Classifiers

Mark Hasegawa-Johnson

University of Illinois

September 19, 2018

Outline

Definition of a Classifier

A classifier is a function $h : \mathcal{X} \to \mathcal{Y}$.

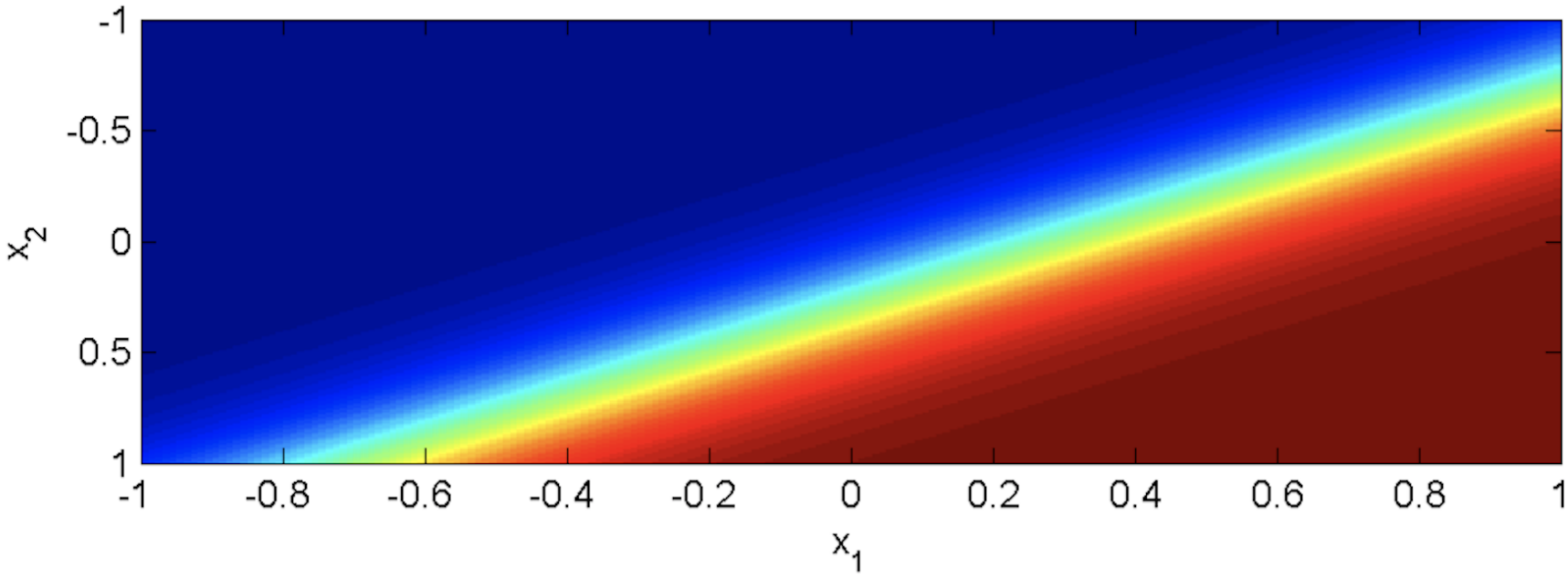
- $x \in \mathcal{X}$ is a "feature vector" (a.k.a. "observation," a.k.a. "data point," a.k.a. "token"). Examples:
  - Real-valued feature vector: $\mathcal{X} = \Re^d$
  - Arbitrary-length vector sequence: $\mathcal{X} = (\Re^d)^*$
- $y \in \mathcal{Y}$ is a label. Examples:
  - Category label, e.g.,: $\mathcal{Y} = \{\text{cat, dog, bunny, boy, girl}\}$
  - Binary label: $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, +1\}$
  - Quality score: $\mathcal{Y} = \Re$

Example: Linear Dichotomizer

A *dichotomizer* is a classifier with binary labels. A *linear classifier* is one that classifies its input by thresholding a linear function. A *linear dichotomizer* $h : \Re^d \to \{-1, 1\}$ is defined by a weight vector, $\vec{w} \in \Re^d$, and a bias $b \in \Re$:

$$h(\vec{x}) = \text{sign}\left(\vec{w}^T \vec{x} + b\right)$$

Example: Linear Dichotomizer



- The label "blue" is given to all vectors above the yellow line, the label "red" to all below it.
- The yellow line is called the **separatrix**. For a linear dichotomizer, the separatrix is the following set:

$$\text{separatrix} = \left\{ \vec{x} : \vec{w}^T \vec{x} + b = 0 \right\}$$

- The vector $\vec{w}$ is perpendicular to the separatrix.

Example: Linear Polychotomizer

A *polychotomizer* classifies its output into one of a finite set of output classes. A *linear polychotomizer* $h : \Re^d \rightarrow \{0, \ldots, M-1\}$ is defined by M different weight vectors, $\vec{w}_m \in \Re^d$, and biases $b_m \in \Re$, for $0 \leq m \leq M-1$:

$$h(\vec{x}) = \underset{m}{\text{argmax}} \left(\vec{w}_m^T \vec{x} + b_m \right)$$

Classification

Classification is the following process:

- Input: an observation, x, whose correct label is unknown.
- Output: $\hat{y} = h(x)$, a hypothesis about the label.

Training a Classifier

Training a classifier is the following process:

- Input: A set of n labeled training examples, $\mathcal{D}_{train} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Output: A set of parameters, θ, that defines the function $h(x)$.
  - **Example:** For a linear polychotomizer, the set of parameters is $\theta = \{\vec{w}_0, b_0, \ldots, \vec{w}_{M-1}, b_{M-1}\}$, and the classification function given these parameters is $h(\vec{x}) = \text{argmax}_m \left(\vec{w}_m^T \vec{x} + b_m\right)$.

Testing a Classifier

Testing a Classifer is the following process:

- Input: A set of m labeled testing examples, $\mathcal{D}_{test} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
- Process
  - $\mathcal{L}(y_i, \hat{y}_i)$ is the **loss** that you incur if the **reference label** is y_i, but the **hypothesis** is $\hat{y}_i$. The most common is the zero-one loss function ($\mathcal{L} = 0$ if no error, $\mathcal{L} = 1$ if error):

$$\mathcal{L}(y_i, \hat{y}_i) = [y_i \neq \hat{y}_i]$$

Here, $[\cdot]$ is called the "unit indicator function," defined as

$$[P] = \begin{cases} 1 & P \text{ true} \\ 0 & P \text{ false} \end{cases}$$

Testing a Classifier

Testing a Classifer is the following process:

- Input: A set of m labeled testing examples, $\mathcal{D}_{test} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
- Output: error rate, which might be just be the average per-token loss (the classification error rate):

$$\mathcal{L}(\mathcal{D}_{test}, h) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(y_i, h(x_i))$$

Train, Dev, and Eval Datasets

Suppose you have M different ideas for how you build a classifier. Each of your M different ideas involves training a set of parameters.

- Each of the M classifiers is first trained using your **Training Set**.
- Then you compute the error rate using the **Development Test Set** (devset). You choose one of the M classifiers: the one that has the lowest error rate on the devset.
- Then, finally, you compute the error rate of your best classifier on the **Evaluation Test Set**. This gives you an estimate of how well the classifier will actually perform in the real world.

Typically we use about 70% of the data for training, 15% for dev, and 15% for eval, though that varies.

Outline

Nearest-Neighbor Classifier

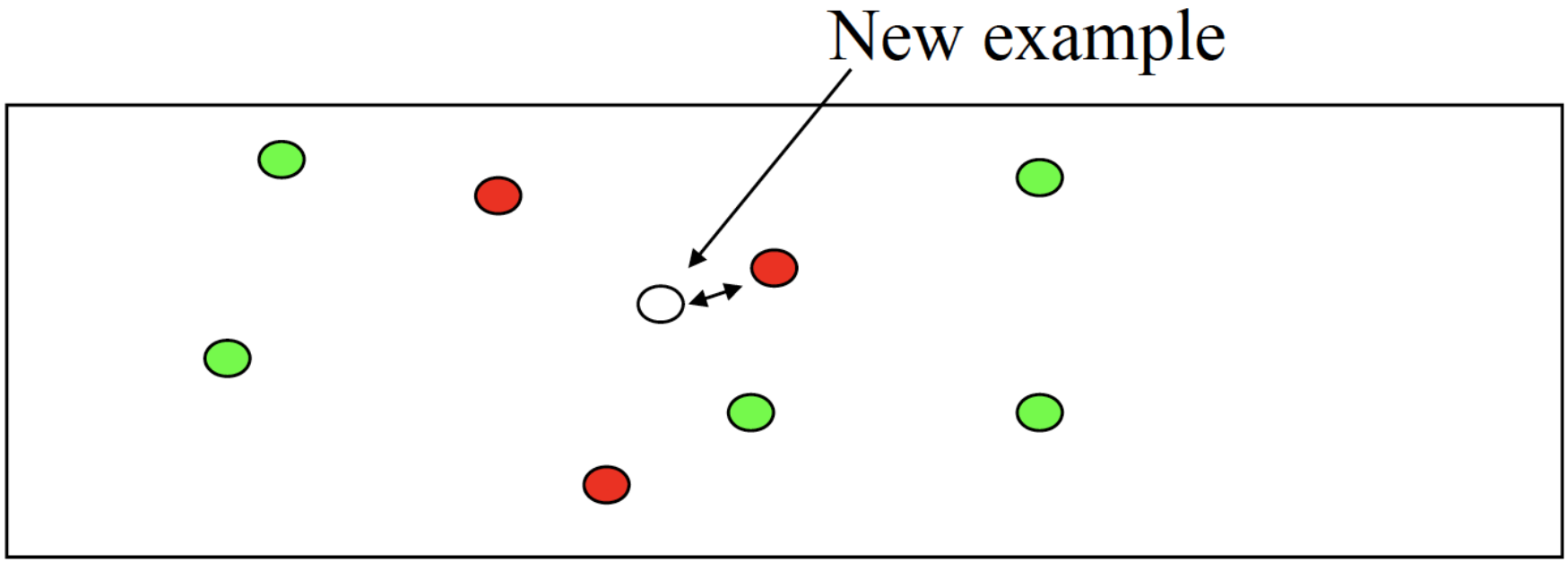
A *nearest-neighbor* classifier is one that classifies a **test vector**, $\vec{x}_0$, by finding out which **training vector** is closest:

$$h(\vec{x}_0) = y_{i^*} \quad \text{such that} \quad i^* = \underset{i=1}{\overset{n}{\operatorname{argmin}}} \|\vec{x}_0 - \vec{x}_i\|$$

where the symbol $\|\vec{x}_0 - \vec{x}_i\|$ means any valid distance function, for example, it might be Euclidean distance:

$$\|\vec{x}_0 - \vec{x}_i\| = \sqrt{\sum_{k=1}^{d} (x_{k0} - x_{ki})^2}$$

Nearest-Neighbor Classifier

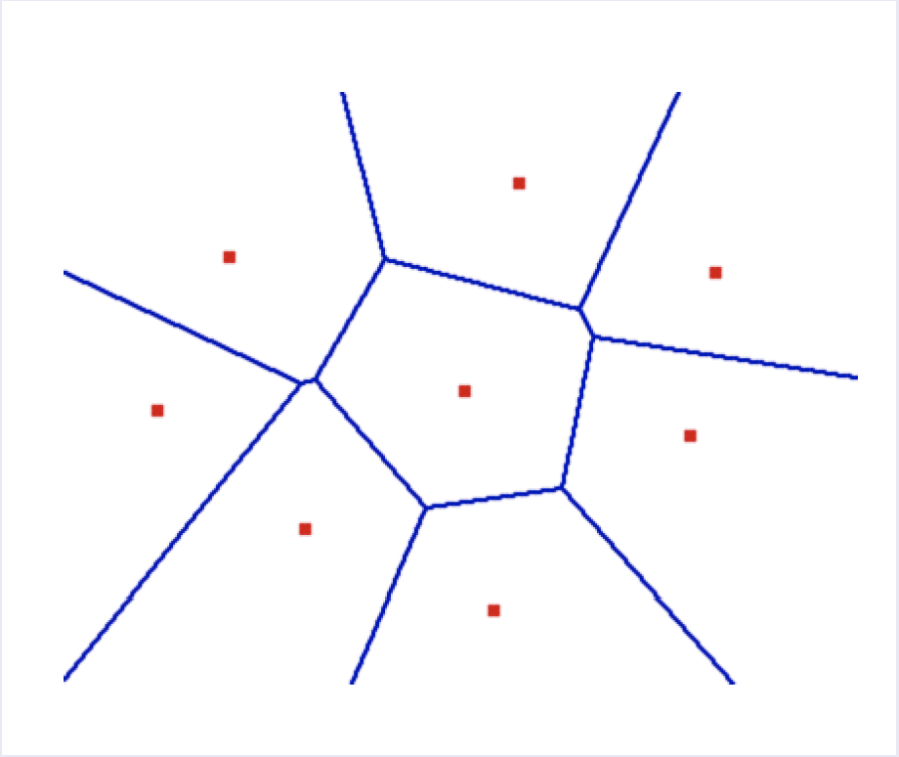


In this example, the test token (white) would be classified into class "red" because the nearest training token is from that class.

Credit: http://classes.engr.oregonstate.edu/eecs/spring2012/cs534/notes/knn.pdf

Voronoi Regions

- Given a set of points, a **Voronoi diagram** describes the areas that are nearest to any given point.
- These areas can be viewed as zones of control.

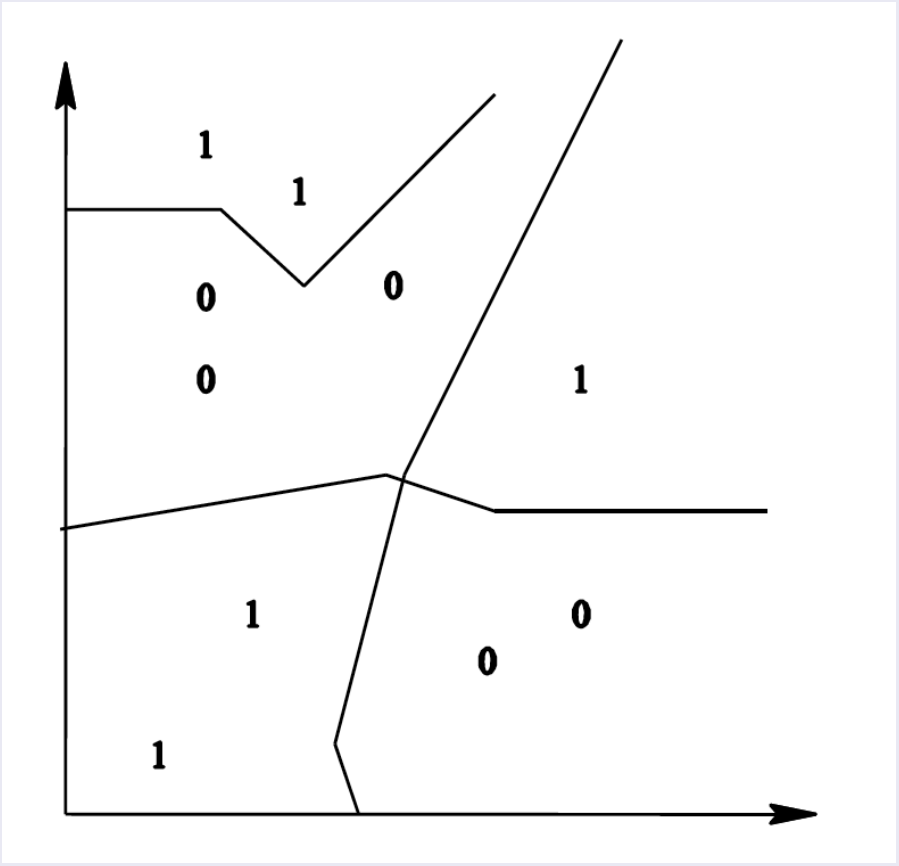


Credit:

http://classes.engr.oregonstate.edu/eecs/spring2012/cs534/notes/knn.pd

Nearest Neighbor Decision Boundaries

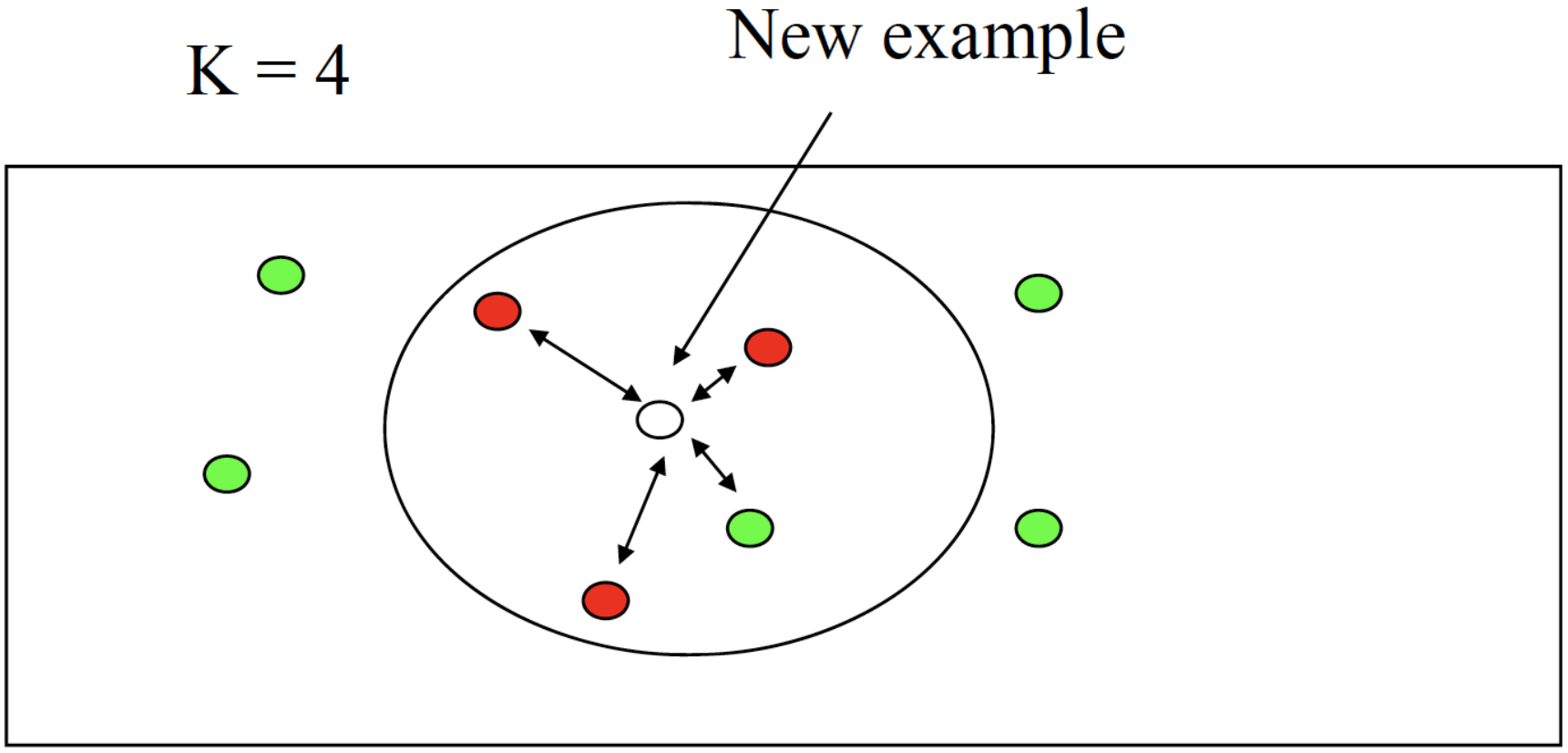
- All Voronoi regions corresponding to a token with $y_i = 0$ are in the class-0 decision region. Those corresponding to $y_i = 1$ are in the class-1 decision region.
- The boundary between these two regions is the **separatrix**.
- For a nearest neighbor classifier with Euclidean distance, the separatrix is piece-wise linear.



Credit:

http://classes.engr.oregonstate.edu/eecs/spring2012/cs534/notes/knn.p

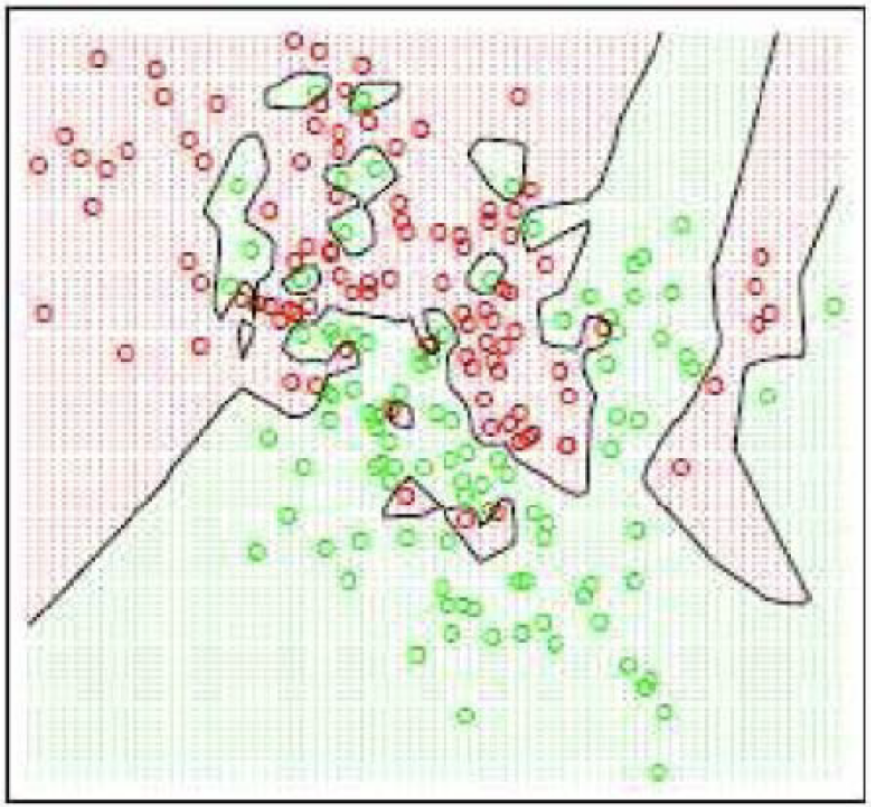
K-Nearest-Neighbors (KNN) Classifier



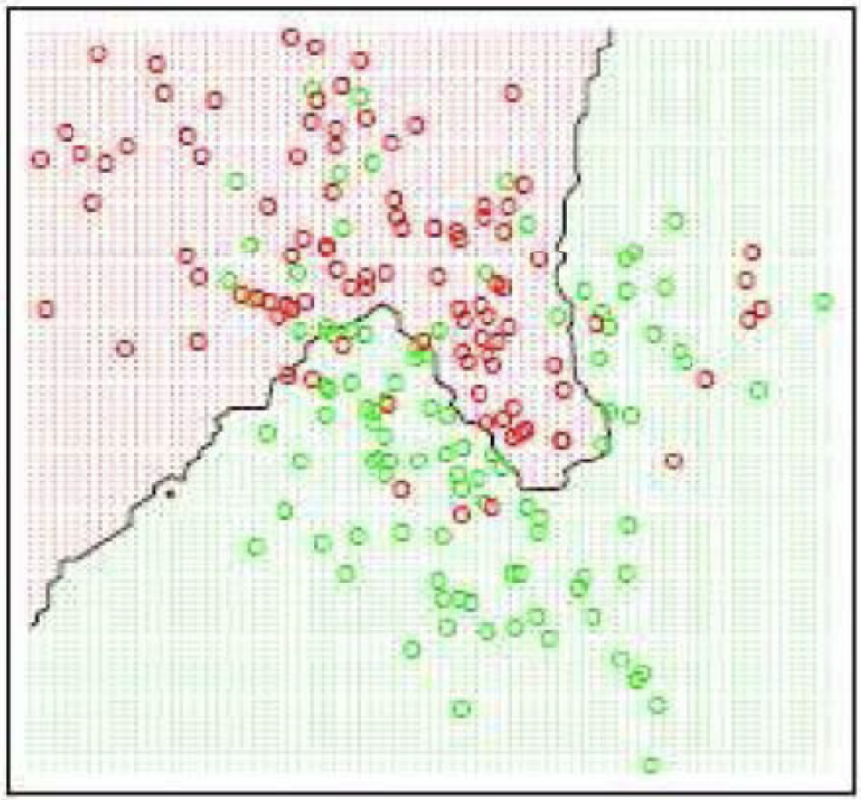
Find the K nearest neighbors and have them vote. Has a smoothing effect. This is especially good when there is noise in the reference labels. (Credit: http://classes.engr.oregonstate.edu/eecs/spring2012/cs534/notes/knn.pdf)

Effect of K

K=1



K=15



Figures from Hastie, Tibshirani and Friedman (Elements of Statistical Learning)

Advantages and Disadvantages of KNN

- Advantages:
  - **Asymptotically Optimal**: as the size of the training dataset increases, a KNN classifier is able to learn the best possible separatrix (lowest possible error rate)
  - **Training Complexity**: just store the training data, no computations required at all.
- Disadvantages:
  - **Test Complexity:** in order to classify a new vector, $\vec{x}_0$, you have to compute $\|\vec{x}_0 - \vec{x}_i\|$ for all n of the training tokens.
  - **Storage Cost:** you have to store the entire training dataset on disk. In other words, $\theta = \mathcal{D}_{tr}$, the parameter set is the training set.

Outline

Bayesian Classifier

A Bayesian classifier is one that stores a model of the full probability distribution, $p_{X,Y}(x, y)$. The classification function is then given by

$$h(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, p_{Y|X}(y|x)$$

Notice that $p_{X,Y}(x, y)$ is a probability mass function (pmf) if X and Y are both discrete, it's a probability density function (pdf) if they're both continuous, and it's a mixed distribution if, for example, Y is discrete but X is continuous.

Motivation: Minimum Probability of Error

If your model of $p_{X,Y}(x, y)$ is correct, then a Bayesian classifier minimizes the expected error rate:

$$E\left[\mathcal{L}(y, h(x))\right] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} \left[y \neq h(x)\right] p_{X,Y}(x, y) = \Pr\left\{y \neq h(x)\right\}$$

The MPE (minimum probability of error) classification rule is given by the MAP (maximum *a posteriori*) classification rule:

$$h(x) = \operatorname{argmax} p_{Y|X}(y|x)$$

The resulting lowest-possible error rate is called the **Bayes error rate**:

$$\mathcal{L}_{Bayes} = \Pr\left\{y \neq \operatorname{argmax} p(y|x)\right\}$$

Example

Suppose $p_{X,Y}(x, y)$ is given by the following table:

	x				
y	0	1	2	3	4
-1	0.03	0.2	0.05	0.2	0.02
$+1$	0.1	0.05	0.2	0.05	0.1

Then the Bayesian classifier is:

$$h(x) = \begin{cases} -1 & x \in \{1, 3\} \\ +1 & x \in \{0, 2, 4\} \end{cases}$$

Bayes Rule

Bayesian classifiers are called "Bayesian" because we use Bayes' rule to get $p(y|x)$:

$$p_{Y|X}(y|x) = \frac{p_{X,Y}(x,y)}{p_X(x)} = \frac{p_{X,Y}(x,y)}{\sum_{y \in \mathcal{Y}} p_{X,Y}(x,y)}$$

In most cases, the label set $\mathcal{Y}$ is small and finite, while the observation set $\mathcal{X}$ is a large real-valued vector space. Storing $p(x,y)$ is hard, but it's easy to store $p(y)$ (as a table), and $p(x|y)$ (in the form of a parameterized probability density, for example, a Gaussian). In this case,

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{\sum_{y \in \mathcal{Y}} p_{X|Y}(x|y)p_Y(y)}$$

The Parameters of a Bayesian classifier

The parameters of a Bayesian classifier, which you would store on disk in order to store the classifier, are therefore $\theta = \{\pi_y, \lambda_y\}$, where

- $\pi_y = p_Y(y)$ is the *a priori* class probability. It is a probability mass function, so it's necessary that $\pi_y \geq 0$ and $\sum_{y \in \mathcal{Y}} \pi_y = 1$.
- λ_y is the set of *likelihood parameters* for class y. For example, for a one-dimensional Gaussian distributed observation, $\lambda_y = \{\mu_{x|y}, \sigma_{x|y}\}$, and

$$p_{X|Y}(x|y) = \frac{1}{\sqrt{2\pi\sigma_{x|y}^2}} e^{-\frac{1}{2}\left(\frac{x-\mu_{x|y}}{\sigma_{x|y}}\right)^2}$$

The Four Probabilities of a Bayesian Classifier

- **Prior** (*a priori* class probability) is $p_Y(y)$. This is your estimate, before you have any observations, of the probability that $Y = y$.
- **Posterior** (*a posteriori* class probability) is $p_{Y|X}(y|x)$. This is your estimate of the probability that $Y = y$, after you have an observation.
- **Likelihood** is $p_{X|Y}(x|y)$.
- **Evidence** is $p_X(x)$. If you're investigating a crime scene, then $p_X(x)$ describes the complete set of evidence you have, even though most of the evidence is actually irrelevant to figuring out the value of y.

Outline

Summary

- Linear Classifier: $h(\vec{x}) = \text{argmax}_{y \in \mathcal{Y}} \left(\vec{w}_y^T \vec{x} + b_y \right)$
- Nearest Neighbor Classifier: $h(x) = y_i$ for $i = \text{argmin} \, \|x - x_i\|$.
- Bayesian Classifier: $h(x) = \text{argmax} \, p(y|x)$