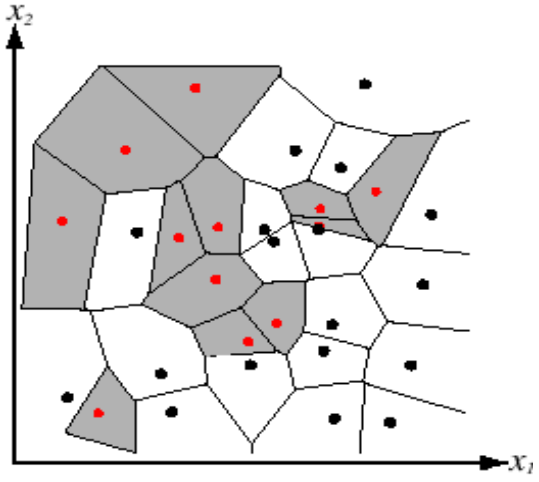
Nearest-neighbor rule leads to an error rate greater than *E**

But even for *k* =1, as $n \rightarrow \infty$, it can be shown that nearest neighbor rule error rate is smaller than *2E**

As we increase *k*, the upper bound on the error gets better and better, that is the error rate (as $n \rightarrow \infty$) for the *kNN* rule is smaller than *cE,with smaller *c* for larger *k***

If we have a lot of samples, the kNN rule will do very well !

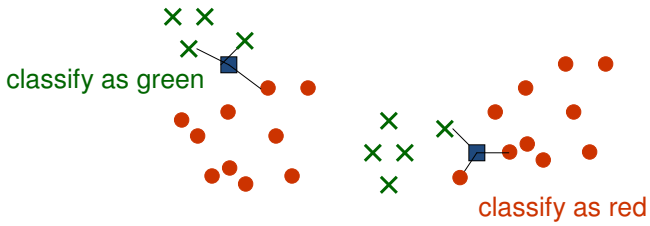
1NN: Voronoi Cells



kNN: How to Choose k?

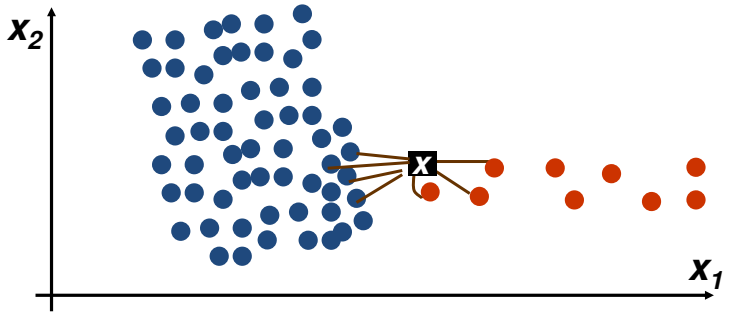
- **In theory, when the infinite number of samples is available, the larger the *k*, the better is classification (error rate gets closer to the optimal Bayes error rate)**
- **But the caveat is that all *k* neighbors have to be close to *x***
  - **Possible when infinite # samples available**
  - **Impossible in practice since # samples is finite**

kNN: How to Choose k?



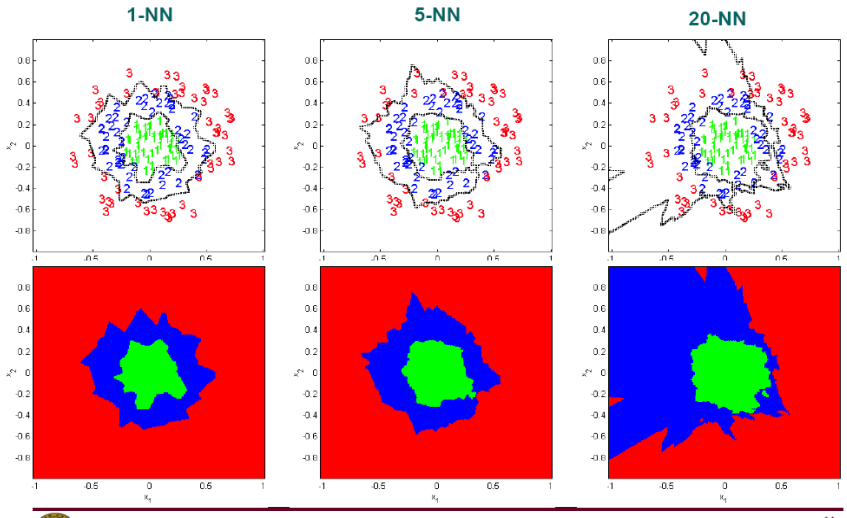
- How to choose ***k***?
  - "rule of thumb" is $k = \sqrt{n}$, where n is the number of samples
    - Interesting theoretical properties
  - in practice, $k = 1$ is often used for efficiency

kNN: How to Choose k?



- For k = 1, …,5 sample x is classified correctly as the red class
- For larger k classification of x is wrong (blue class)
- can find a good k through cross-validation, to be studied later

k-NN versus 1-NN

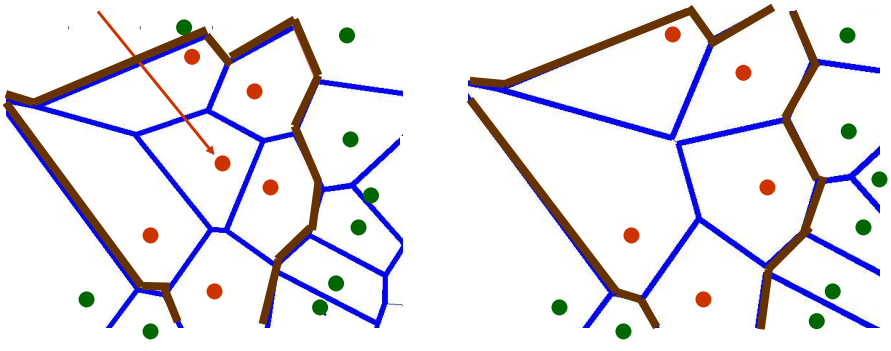


kNN: Computational Complexity

- 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 one nearest neighbor
  - ***O***(***knd***) to find ***k*** closest examples examples
  - Thus complexity is ***O***(***knd***)
- This is prohibitively expensive for large number of samples
- **But we need 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, we can remove it:**



- **number of samples decreases**
- **decision boundaries stay the same**

Reducing Complexity: Partial Distance

- compute partial distances using a subset of dimensions and eliminating the points with partial distances greater than the full distance of the current closest points
- Advantages:
  - complexity decreases
  - we are guaranteed to find closes neighbor(s)
- Disadvantages:
  - complexity may not decrease significantly, how much complexity decreases depends on our luck and data layout

kNN: Selection of Distance

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

$$D(a,b) = \sqrt{\sum_k (a_k - b_k)^2} = \sqrt{a \cdot b}$$

- However some features (dimensions) may be much more discriminative than other features (dimensions)
- Euclidean distance treats each feature as equally important

kNN: Selection of Distance

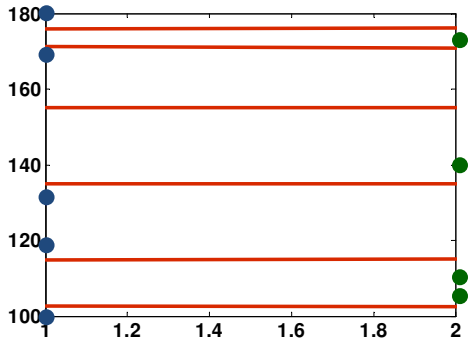
- Extreme Example
  - feature 1 gives the correct class: 1 or 2
  - feature 2 gives irrelevant number from 100 to 200
- Suppose we have to find the class of x=[1 100] and we have 2 samples [1 150] and [2 110]

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

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

- x = [1 100] is misclassified!
- The denser the samples, the less of the problem
  - But we rarely have samples dense enough

kNN: Extreme Example



- decision boundaries for blue and green classes are in red
- These boundaries are really bad because
  - feature 1 is discriminative, but it's scale is small
  - feature 2 gives no class information but its scale is large

kNN: Selection of Distance

- Notice the 2 features are on different scales:
  - feature 1 takes values between 1 or 2
  - feature 2 takes values between 100 to 200
- Need to "normalize" feature values to be on the same scale
- Two approaches:
  1. linearly scale the range of each feature to be, say, in [0,1]

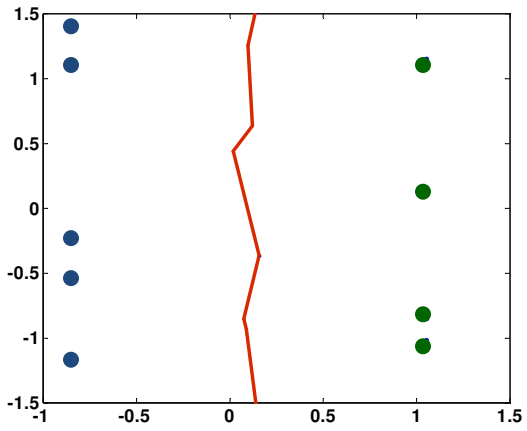
$$f_{new} = \frac{f_{old} - f_{min}}{f_{max} - f_{min}}$$

kNN: Selection of Distance

2. linearly scale to zero mean variance 1:
   - If Z is a random variable of mean m and variance σ^2, then $(Z - m)/\sigma$ has mean 0 and variance 1
   - Thus for each feature f, compute its sample mean and variance, and let the new feature be $[f - \text{mean}(f)]/\text{sqrt}[\text{var}(f)]$
   - If C is a matrix with all the samples piled up as rows (that is ith column of C has the ith feature), then can do this in matlab for all features simultaneously

     Cn=(C-repmat(mean(C),size(C,1),1))*diag(1./sqrt(var(C)))
- Let's apply scaling to zero mean, variance 1 to previous example

kNN: Normalized Features



The decision boundary (in red) is very good now!

kNN: Selection of Distance

- **However in high dimensions if there are a lot of irrelevant features, normalization will not help**

$$D(a,b) = \sqrt{\sum_k (a_k - b_k)^2} = \sqrt{\underbrace{\sum_i (a_i - b_i)^2}_{\text{discriminative feature}} + \underbrace{\sum_j (a_j - b_j)^2}_{\text{noisy features}}}$$

- **If the number of discriminative 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

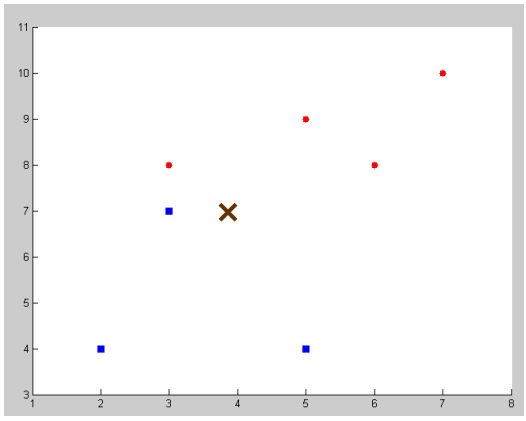
$$D(a,b) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$

- Can use our prior knowledge about which features are more important
- Can learn the weights w_k from the validation data
  - Increase/decrease weights until classification improves

kNN in Matlab

$$Class1 = \begin{bmatrix} 2 & 4 \\ 3 & 7 \\ 5 & 4 \end{bmatrix}$$

$$Class2 = \begin{bmatrix} 3 & 8 \\ 5 & 9 \\ 7 & 10 \\ 6 & 8 \end{bmatrix}$$



- Want to classify sample $x = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$

kNN Code in Matlab without Loops

```
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);
```

$$Class1 = \begin{bmatrix} 2 & 4 \\ 3 & 7 \\ 5 & 4 \end{bmatrix}$$

$$Class2 = \begin{bmatrix} 3 & 8 \\ 5 & 9 \\ 7 & 10 \\ 6 & 8 \end{bmatrix}$$

$$x = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

$k = 3$

kNN Code 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;];
```

$numClass1 = 3$

$numClass2 = 4$

$totalSamples = 7$

$$combinedSamples = \begin{bmatrix} 2 & 4 \\ 3 & 7 \\ 5 & 4 \\ 3 & 8 \\ 5 & 9 \\ 7 & 10 \\ 6 & 8 \end{bmatrix} \qquad trueClass = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

$$Class1 = \begin{bmatrix} 2 & 4 \\ 3 & 7 \\ 5 & 4 \end{bmatrix}$$

$$Class2 = \begin{bmatrix} 3 & 8 \\ 5 & 9 \\ 7 & 10 \\ 6 & 8 \end{bmatrix}$$

$$x = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

kNN Code in Matlab

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

$$testMatrix = \begin{bmatrix} 4 & 7 \\ 4 & 7 \\ 4 & 7 \\ 4 & 7 \\ 4 & 7 \\ 4 & 7 \\ 4 & 7 \end{bmatrix} \quad absDiff = \begin{bmatrix} 2 & 3 \\ 1 & 0 \\ 1 & 3 \\ 1 & 1 \\ 1 & 2 \\ 3 & 3 \\ 2 & 1 \end{bmatrix}$$

$$absDiff = \begin{bmatrix} 4 & 9 \\ 1 & 0 \\ 1 & 9 \\ 1 & 1 \\ 1 & 4 \\ 9 & 9 \\ 4 & 1 \end{bmatrix} \quad dist = \begin{bmatrix} 13 \\ 1 \\ 10 \\ 2 \\ 5 \\ 18 \\ 5 \end{bmatrix}$$

$$x = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

$$combinedSamples = \begin{bmatrix} 2 & 4 \\ 3 & 7 \\ 5 & 4 \\ 3 & 8 \\ 5 & 9 \\ 7 & 10 \\ 6 & 8 \end{bmatrix}$$

$$trueClass = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

kNN Code in Matlab

```
[Y,I] = sort(dist);
neighborsInd = I(1:k);
neighbors = trueClass(neighborsInd);
```

$$Y = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 5 \\ 10 \\ 13 \\ 18 \end{bmatrix} \quad I = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 7 \\ 3 \\ 1 \\ 6 \end{bmatrix}$$

$$neighborsInd = \begin{bmatrix} 2 \\ 4 \\ 5 \end{bmatrix} \quad neighbors = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

$$dist = \begin{bmatrix} 13 \\ 1 \\ 10 \\ 2 \\ 5 \\ 18 \\ 5 \end{bmatrix}$$

$$trueClass = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

$$k = 3$$

kNN Code in Matlab

```
class1=find(neighbors == 1);
class2=find(neighbors == 2);
joint = [size(class1,1);size(class2,1)];
[value class]=max(joint);
```

$$class1 = [1]$$

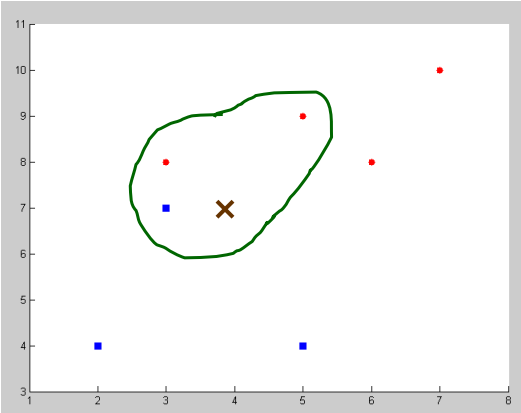
$$class2 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$joint = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$class = 2$$

$$neighbors = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

kNN Code in Matlab



Video

http://videolectures.net/aaai07_bosch_knnc/

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 best ***k*** may be difficult
- Test stage is computationally expensive
  - No training stage (in the "vanilla" version), all the work is done during the test stage. This is actually the opposite of what we want ideally: usually we can afford training step to take a long time, but test step we want to be very fast.
- Need large number of samples for accuracy