Pattern Classification and Recognition:

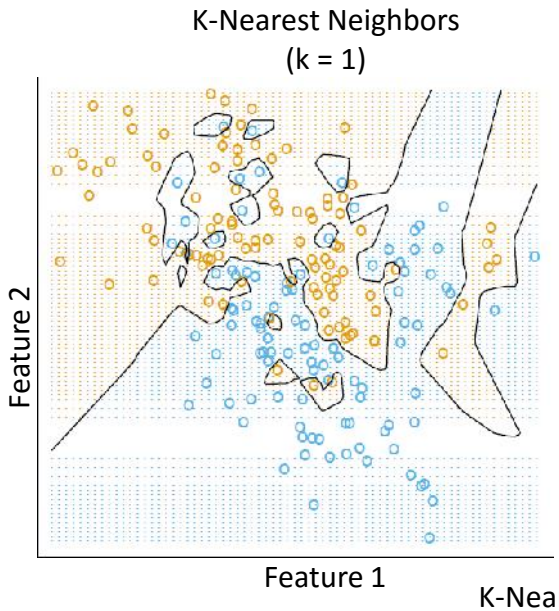
Nearest Neighbors Classification

ECE 681

Spring 2016

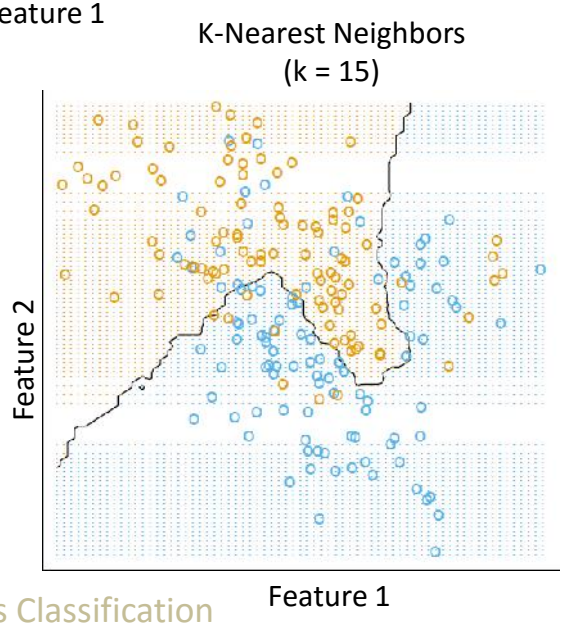
Stacy Tantum, Ph.D.

k-Nearest Neighbors (KNN) Classification



What does it do?

compare a new test data point
to previously seen training data



The k number of neighbors is belong to one class

What do we need?

What is closest?

How to mesure distance

Decision rules

Measuring Distance

"How close" or
"How far apart"
are two points?

Properties of Distance Metrics

- Reflexivity

  Dis(x,x)= 0

- Symmetry

  Dis(x,y)=Dis(y,x)

- Non-negativity

  Dis(x,y)>0, x!=y

- Triangle Inequality

  Dis(x,z)>Dis(x,y)+Dis(y,z)

Minkowski Distance (of order p > 0)

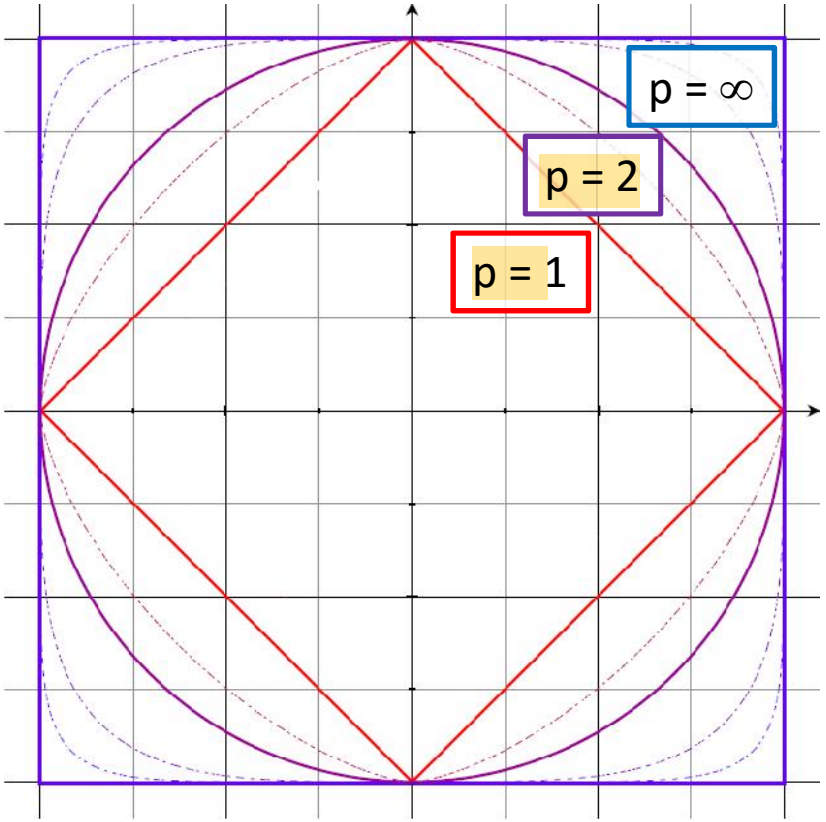
L1-> city block distance

L2->euclidean distance as the crow files

L(infinite)->chebyshev distance

L0->Hamming distance

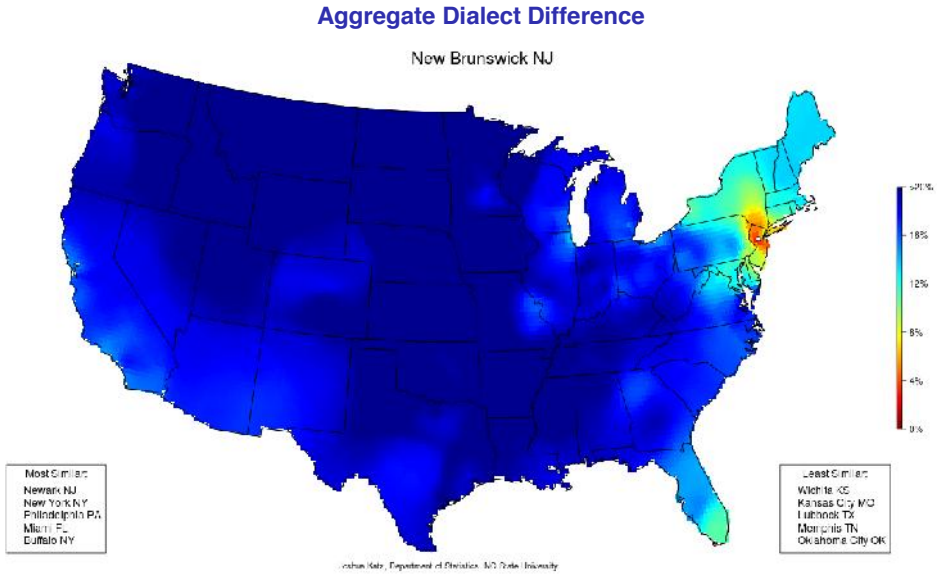
For d element vectors with point x and y:
Dis(x,y) = SUM(|xj-yj|^p)^(1/p)
so called p-norm

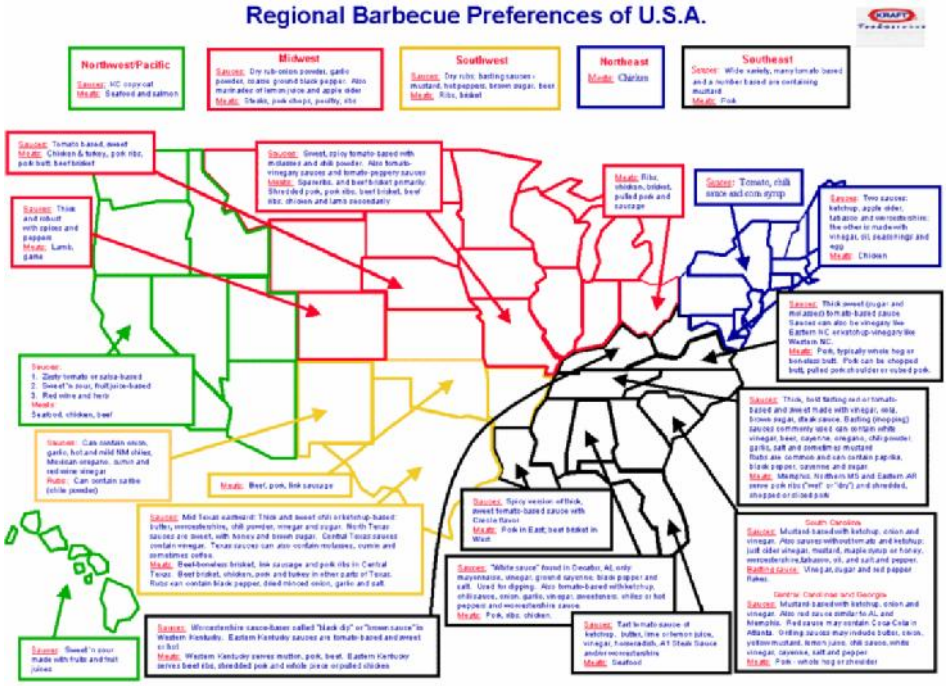


Lines of Mikowski distance from the origin equal to 1 for various orders p

Problem-Dependent Distance Metrics

"Distance" doesn't necessarily need to be the length of the path travelled

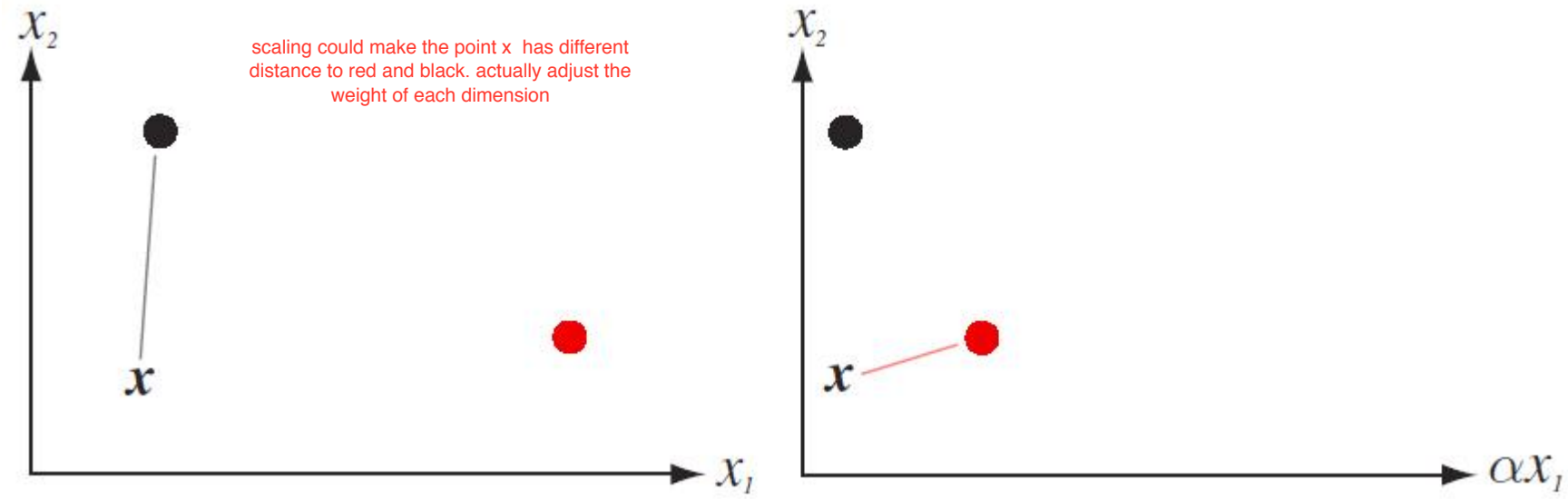




Training Data (Examplars)

Scaling (often) matters!

alter decision statistic and more easy to find decision



Data Scaling (Normalization)

- Autoscaling (z-Scoring)

  zero-mean, unit variance

- Maximum scaling

  not useful, may assume lower bound is 0 and xj is positive

  xj_new=xj/Vj

- Range scaling

  Lj is the lower bound of jth feature

  Vj is the upper bound of jth feature

  xj_new=(xj-Lj)/(Vj-Lj)

KNN Decision Rule

Assign "class-of-interest"
e.g., "red class"

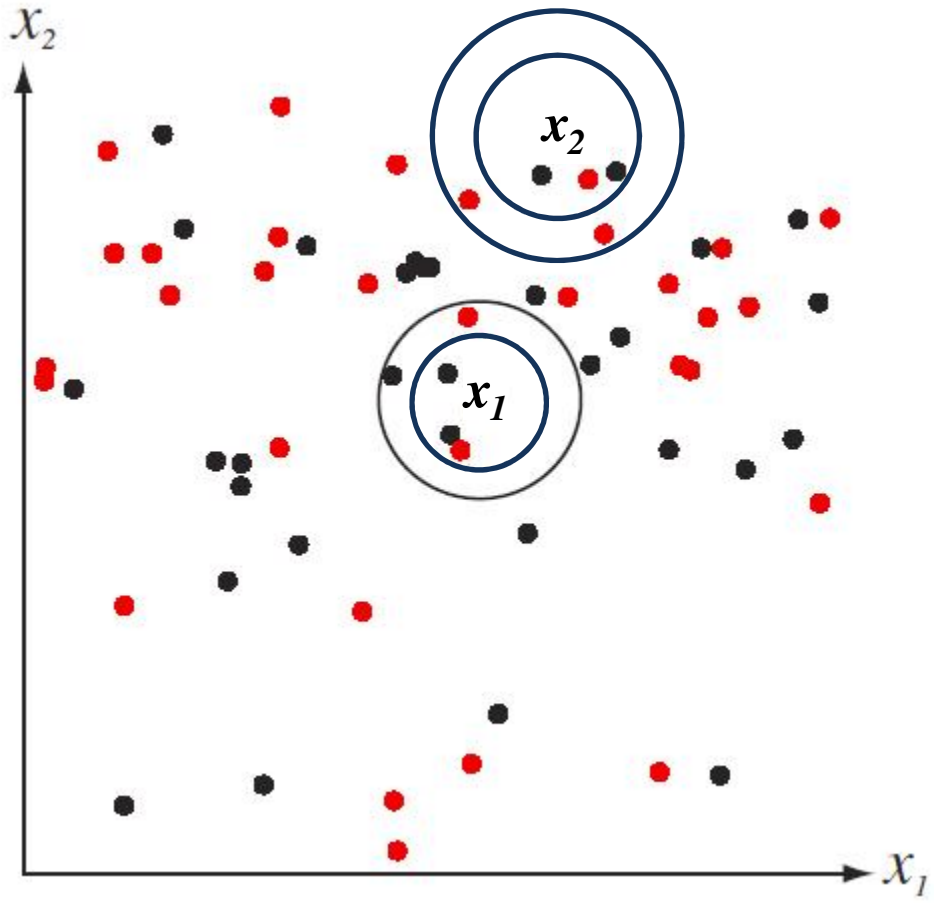
Assign indicator variables
e.g., $I_{red} = 1$ and $I_{black} = 0$

$$\lambda(x) = \frac{1}{k}\sum I \text{ for k nearest points}$$

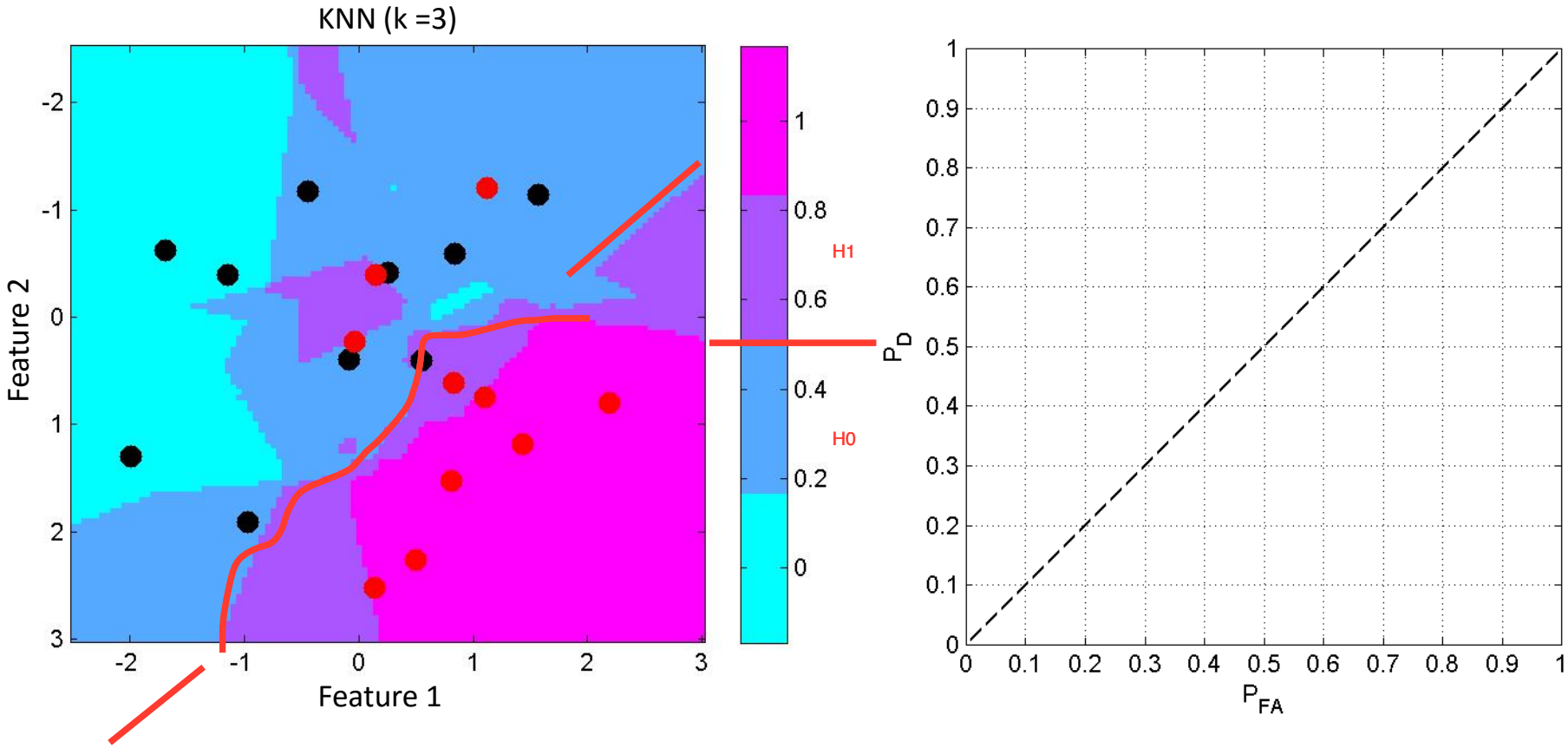
$$= \frac{\#\ \text{red data points in k nearest}}{k}$$

>0.5, choose red, other choose black

k means k point, could change the outcome



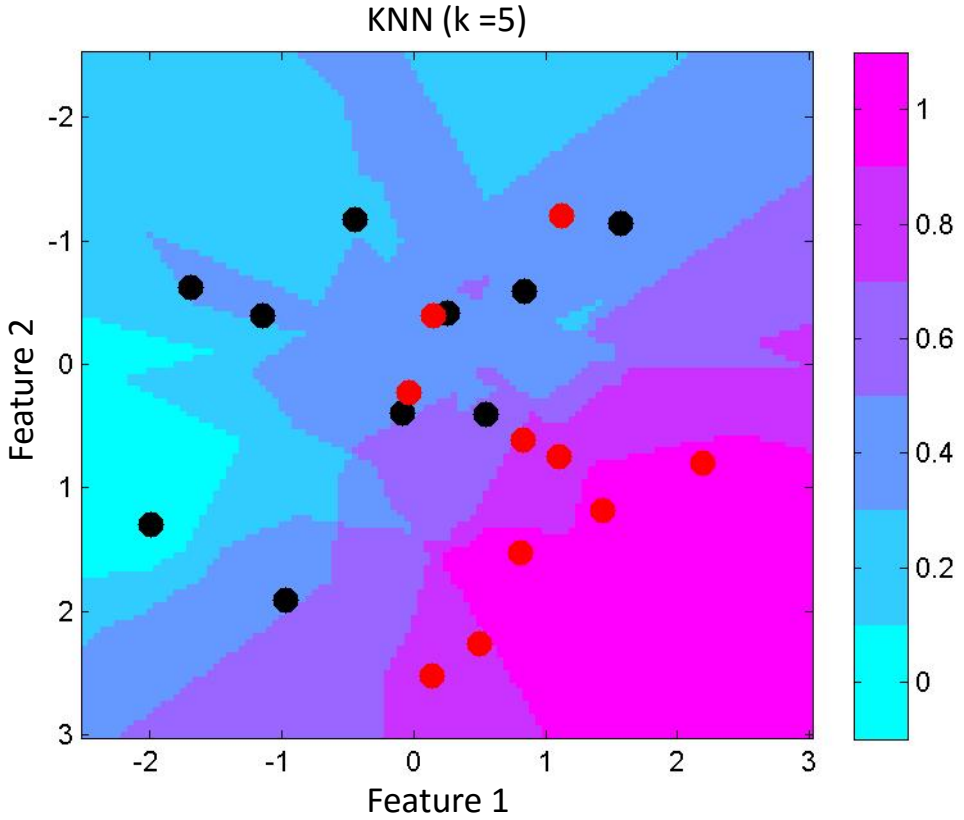
KNN Visualization & Performance Evaluation

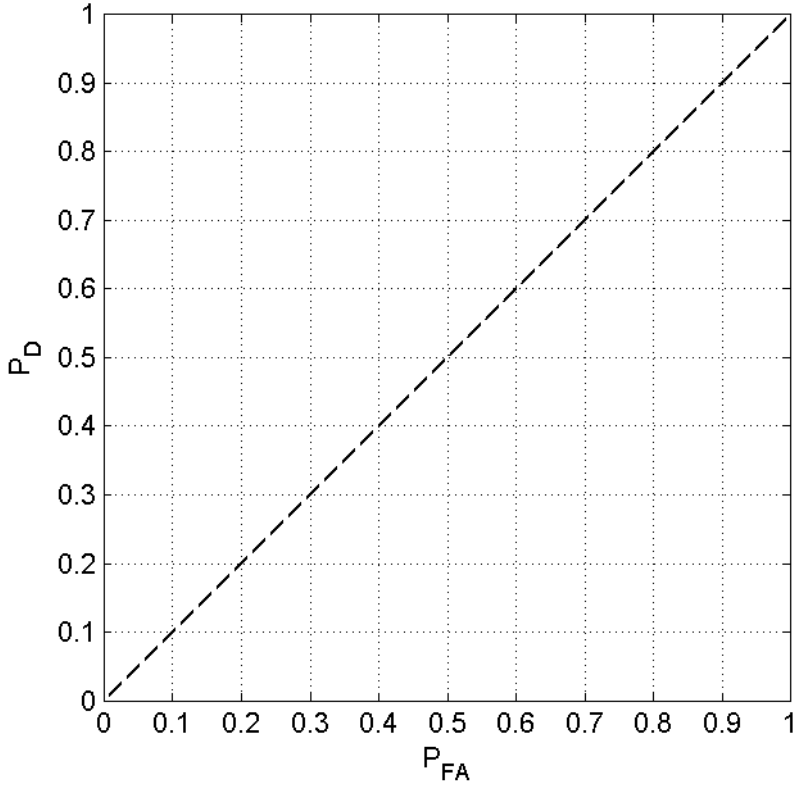


Majority decision boundary, the number of boundary = point on ROC(link each point)

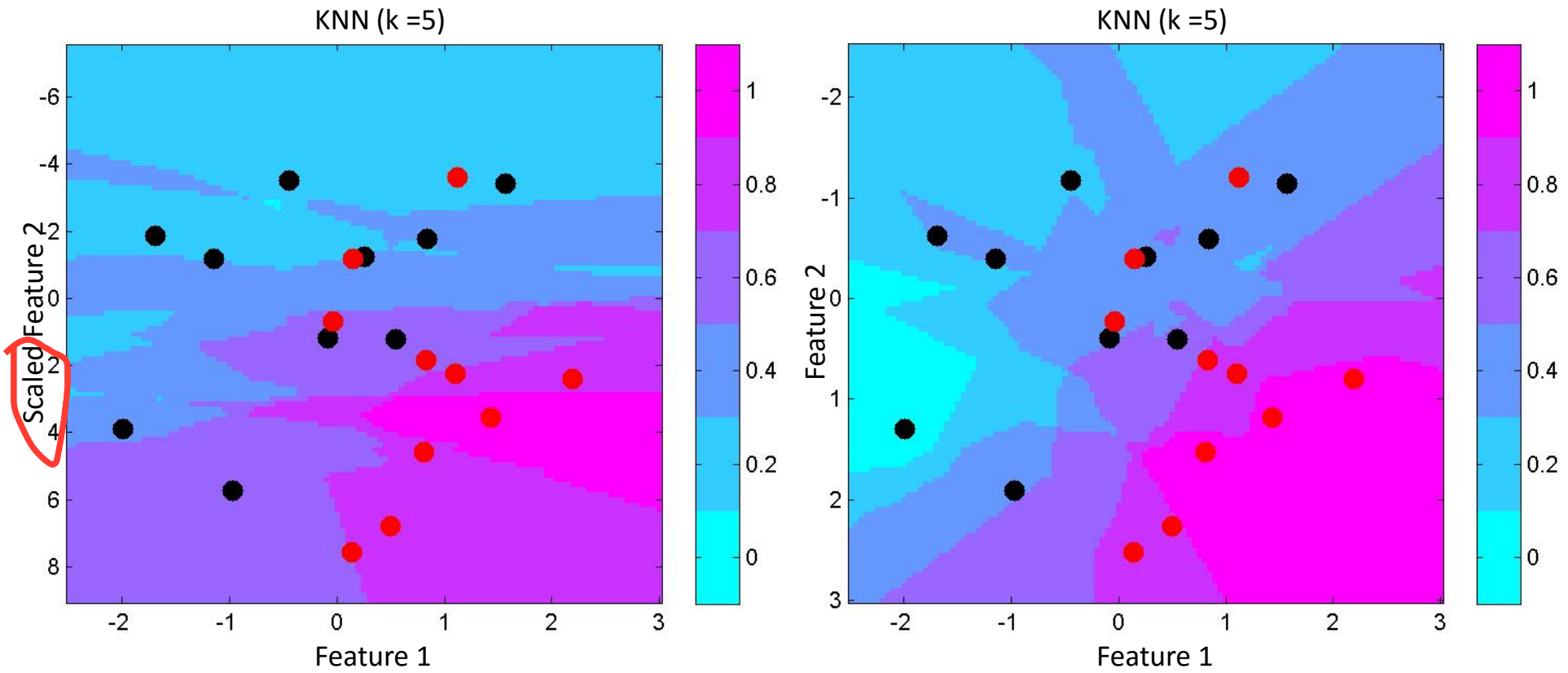
Than count the red and black point number/ the whole number of points = the frequency

KNN Visualization & Performance Evaluation





KNN Visualization & Performance Evaluation



Pfa/Pd=0.3/0.8

Pfa/Pd=0.1/0.7

Pcorrect decision = Pd * (nH1/ntotal)+(1-Pfa)*(nH0/ntotal)

Classifier Coding Tips / Suggestions

classifier = trainClassifier(features,targets,classifierOptions)

- return a classifier structure
  - *include the training features and targets (useful for visualization)*

decisionStatistics = runClassifier(testFeatures,classifier)

plotClassifier(classifier)

- produces an image of the decision surface with training data points super-imposed on top
  - *This is where having the training features and targets in the classifier structure is useful*

Classifiers: Training and Testing

Code your own classifier

- **Using a canned function is not ok**

What do you need to run the classifier?

Save those things:

- Parameters
- Calculated quantities

and also save what you would need to replicate training (helpful for trouble-shooting):

- Classifier type (i.e., "KNN")
- Training data (including relevant metadata)
- Training parameters

Suggestion: Save all the classifier information in a structure (that you design)

If everything you need is saved with the classifier, all you need to run the classifier to generate a decision statistic for each test point is

- The classifier
- The training data

Possibly helpful function: `eval`

- Allows you to dynamically create a command to be executed by matlab

Classifiers: Visualizing Decision Surfaces

Hypothesize a grid of test data

Calculate the decision statistic at each grid point

- By running the classifier with the list of hypothesized test data as the test data

Image (`imagesc`) the grid of decision statistics (the decision surface)

assume data already in range 0->255

map date to 0-255 for you

Using only information stored in the classifier structure:

```
x1 = linspace(min(xTrain(:,1))*0.8,
max(xTrain(:,1))*1.2,251);
x2 = linspace(min(xTrain(:,2))*0.8,
max(xTrain(:,2))*1.2,251);

% Create the grid of test data points
[xTest1,xTest2] = meshgrid(x1,x2);

% Each column is a feature
xTest = [xTest1(:) xTest2(:)];

% Run the classifier with these test data
dsTest = runClassifier(classifierStructure,xTest);

% dsTest is a vector, reshape it to a matrix
dsTest = reshape(dsTest,length(x2),length(x1));

% Image the decision statistic surface
imagesc(x1([1 end],x2([1 end]),dsTest)

% Add the training data points to the surface
hold on
% H0
plot(xTrain(target==0,1),xTrain(target==0,2),'b*')
% H1
plot(xTrain(target==1,1),xTrain(target==1,2),'ro')
```