CS 340 Lec. 4: K-Nearest Neighbors

AD

January 2011

K-Nearest Neighbors

- Introduction
- Choice of Metric
- Overfitting and Underfitting
- Selection of K through Cross-Validation
- Limitations

Supervised Classification

- Assume you are given some training data $\left\{\mathbf{x}^i, y^i\right\}_{i=1}^N$ where $x^i \in \mathbb{R}^d$ and $y^i \in \{1, 2, ..., C\}$.
- Given an input test data $\mathbf{x}$, you want to predict/estimate the output label y associated to $\mathbf{x}$.
- Decision trees are applicable but not very practical and difficult to fit.
- $K-$NN (K-Nearest Neighbors) is a very simple and reasonably powerful alternative.

Nearest Neighbors: The simplest supervised classifier?

- Let us introduce a distance $D : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+$ which a numerical description of how far apart two points in the input space are.
- Mathematically a distance must satisfy three conditions

Positivity	$D(\mathbf{x}, \mathbf{x}') \geq 0$ and $D(\mathbf{x}, \mathbf{x}') = 0$ if and only if $\mathbf{x} = \mathbf{x}'$
Symmetry	$D(\mathbf{x}, \mathbf{x}') = D(\mathbf{x}', \mathbf{x})$
Triangle inequality	$D(\mathbf{x}, \mathbf{x}') \leq D(\mathbf{x}, \mathbf{x}'') + D(\mathbf{x}'', \mathbf{x}')$

- For example, you can pick

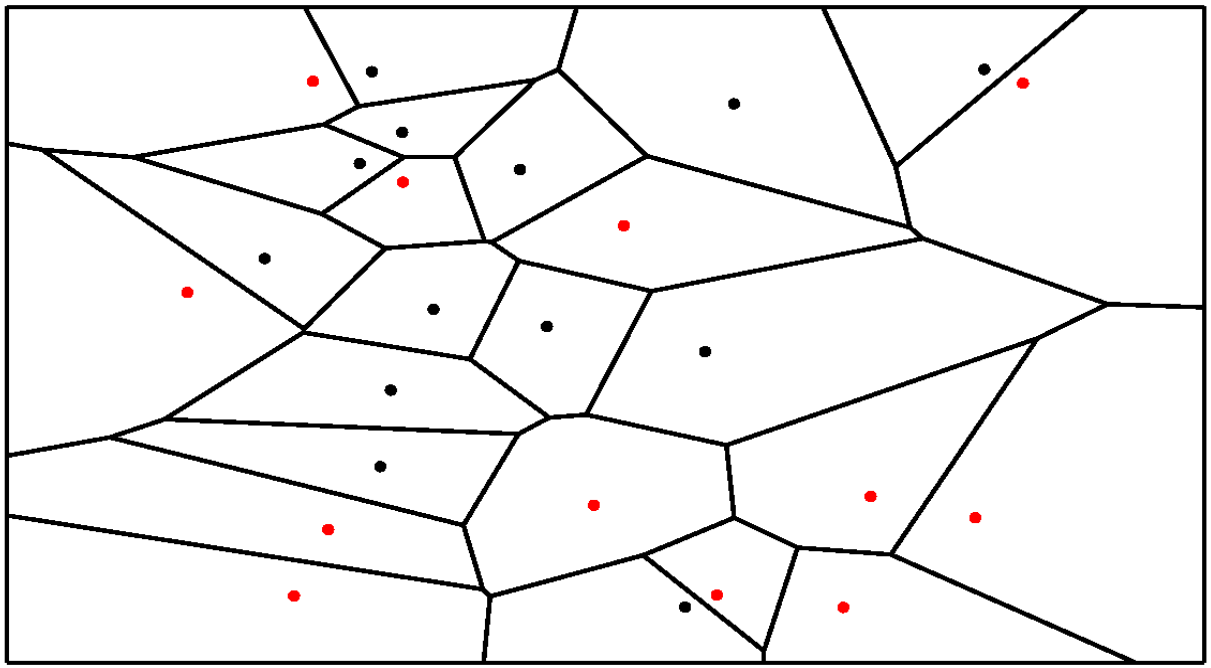
L_1 distance	$D(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{d} \lvert x_k - x'_k \rvert$
L_2 (Euclidean) distance	$D(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{k=1}^{d} (x_k - x'_k)^2}$
L_∞ distance	$D(\mathbf{x}, \mathbf{x}') = \max_{k \in \{1,2,\ldots,d\}} \lvert x_k - x'_k \rvert$

Nearest Neighbor classifier

- For $K = 1$, the 1-NN classifier ouputs looks at the point in the training set that is the nearest to the test input $\mathbf{x}$ and outputs its label; i.e.

$$\widehat{y}(\mathbf{x}) = y^k \text{ where } k = \underset{i \in \{1,2,\ldots,N\}}{\arg\min} \; D\left(\mathbf{x}, \mathbf{x}^i\right)$$

- This corresponds to a so-called Voronoi tesselation of the space.



Nearest Neighbors classifier

- For any $K \geq 1$, we look the K points in the training set that are nearest to the test input $\mathbf{x}$, counts how many members of each class are in this set, and do a majority voting.

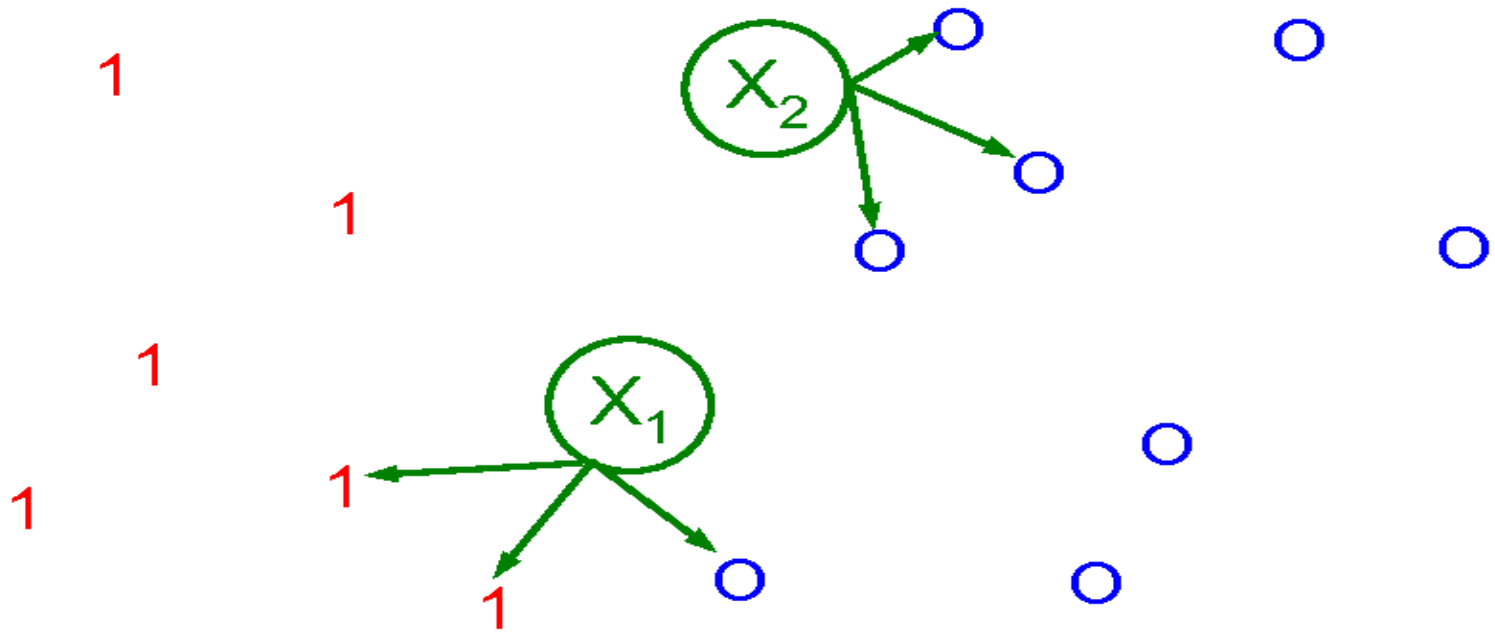


Illustration of a K-nearest neighbors classifier in $\mathbb{R}^2$ for $K = 3$ for test input $\mathbf{x}^1$ and $\mathbf{x}^2$

Practical Issues: Normalisation

- In practice, the different components of $\mathbf{x} = (x_1, x_2, ..., x_d)$ can have very different scales; e.g. $x_1 \in [-1, 1]$ and $x_2 \in [10^5, 10^9]$.
- A standard approach consists of normalizing these features; i.e. for $k = 1, ..., d$

$$\overline{x}_k = \frac{x_k - m_k}{\sigma_k}$$

where $m_k = \frac{1}{N}\sum_{i=1}^{N} x_k^i$ and $\sigma_k^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_k^i - m_k\right)^2$ are the empirical mean and variance.

- We then use K-NN on the training data $\left\{\overline{\mathbf{x}}^i, y^i\right\}_{i=1}^{N}$ with the rescaled test input $\overline{\mathbf{x}}$.
- Equivalently, this can be thought of using a different distance; e.g. if we consider say the L1 distance

$$D\left(\overline{\mathbf{x}}, \overline{\mathbf{x}}'\right) = \sum_{k=1}^{d} \left|\overline{x}_k - \overline{x}'_k\right| = \sum_{k=1}^{d} \frac{\left|x_k - x'_k\right|}{\sigma_k}.$$

"Generalizations"

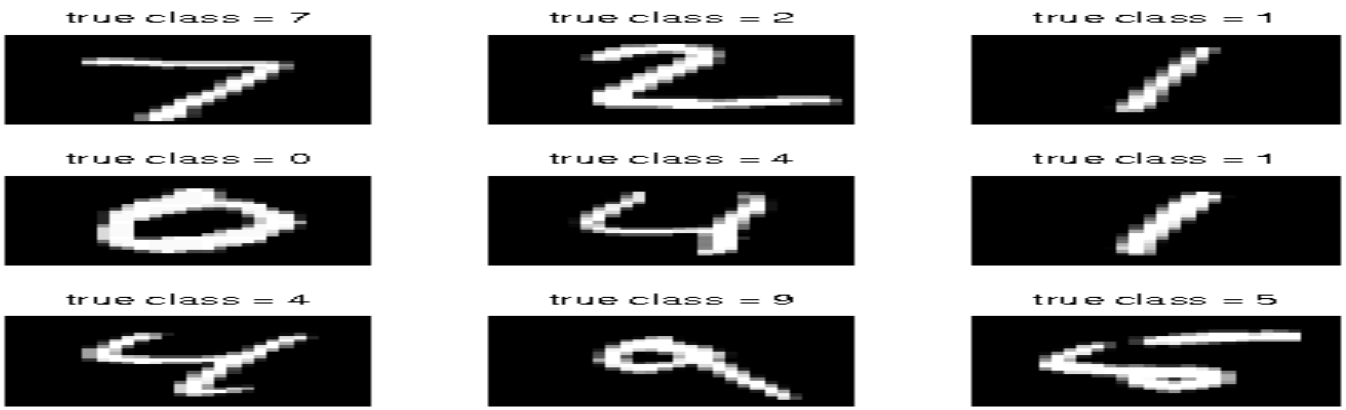
- We have considered the case where $\mathcal{X} = \mathbb{R}^d$. In numerous applications, $\mathcal{X} = \{0,1\}^d$ or $\mathcal{X}$ could be the set of directed graphs, strings etc.
- As long as we can define a valid distance, K-NN still applies.
- For example for $\mathcal{X} = \{0,1\}^d$, we can still use the L1 distance (known as Hamming)

$$D\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{k=1}^{d} \left| x_k - x'_k \right|$$

  which counts the number of entries which differ in $\mathbf{x}$ and $\mathbf{x}'$.
- Over recent years, many distance have been introduced for structured objects.

Application: Handwriting Recognition

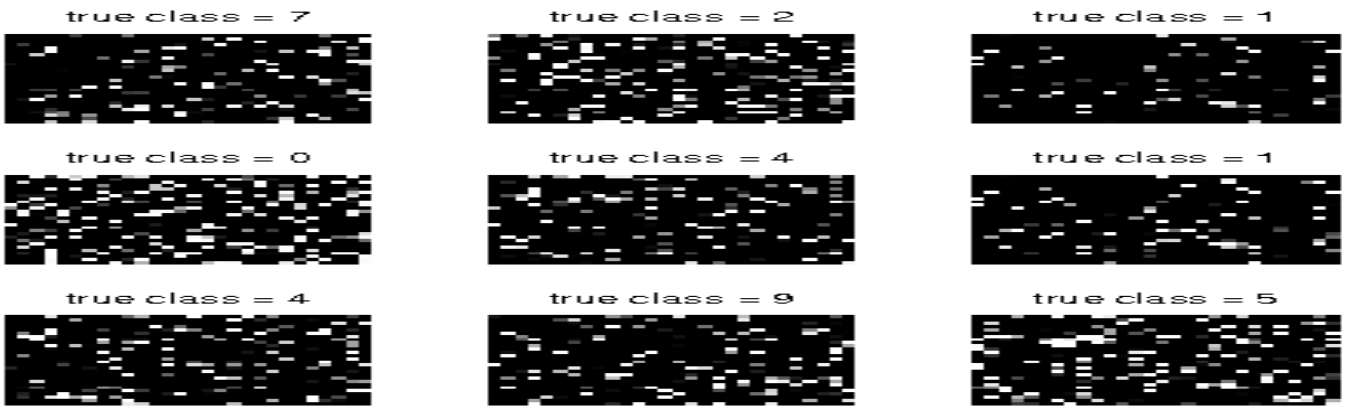
- **Objective**: recognizing isolated (i.e., non-overlapping) digits, as in ZIP or postal codes.



- **Training and Test Data**: The MNIST15 dataset contains 60,000 training images and 10,000 test images of the digits 0 to 9, as written by various people.
- **Details**: Images are 28×28 and have grayscale values in the range 0:255.

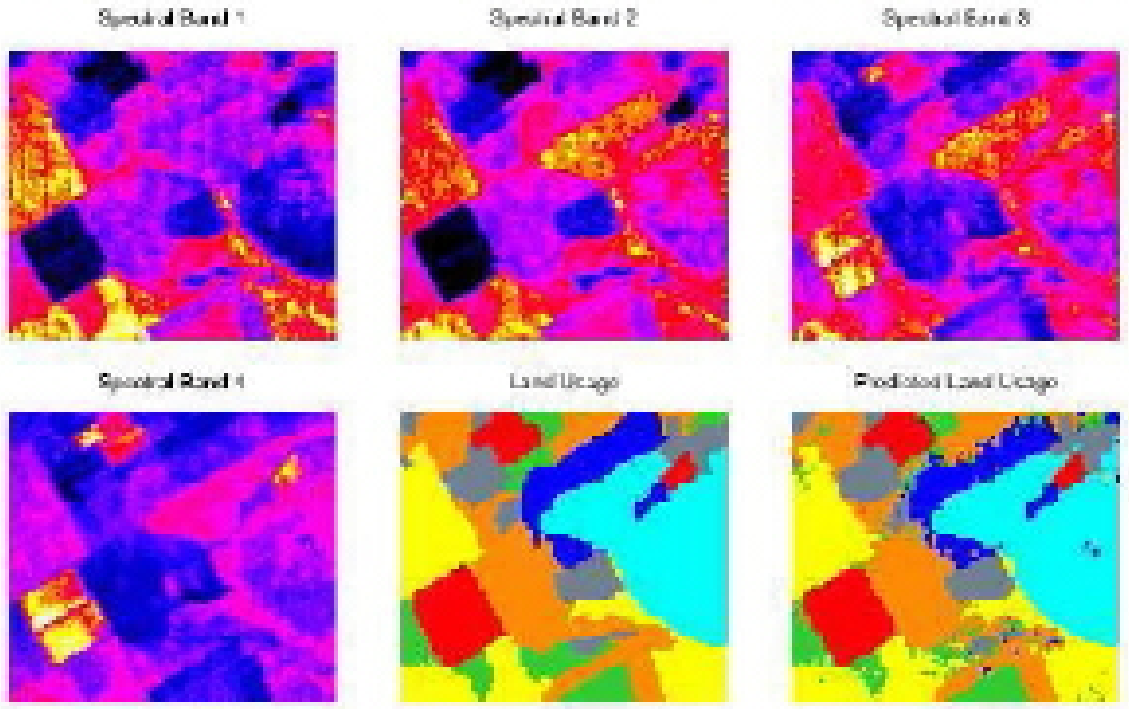
Application: Handwriting Recognition

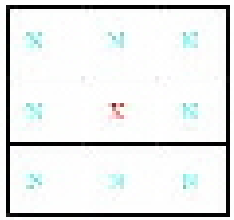
- **Results**: 1-NN obtains a miss-classification rate of only 3.09% on the test data using the Hamming distance!
- This problem might look easy to you but remember that we do not use any spatial information. The K-NN classifier would obtain exactly the same results if the training and test data were permuted as it is invariant to the order of the features.



Application: Pixel Labelling of LANDSAT Images

- LANDSAT images for an agricultural area in 4 spectral bands; manual labeling into 7 classes (red soil, cotton, vegetation, etc.);
- Output of 5-NN using each 3x3 pixel block in all 4 channels (9*4=36 dimensions).





- This approach outperformed all other methods in STATLOG project.

Overfitting

- For $K = 1$, we have no training error but are exposed to overfitting.
- Increasing K yields smoother predictions, since we average over more data.
- For $K = N$, we predict the same output whatever being $\mathbf{x}$!

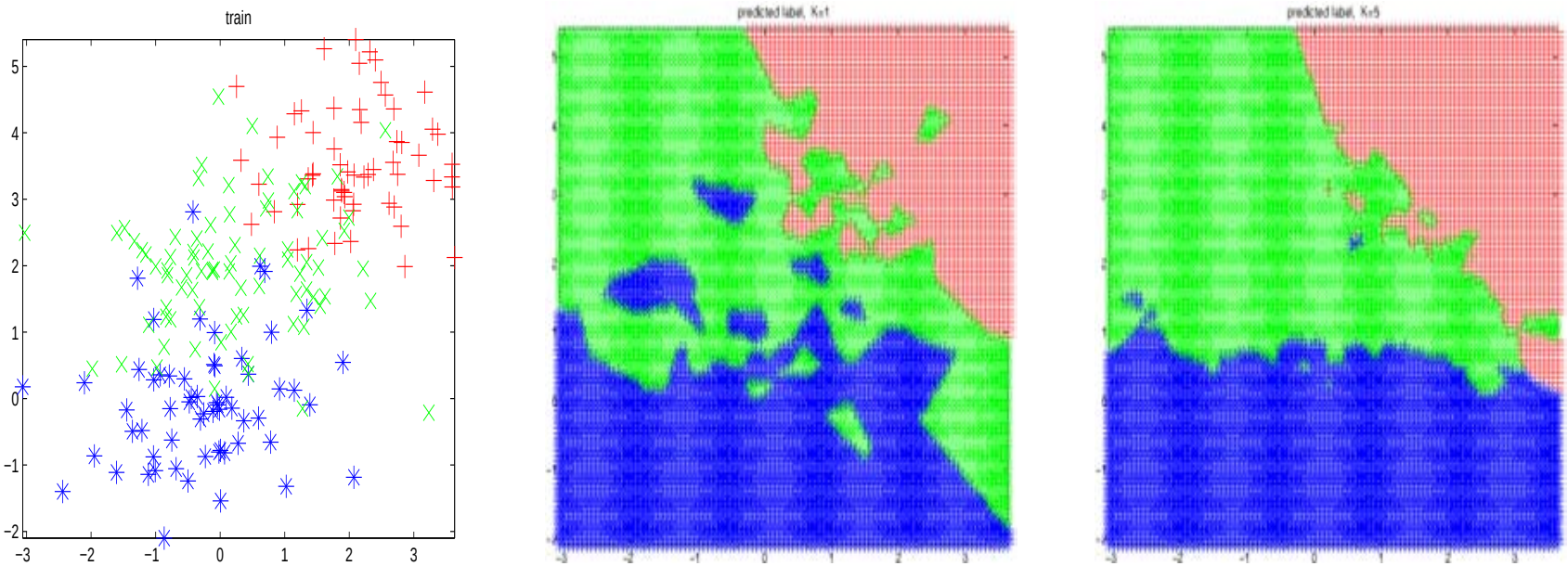


Figure: Training data (left), 1-NN (center) and 5-NN (right)

How to Select K

- We want to select K so as to obtain a small classification error on the test data but, in real-world applications, we cannot evaluate this error on the test set!
- A simple idea to evaluate the error rate consists of splitting the training data into two blocks: a block used as training data and the other block known as validation set.
- **Example**: Assume you are given $\left\{\mathbf{x}^i, y^i\right\}_{i=1}^{N}$ training data, then only $N_{train} < N$ data, say $\left\{\mathbf{x}^i, y^i\right\}_{i=1}^{N_{train}}$ are used as training data whereas the remaining $N_{valid} = N - N_{train}$ data $\left\{\mathbf{x}^i, y^i\right\}_{i=N_{train}+1}^{N}$ are used to assess the performance of the classifier using

$$\underbrace{Err}_{\text{Error rate}} = \frac{1}{N_{valid}} \sum_{i=N_{train}+1}^{N} \mathbb{I}\left(\widehat{y}\left(\mathbf{x}^i\right) \neq y^i\right).$$

- We compute Err for various values of K and select the one which minimizes Err.
- This is a very common, general and useful procedure!

Cross-Validation

- If N is small, this technique is unreliable as the model won't have enough data to train on, and we won't have enough data to make a reliable estimate of the future performance.
- A simple and popular solution to this is M-fold **cross validation** (CV). We split the training data into M folds then, for each fold $k \in \{1, 2, ..., M\}$, we train on all the folds but the k'th, and test on the k'th, in a round-robin fashion to estimate $Err = \frac{1}{M} \sum_{k=1}^{M} Err_k$. N-fold CV is called leave-one-out CV.

Figure: 5-fold cross validation

Cross-Validation for K-NN

- Illustration of the 10-fold CV for K-NN.

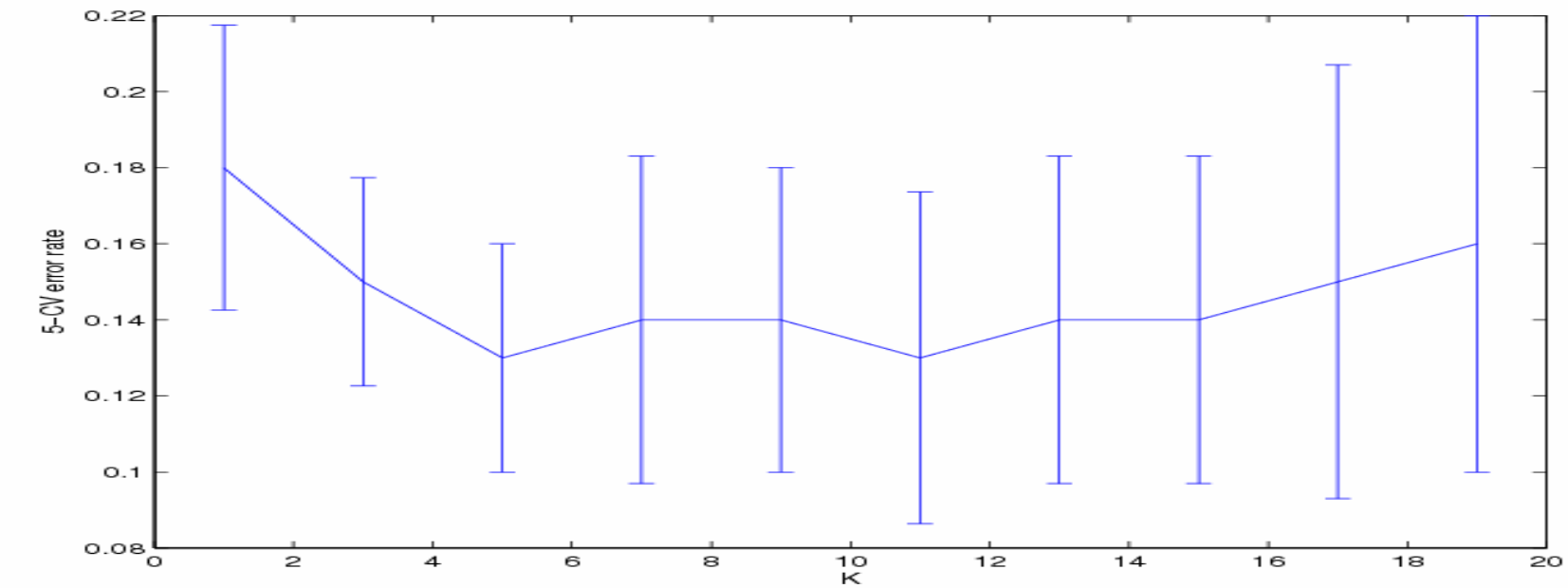


Figure: 10 fold-CV error rate as a function of K

- In this case, we would pick $K = 11$.

Problems with K-NN

- Can be slow to find nearest neighbor in high-dimensional space.
- Need to store all the training data, so takes a lot of memory.
- Need to specify the distance function.
- Does not give probabilistic output.
- Difficult to interpret.
- Curse of dimensionality...

Reducing Running Time of K-NN

- Takes $O(Nd)$ to find the exact nearest neighbor
- Use a branch and bound technique where we prune points based on their partial distances

$$D_r(\mathbf{x}, \mathbf{x}') = \sum_{k=1}^{r} (x_k - x'_k)^2 .$$

- Structure the points hierarchically into a kd-tree (does offline computation to save online computation).
- Use locality sensitive hashing (a randomized algorithm).
- Various heuristic algorithms have been proposed to prune/edit/ condense "irrelevant" points that are far from the decision boundaries.
- Later we will study sparse kernel machines that give a more principled solution to this problem.

A Probabilistic Version of K-NN

- A classification function returns a single best guess $\widehat{y}(\mathbf{x})$ of y given an input $\mathbf{x}$.
- A probabilistic classifier returns a probability distribution over outputs given an input:

$$\Pr(\widehat{y}(\mathbf{x}) = i | \mathbf{x}) \geq 0$$

$$\sum_{i=1}^{C} \Pr(\widehat{y}(\mathbf{x}) = i | \mathbf{x}) = 1.$$

- For $C = 2$ if $\Pr(\widehat{y}(\mathbf{x}) = i | \mathbf{x}) \approx 0.5$ (very uncertain), the system may choose not to classify as 0/1 and instead ask for human help.
- Useful to fuse different predictions $\widehat{y}(\mathbf{x})$ of y.

A Basic Probabilistic K-NN

- We can compute the empirical distribution over labels in the K-neighborhood; i.e. we set

$$\Pr\left(\widehat{y}\left(\mathbf{x}\right)=i\middle|\,\mathbf{x}\right)\approx\frac{1}{K}\sum_{\{j:\mathbf{x}^j\text{ is one of the }K\text{-NN of }\mathbf{x}\}}\mathbb{I}\left(y^j=i\right)$$

- **Example**: let $C=3$, $K=5$ and the 5 nearest neighbor of $\mathbf{x}$ have labels $\{2,3,3,1,2\}$ then

i	1	2	3
$\Pr\left(\widehat{y}\left(\mathbf{x}\right)=i\middle\vert\,\mathbf{x}\right)$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{2}{5}$

A Basic Probabilistic K-NN

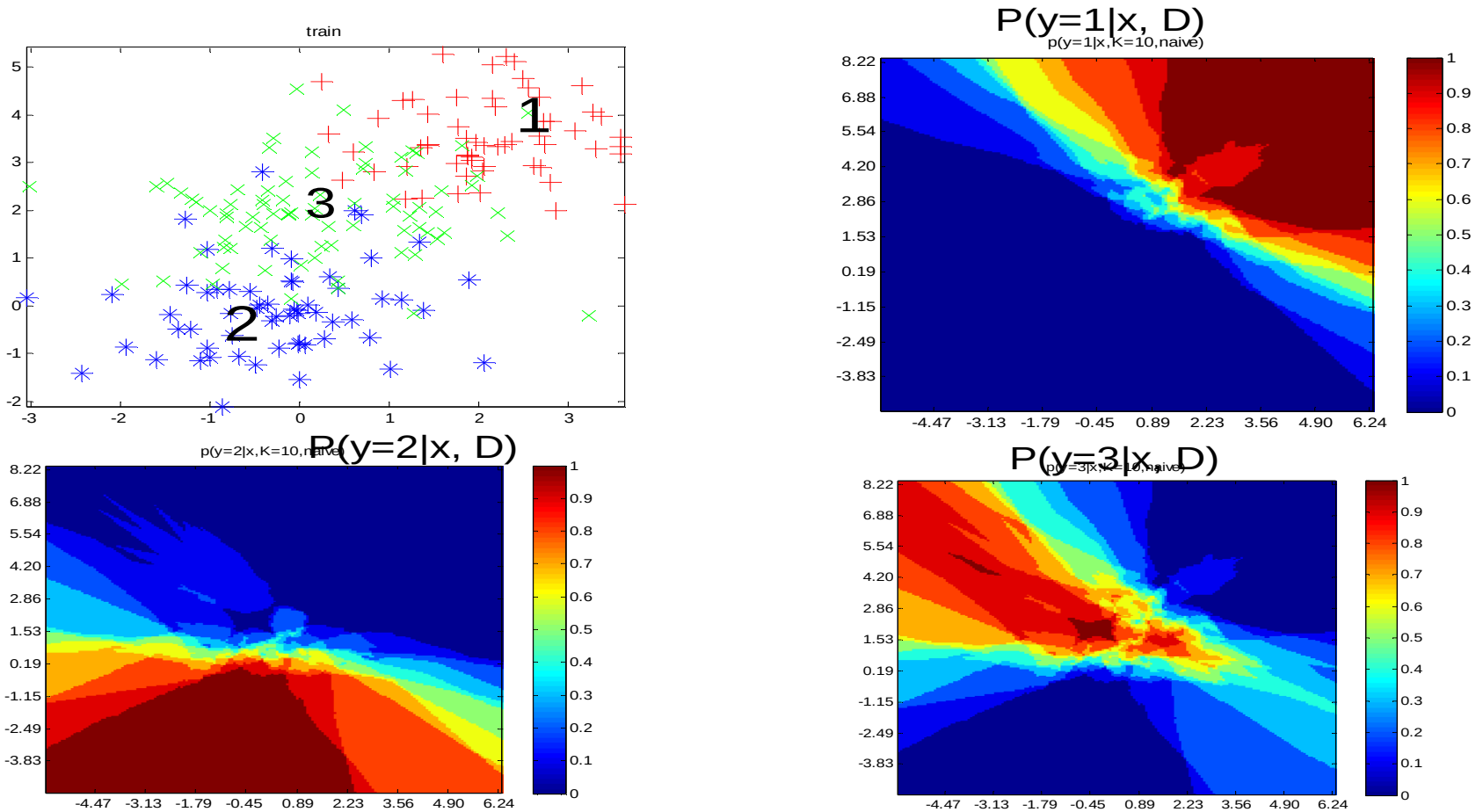


Figure: Illustration of the outpout of a probabilistic KNN classifier

Curse of Dimensionality for K-NN

- To explain the curse, consider using a KNN classifier where the inputs are uniformly distributed in the unit hypercube $[0,1]^d$.
- Suppose we want to take our decision for a test point $\mathbf{x}$ by "growing" a hypercube around $\mathbf{x}$ until it contains a desired fraction s of the training data points.
- The expected edge length of this cube will be $e_d(s) = s^{1/d}$ as $e(s)^d = s$.
- If $d = 10$ and we want to base our estimate on 1% of the data, we have $e_{10}(0.01) = 0.01^{1/10} \approx 0.63$ so we need to extend the cube 63% along each dimension around $\mathbf{x}$. Since the entire range of the data is only 1 along each dimension, the method is no longer very local, despite the name "nearest neighbor".

Curse of Dimensionality for K-NN

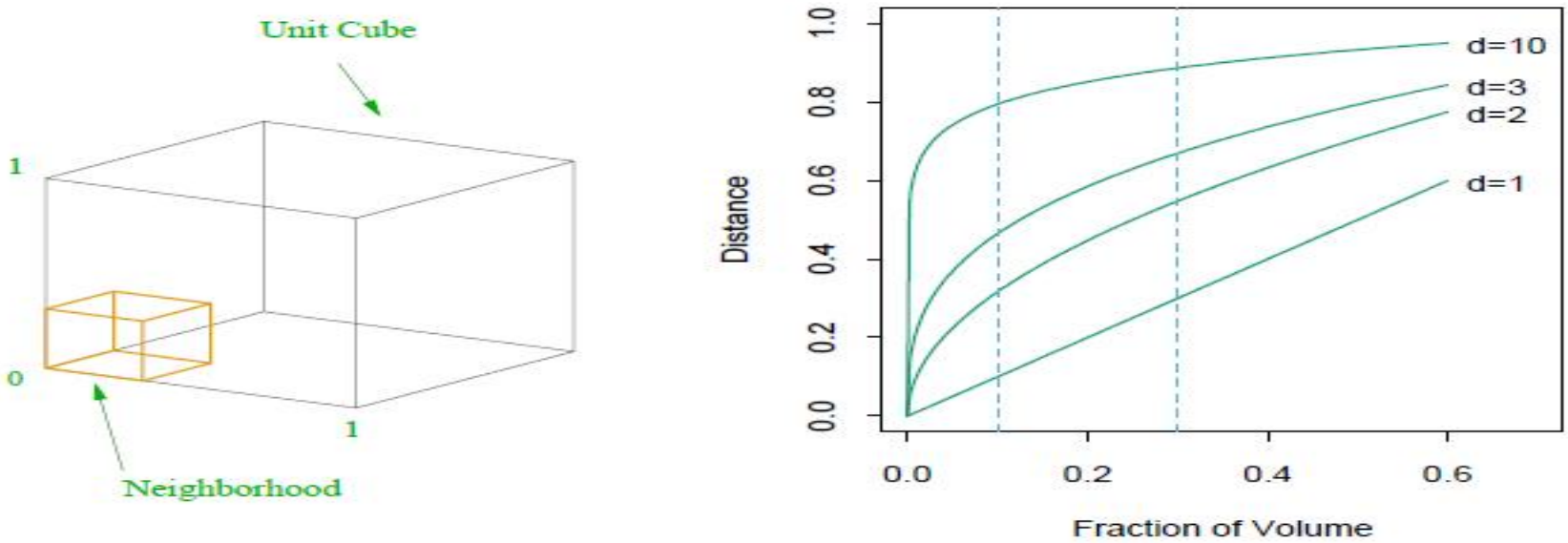


Figure: Illustration of the curse of dimensionality

Limiting the Curse of Dimensionality

- **Feature selection**: eliminate some of the "irrelevant" features x_i; e.g. the car you drive might not be a good indicator whether you are obese or not.
- **Dimension reduction**: find a low-dimensional manifold on which the data lies, and measure distance in that subspace.

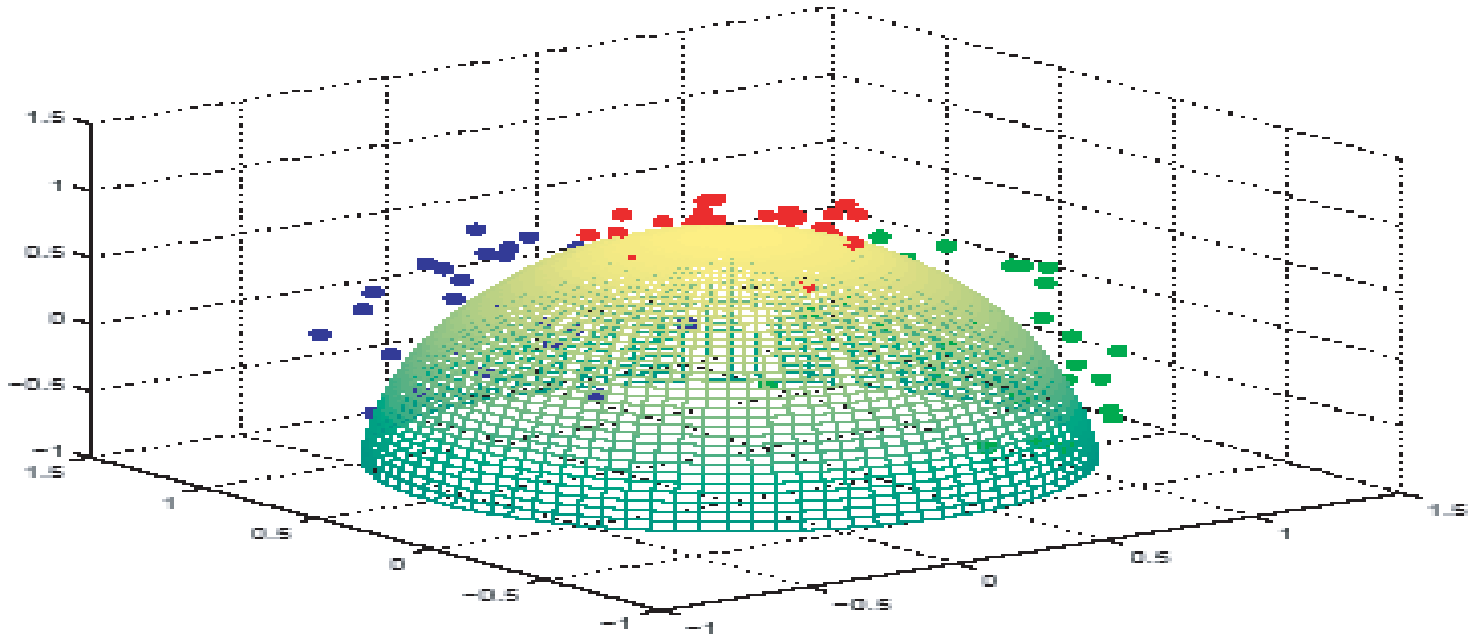


Figure: Simulated data in three classes, near the surface of a half-sphere