San José State University

Math 251: Statistical and Machine Learning Classification

Instance-based classifiers

Dr. Guangliang Chen

Outline

- kNN classification: algorithm, theory, and implementation
- Nearest local centroid (NLC) classifier
- Assignment 1

Introduction

Recall the machine learning classification problem: Given training data and their labels

$$X_{\text{train}} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}, \qquad \text{where} \quad \mathbf{x}_i \in \mathbb{R}^d,\ y_i \in \{1, \ldots, c\},$$

predict the label of a new data point $\mathbf{x}_0 \in \mathbb{R}^d$ by learning a function f from the training set

$$f : \mathbf{x}_0 \in \mathbb{R}^d \quad \longmapsto \quad y_0 \in \{1, \ldots, c\}$$

In this lecture, we introduce two instance-based classifiers: kNN and NLC.

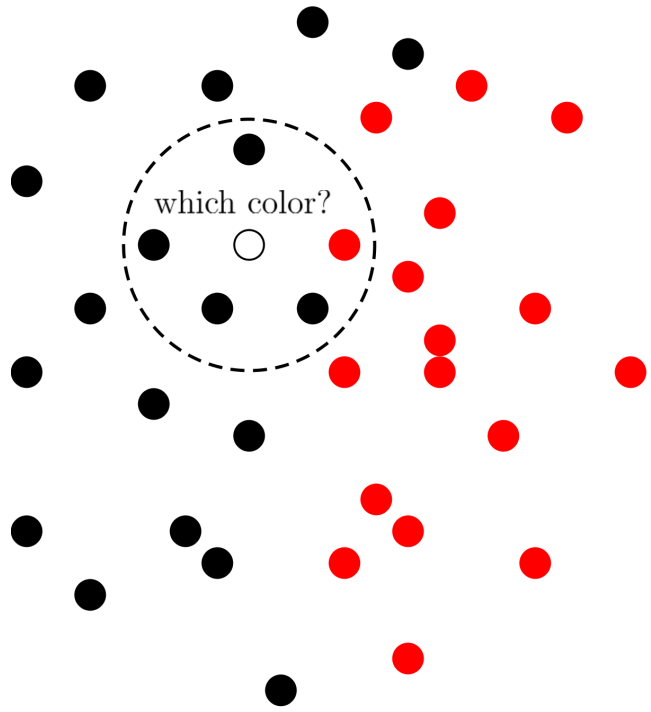
kNN classification

Main idea: Determine the label of a new point $\mathbf{x}_0$ based on those of the k closest training points through majority voting:

$$\hat{y}_0 = \arg\max_j \sum_{\mathbf{x}_i \in k\mathrm{NN}(\mathbf{x}_0)} I(y_i = j),$$

where

- $k\mathrm{NN}(\mathbf{x}_0)$: the set of k nearest neighbors of $\mathbf{x}_0$ in X_{train},
- I: indicator function



Critical questions in kNN classification

- What do we mean by "close"? $\longleftarrow$ distance metric
- How large should k be? $\longleftarrow$ model complexity
- How fast is it? $\longleftarrow$ speed
- How accurate is it? $\longleftarrow$ accuracy
- What do we know about its theoretical properties (e.g., linear/nonlinear, advantages, and limitations)? $\longleftarrow$ model and analysis

Common distance metrics

Below are different ways of measuring the distance between $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

- **Minkowski** (ℓ_p for any $p > 0$): $\|\mathbf{x} - \mathbf{y}\|_p = \left(\sum |x_i - y_i|^p\right)^{1/p}$
  - **Euclidean** ($p = 2$): $\sqrt{\sum (x_i - y_i)^2}$
  - **Manhattan/ City-block** ($p = 1$): $\sum |x_i - y_i|$
  - **Chebyshev** ($p = \infty$): $\max |x_i - y_i|$
- **Cosine of the angle**: $1 - \left\langle \frac{\mathbf{x}}{\|\mathbf{x}\|_2}, \frac{\mathbf{y}}{\|\mathbf{y}\|_2} \right\rangle = 1 - \cos\theta$
- **Correlation coefficient**: $1 - \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}}$

Complexity of the kNN classifier

Memory: $\mathcal{O}(nd)$, as it needs to store all the training data

Speed: $\mathcal{O}(n(d+k))$ for a single test point (through direct implementation):

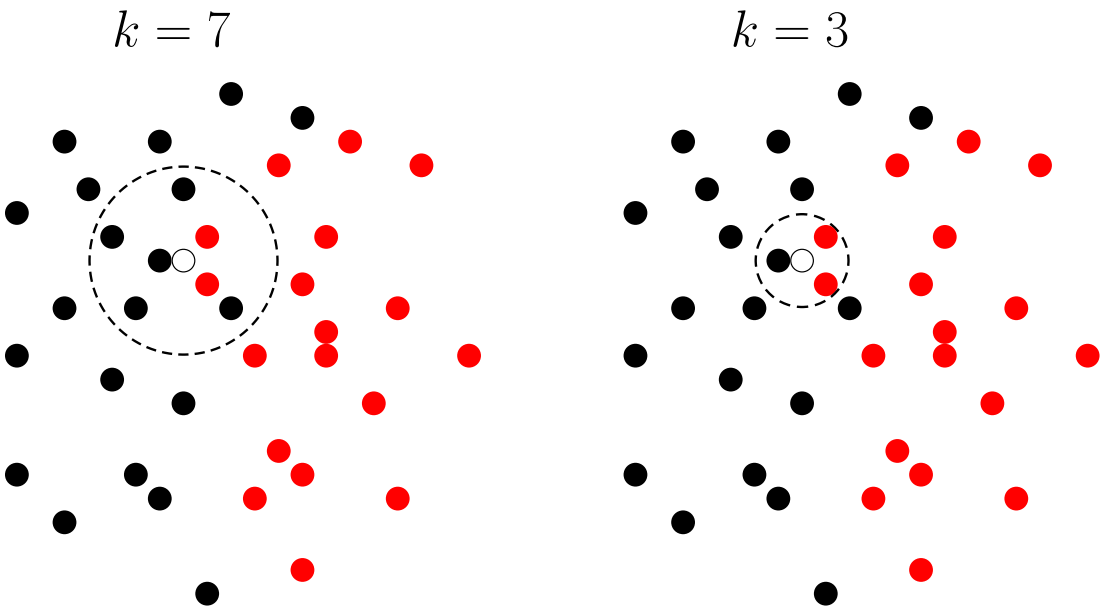
- Calculating distances: $\mathcal{O}(nd)$
- Finding the k closest training points: $\mathcal{O}(nk)$

For m test points, the overall complexity is $\mathcal{O}(mn(d+k))$.

This is a heavy computational burden for large data sets in high dimensions (e.g., for MNIST handwritten digits, $n = 60000, m = 10000, d = 784$).

How to choose the value of k

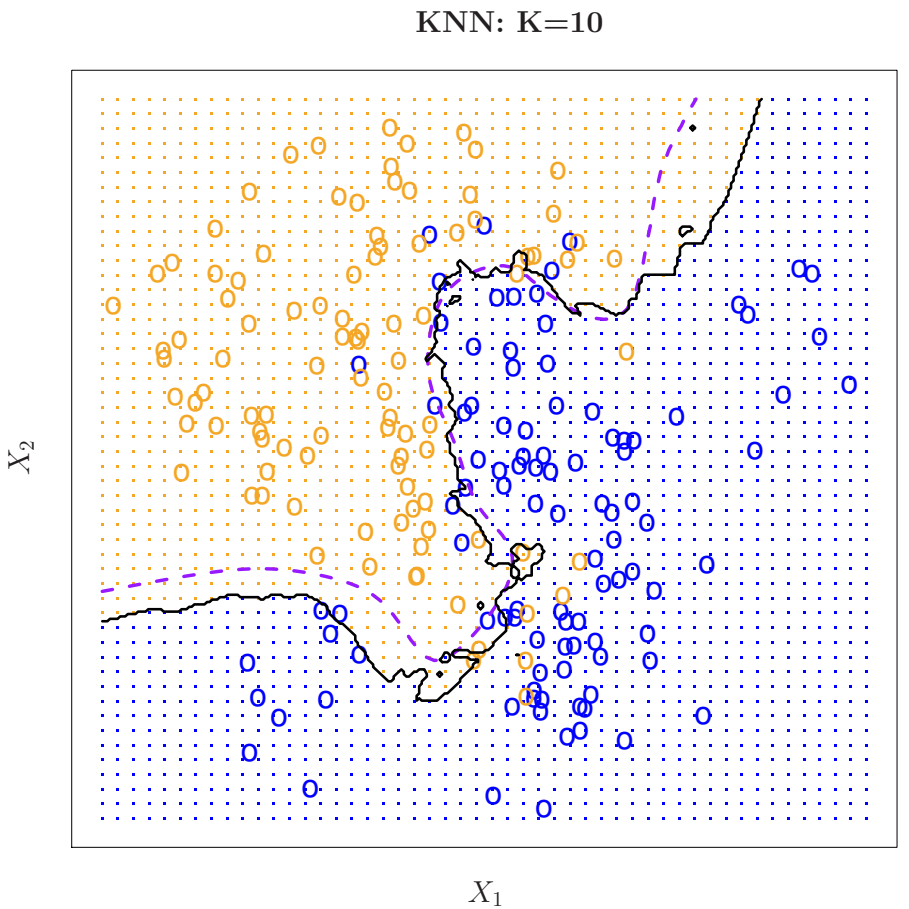
Once we have selected a proper distance metric, we can construct a kNN classifier with any specified value of k. However, different choices of k may lead to different predictions for the same test point.



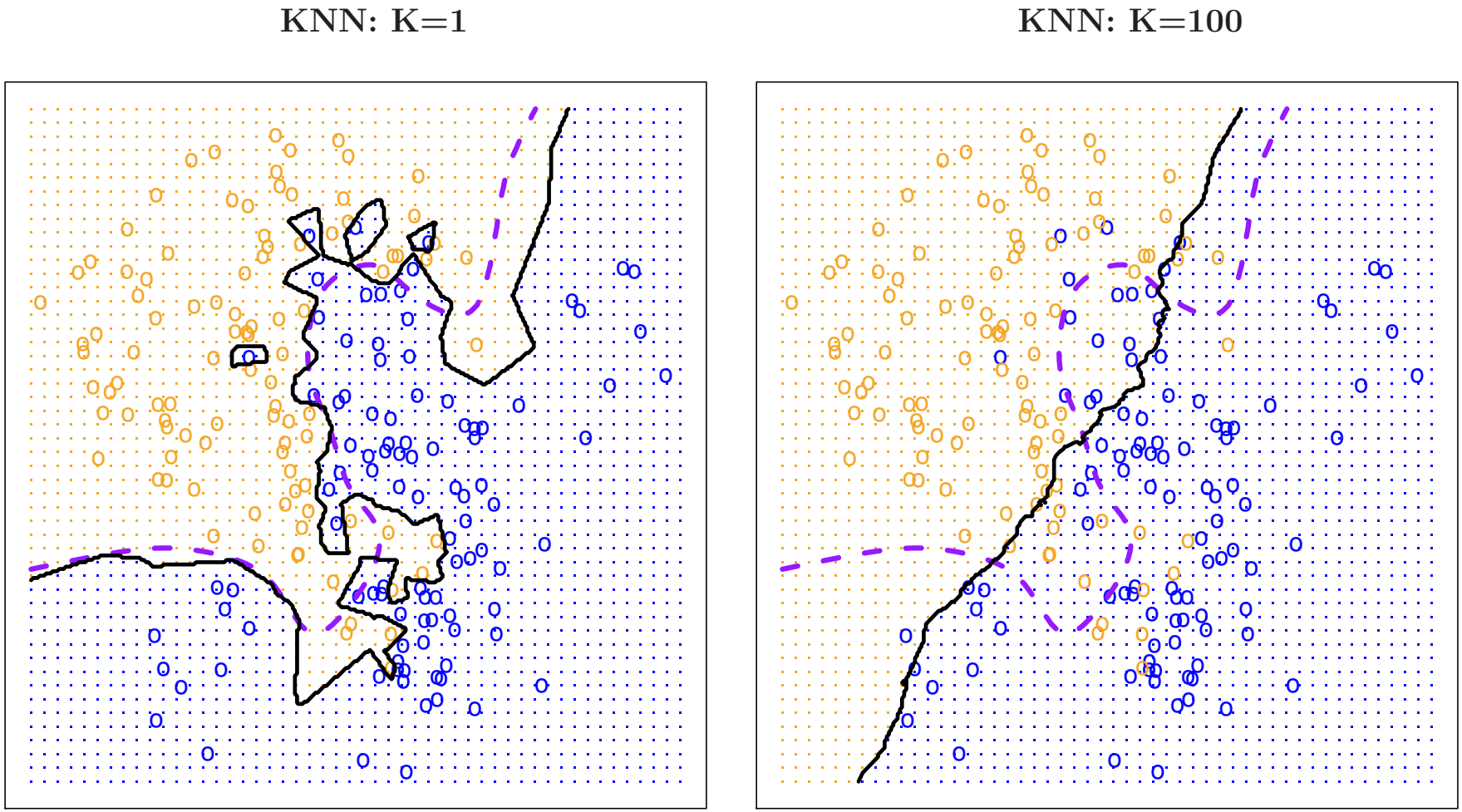
We can study the effect of the parameter k on the kNN classifier through decision maps.

The plot on the right shows a simulated data set with the **model boundary** (purple, dashed line) and the **decision boundary** (black, solid line) of the 10NN classifier.

They seem to match quite well. What about other values of k?



– James, Witten, Hastie and Tibshirani (2015), "An Introduction to Statistical Learning with Applications in R"

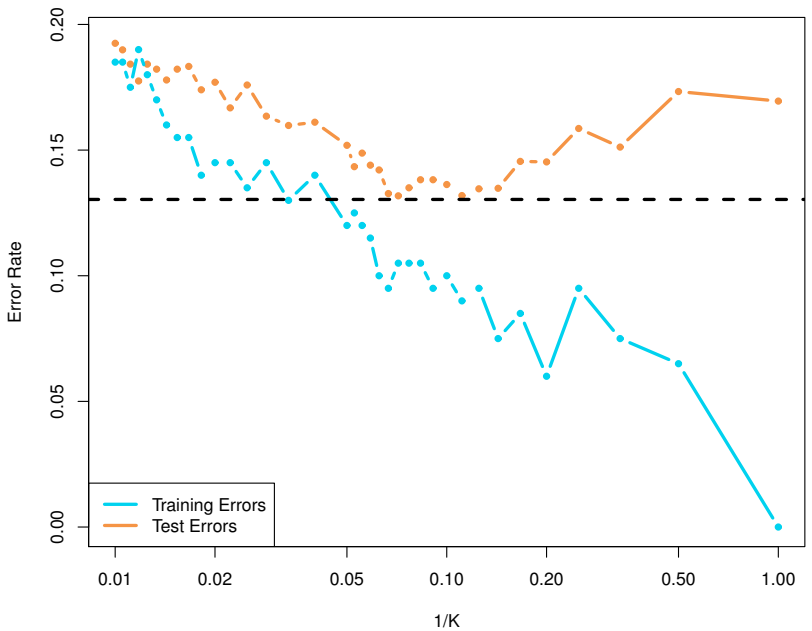

KNN: K=1
KNN: K=100

Clearly, the choice of k is crucial and different values of k lead to quite different decision boundaries.

However, the decision map is only a qualitative criterion and it is expensive to produce.

Can we come up with a quantitative criterion to compare different values of k (on the training set)?

First idea (naive): For each $k = 1, 2, \ldots,$ apply the kNN classifier to the training data and compute a training error: $e_{\text{train}}(k) = \frac{1}{n} \sum I(y_i \neq \hat{y}_i)$.



($1/k$ measures model complexity from simple to complex)

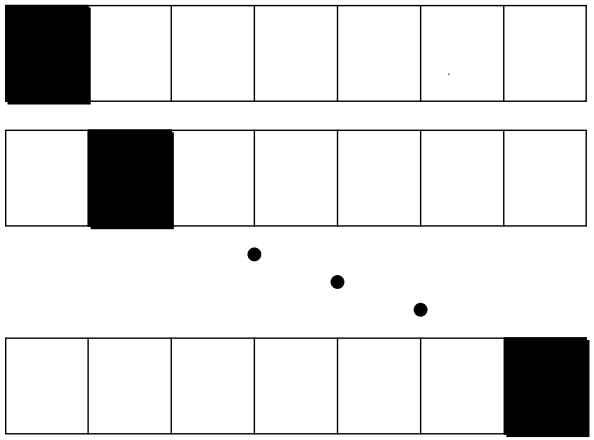
Some observations:

- $k = 1$ minimizes the training error to 0 (as it should be).
- A good range of k is 10 to 20.
- Small k overfits the data while big k underfits the data.

Second idea (good): Fix an integer f (e.g. 10). For each value of k,

- *Partition the training set randomly into f equal-sized subsets (called folds).*
- *Of the f subsets, each one is retained as validation data once and the remaining $f-1$ are used as training data*

This full procedure is called (f-fold) **cross validation (CV)**.



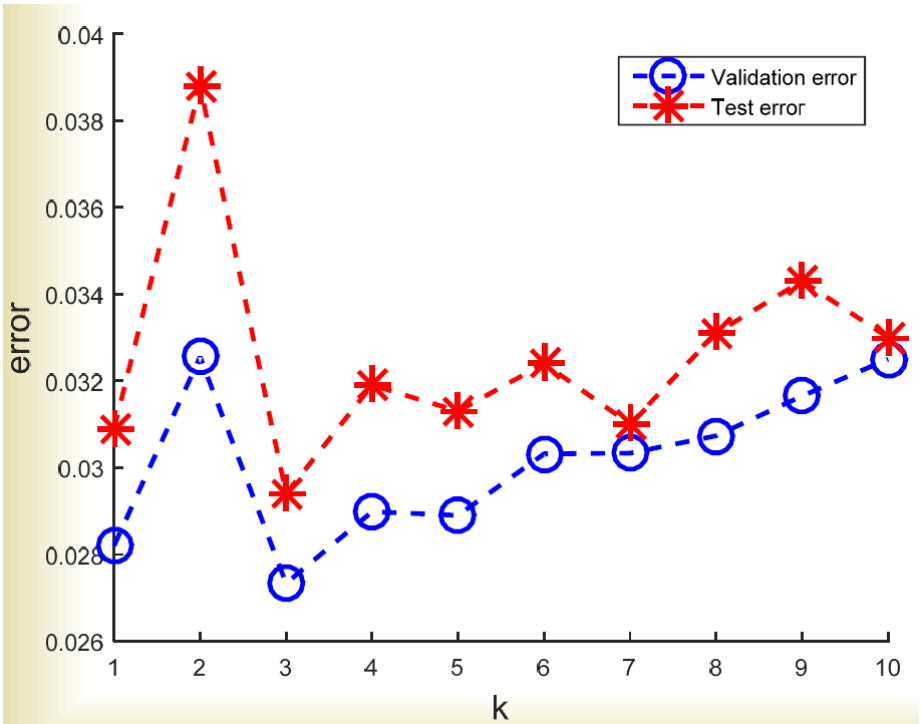
A combined validation error for each k may be computed:

$$e_{\mathrm{CV}}(k) = \frac{1}{f}\sum_{i=1}^{f} e(i).$$

It is also called resubstitution error when $f = n$ (leave-one-out CV).

Cross validation on the MNIST digits

(#folds = 6)



Remarks:

- The best k is 3, yielding the smallest validation error 0.0273 (the corresponding test error is 0.0294)
- In general, validation errors are lower than test errors.

MATLAB commands for kNN classification

% fit a kNN classification model

mdl = fitcknn(Xtr, ytr, 'NumNeighbors', 3);

% apply the model to test data for prediction

pred = predict(mdl, Xtst);

Evaluating the kNN classifier in MATLAB

```
% compute test error
testError = sum(ytst ~= pred)/numel(ytst)

% examine confusion matrix
cMat = confusionmat(ytst, pred)

% calculate 10-fold cross validation error for the above k
cvmdl = crossval(mdl,'kfold',10);
kloss = kfoldLoss(cvmdl)

% can also calculate the resubstitution error
rloss = resubLoss(mdl)
```

kNN classification with other distance metrics

mdl = fitcknn(Xtr, ytr, 'NumNeighbors', 3, 'Distance', DISTANCE);

Common choices of the string variable DISTANCE are:

'euclidean' - Euclidean distance (default)

'cityblock' - City Block distance

'chebychev' - Chebychev distance (maximum coordinate difference)

'cosine' - One minus the cosine of the included angle between observations (treated as vectors)

'correlation' - One minus the sample linear correlation between observations (treated as sequences of values).

kNN classification in Python

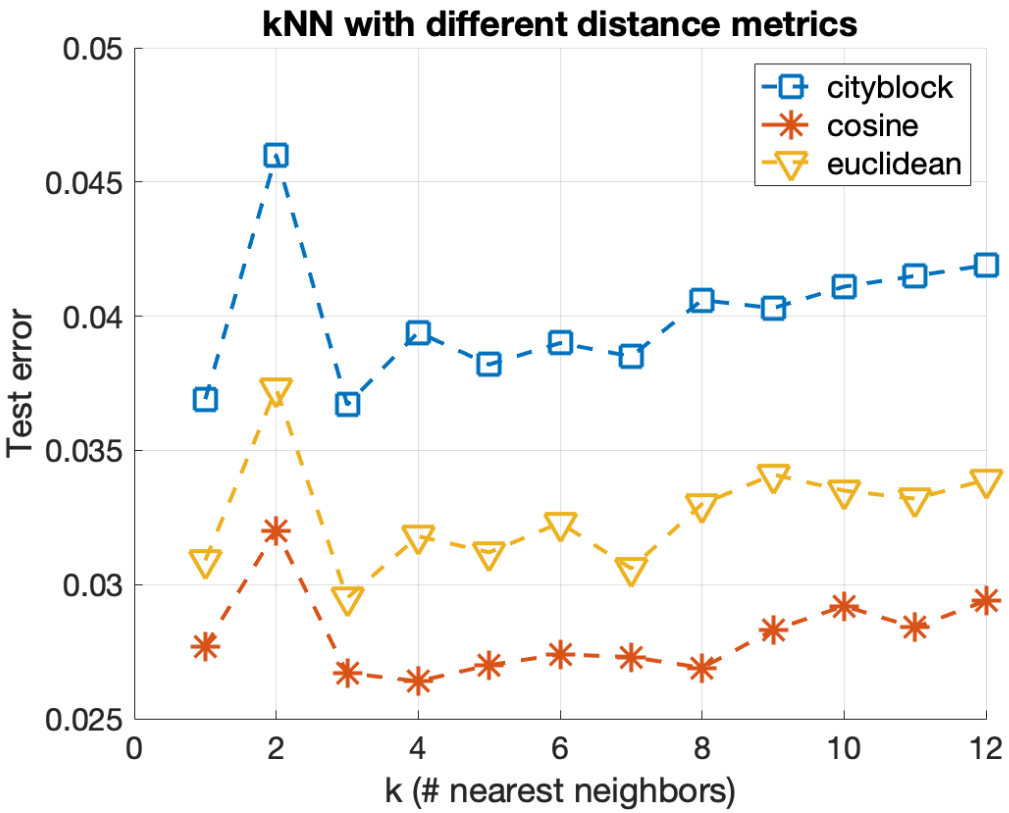
- The *scikit-learn* library[1]
- kNN in Python[2]
- Kevin Zakka's blog[3].

[1] https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification

[2] https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/

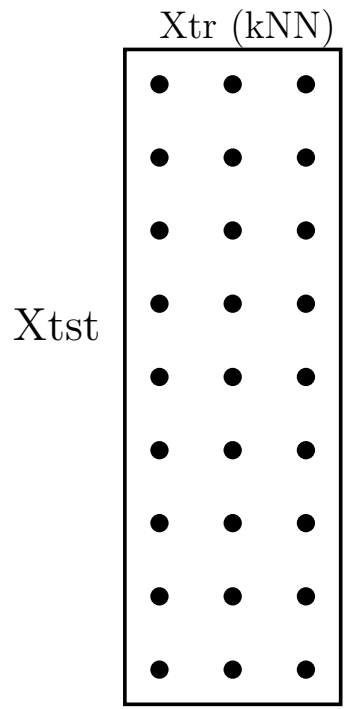
[3] https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

kNN (with different metrics) on the MNIST digits



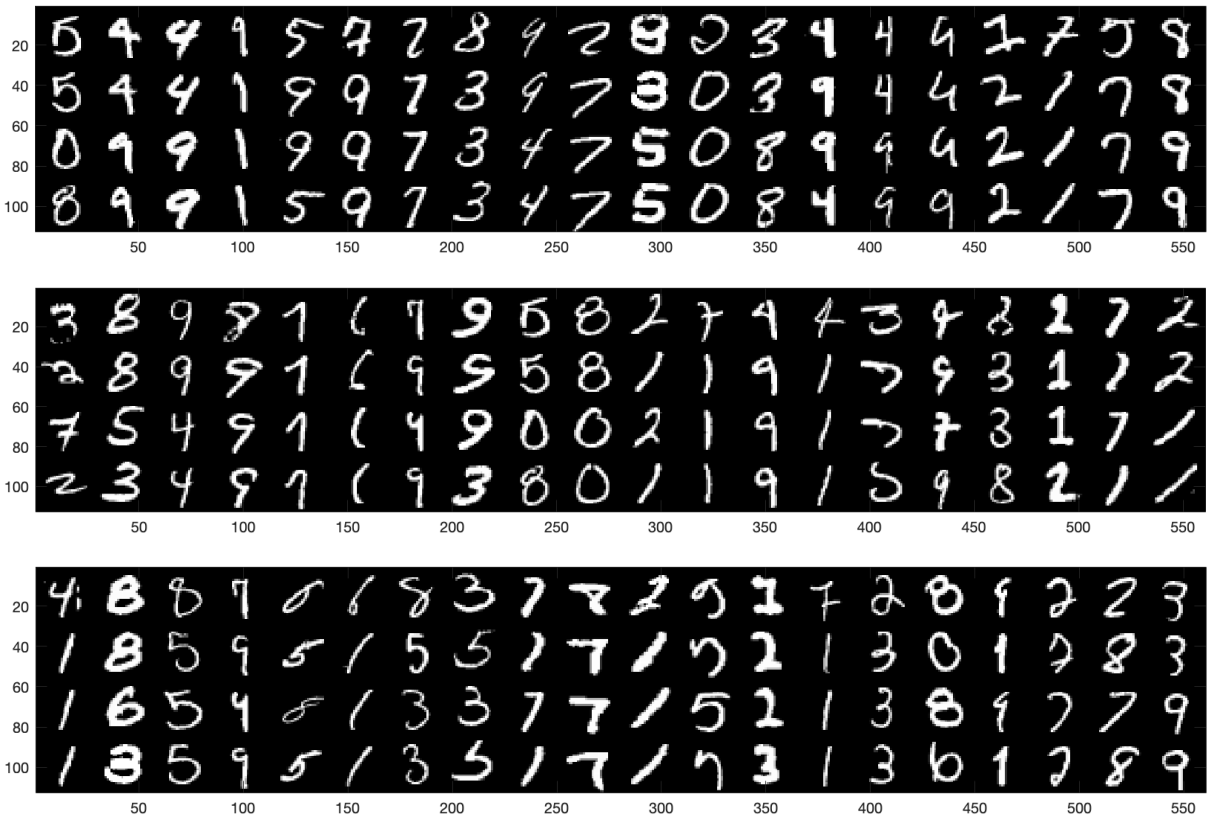
Direct implementation of kNN

knnpoints = knnsearch(Xtr, Xtst, 'K', k);
pred = mode(ytr(knnpoints), 2);



3NN errors made on MNIST

Each column shows a test image followed by the 3 nearest training examples



The statistical interpretation of kNN

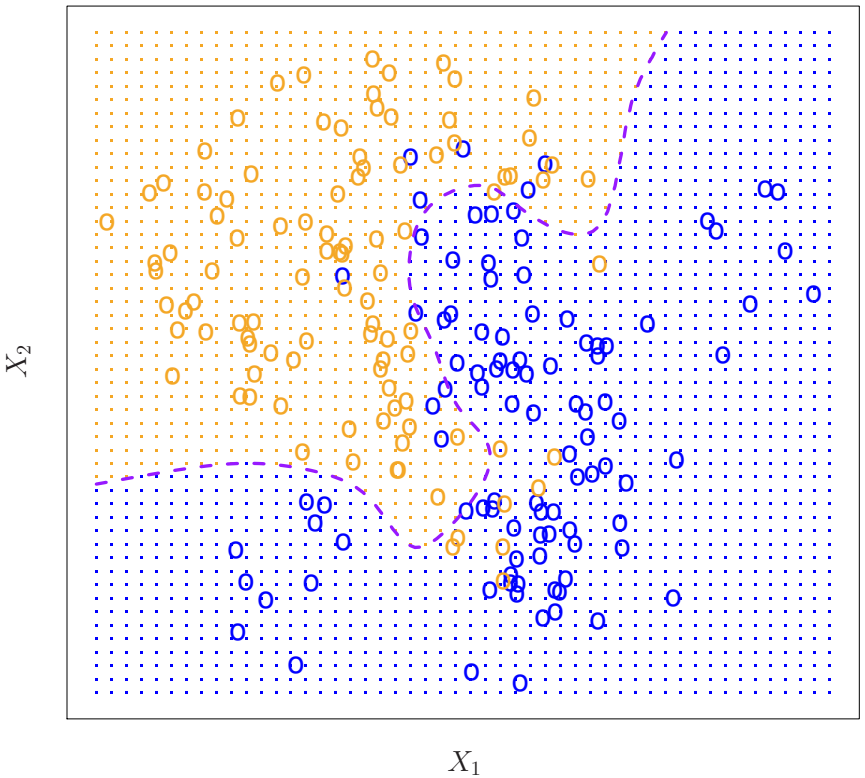
Let Y be the label of a new observation at $\mathbf{x} \in \mathbb{R}^d$. Denote by

$$p_j(\mathbf{x}) = P(Y = j \mid \mathbf{x})$$

for each $j = 1, \ldots, c$.

The $p_j(\mathbf{x})$ represent the probabilities of observing points from different classes at the fixed location $\mathbf{x}$.

See the picture on the right for an illustration (when $c = 2$).



(purple curve: $p_1(\mathbf{x}) = p_2(\mathbf{x})$)

When the sampling process is realized n times at a fixed location $\mathbf{x}$, the counts of the samples from the different classes, $N_1, \ldots, N_c$, have a multinomial distribution:

$$P(N_1 = n_1, \ldots, N_c = n_c \mid \mathbf{x}) = \binom{n}{n_1, \ldots, n_c} p_1(\mathbf{x})^{n_1} \cdots p_c(\mathbf{x})^{n_c}$$

for any $n_1, \ldots, n_c \geq 0$ with $n_1 + \cdots + n_c = n$.

Since $\mathrm{E}(N_j) = np_j(\mathbf{x})$ for each class j, an unbiased estimator of $p_j(\mathbf{x})$ is N_j/n. If we could sample many points from each location $\mathbf{x}$, then we can estimate $p_j(\mathbf{x})$ accurately.

The Bayes classifiers

If one knows the posterior probabilities $p_1(\mathbf{x}), \ldots, p_k(\mathbf{x})$ everywhere in space, then one may form a so-called **Bayes classifier**

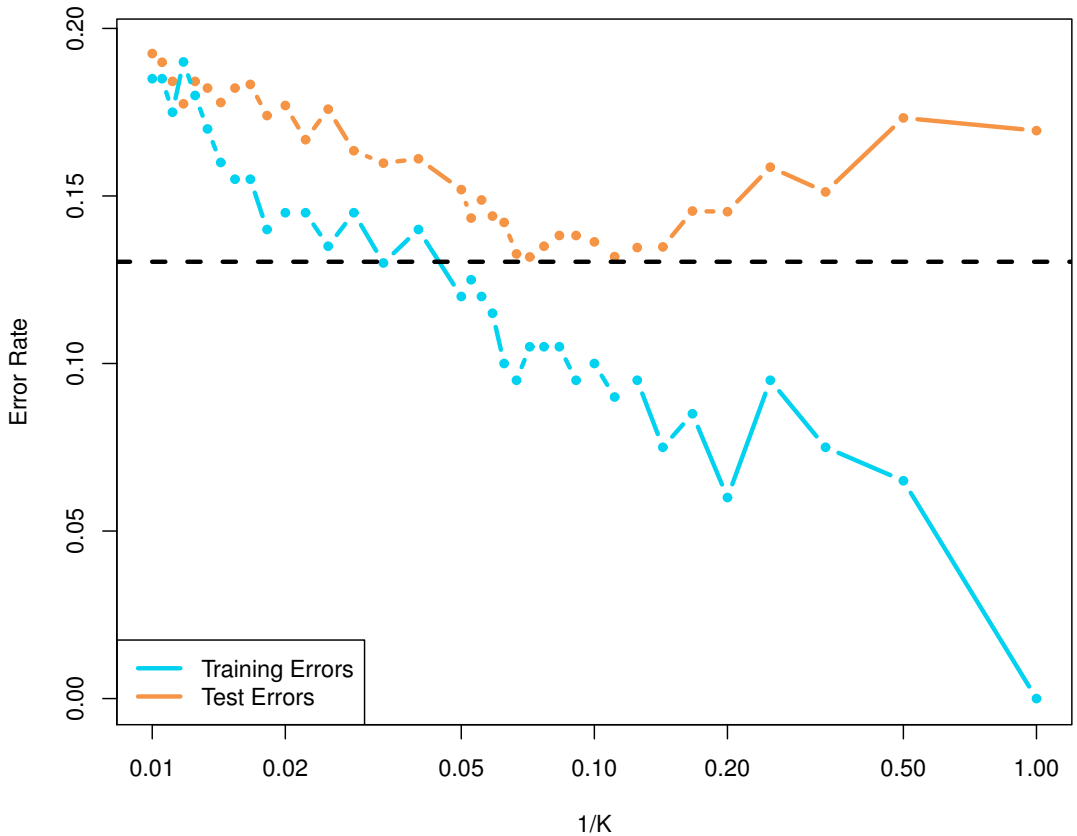
$$\hat{y} = \arg \max_{1 \le j \le k} p_j(\mathbf{x})$$

with **Bayes error rate**

$$1 - E_{\vec{X}} \left(\max_{1 \le j \le k} p_j(\vec{X}) \right)$$

The Bayes classifier is optimal in the sense that this is the lowest possible test error rate that may be achieved by a classifier.

In the previous simulation, the Bayes error rate was .1304 (indicated by the black dashed line).

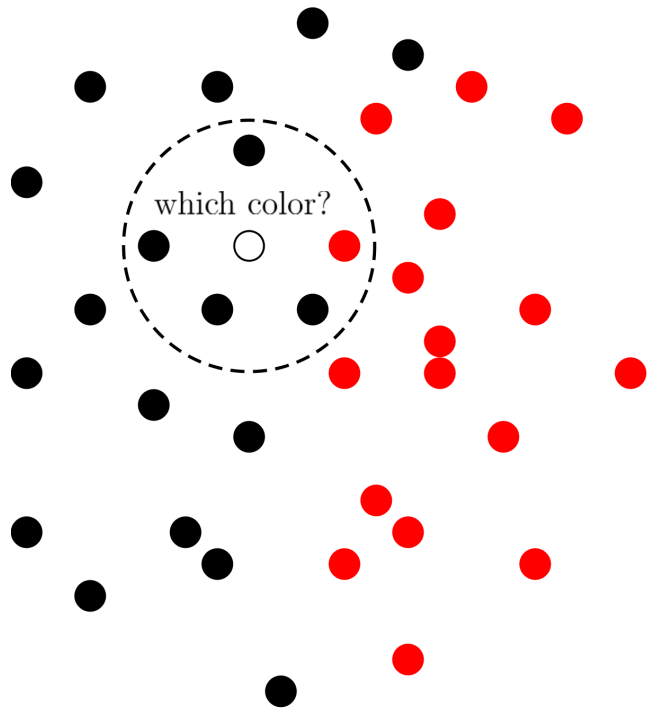


In reality, it is impossible to construct the optimal Bayes classifier since we do not know the true distribution of Y (label) at each $\mathbf{x}$ (location), nor do we have dense samples everywhere.

kNN classification can be viewed as a nonparametric way to approximate the posterior probabilities at each $\mathbf{x}$:

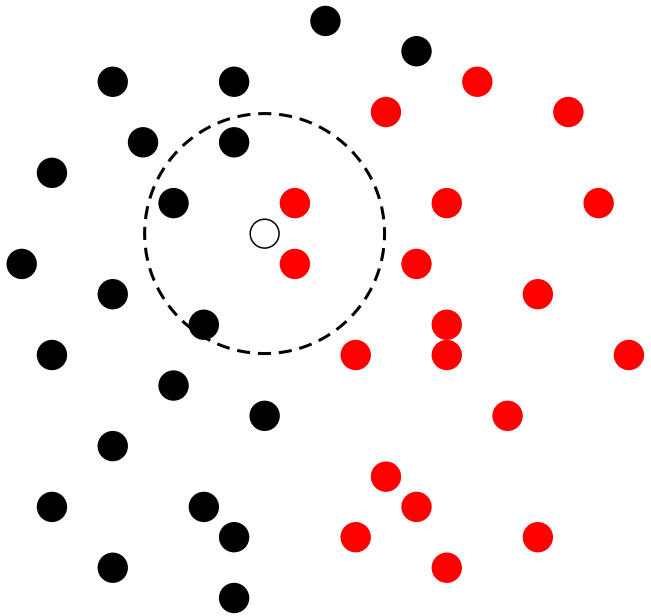
$$p_j(\mathbf{x}) \approx \frac{1}{k} \sum_{\mathbf{x}_i \in k\text{NN}(\mathbf{x})} I(y_i = j)$$

This explains why the kNN classifier works very well when given many training examples.



Weighted kNN

Consider the following example:



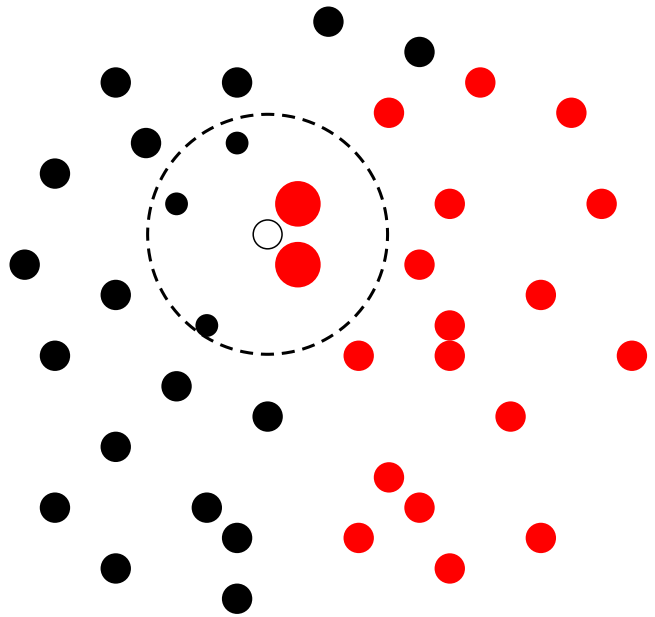
What is your prediction for the label of the test point?

Strategy: To give closer neighbors larger voting weights.

Mathematically, this is achieved as follows:

$$\hat{y} = \arg\max_j \sum_{\mathbf{x}_i \in k\text{NN}(\mathbf{x})} w_i \cdot I(y_i = j)$$

where w_i's are the weights assigned to the nearest neighbors.

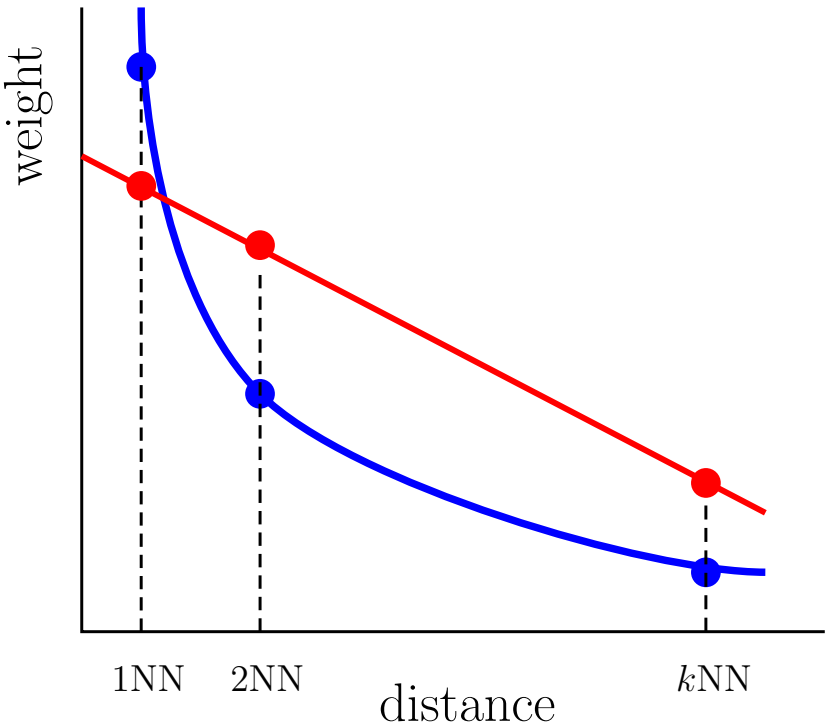


How to choose weights

Generally, the weights w_i should be a decreasing function of the distances from the test point to the nearest neighbors in the training set.

Common weights:

- Linear weights
- (Squared) inverse weights
- Exponential weights
- Normal weights



Weighted kNN in MATLAB

mdl = fitcknn(trainX, trainLabels, 'DistanceWeight', weight);

weight: A string or a function handle specifying the distance weighting function. The choices of the string are:

- '**equal**': Each neighbor gets equal weight (default).
- '**inverse**': Each neighbor gets weight $1/d$, where d is the distance between this neighbor and the point being classified.
- '**squaredinverse**': Each neighbor gets weight $1/d^2$, where d is the distance between this neighbor and the point being classified.

- **A distance weighting function specified using** @. A distance weighting function must be of the form:

  function DW = DISTWGT(D)

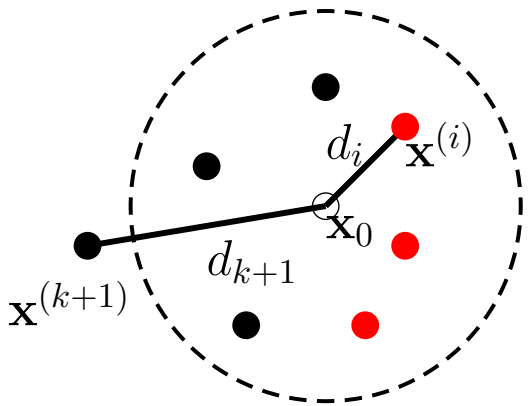
  taking as argument a matrix D and returning a matrix of distance weight DW. D and DW can only contains non-negative numerical values. DW must have the same size as D. DW(I,J) is the weight computed based on D(I,J).

  mdl = fitcknn(trainX, trainLabels, 'DistanceWeight', @DISTWGT);

Self-Implementation details: Let $\mathbf{x}_0$ be a test point with the $k+1$ nearest neighbors in the training set $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}, \mathbf{x}^{(k+1)}$. Denote $d_i = \|\mathbf{x}^{(i)} - \mathbf{x}_0\|$ for $1 \leq i \leq k+1$. Note that d_{k+1} is only used to assist in choosing weights for the first k nearest neighbors ($\mathbf{x}^{(k+1)}$ does not vote).

Recommended formulas:

- Linear weights: $w_i = \frac{d_{k+1} - d_i}{d_{k+1} - d_1}, 1 \leq i \leq k$
- (Squared) inverse weights: $w_i = \frac{1}{\frac{d_i}{d_{k+1}} + 1}$
- Exponential weights: $w_i = e^{-d_i/d_{k+1}}$
- Normal weights: $w_i = e^{-d_i^2/d_{k+1}^2}$

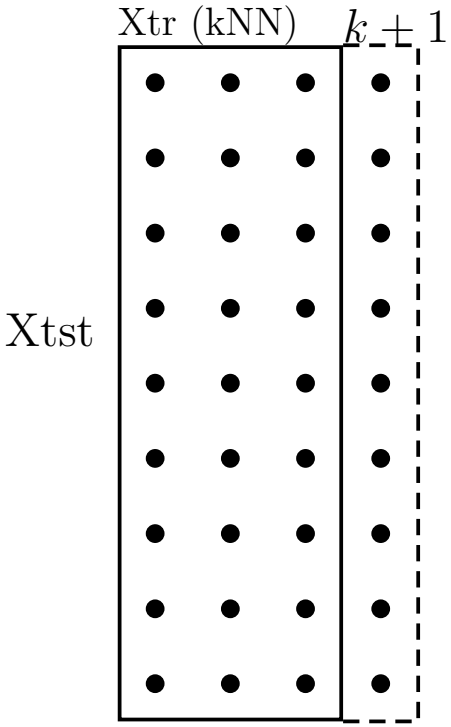


Direct implementation of weighted kNN

[knnpoints,knndists] = knnsearch(Xtr, Xtst, 'K', k+1);

knnlabels = ytr(knnpoints);

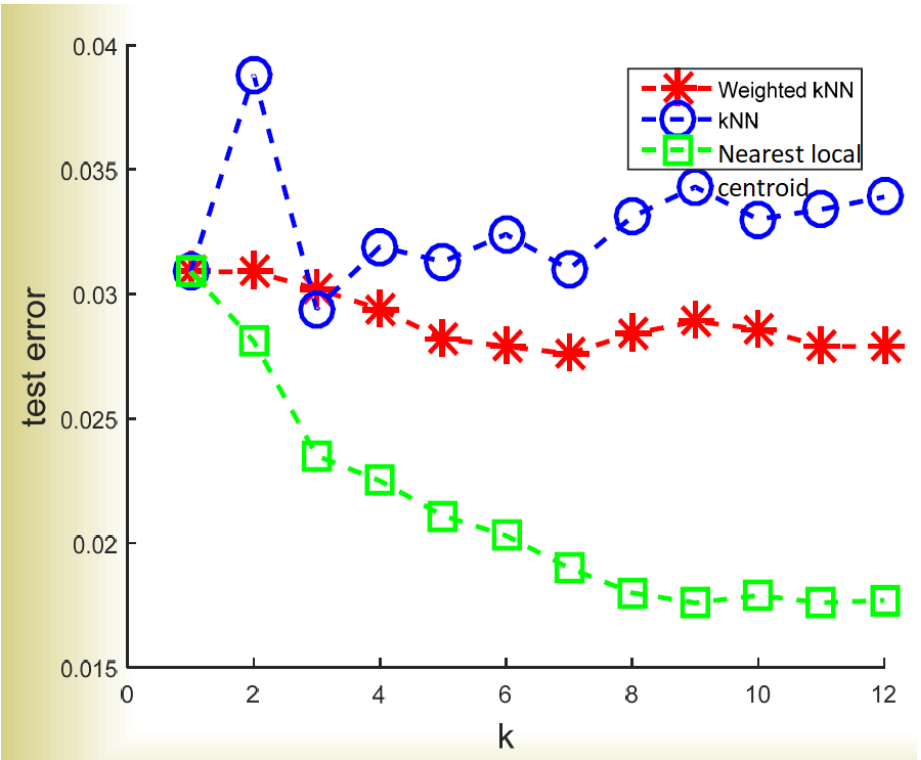
% linear weights
W = (knndists(:,end) - knndists(:,1:k)) ./ ...
(knndists(:,end) - knndists(:,1));



Weighted kNN on MNIST

We fix $k = 3$ and experiment with the following kinds of weights:

- Plain (no weights): 2.94%
- Inverse: 2.83%
- Squared inverse: 2.83%
- Linear: 3.02%



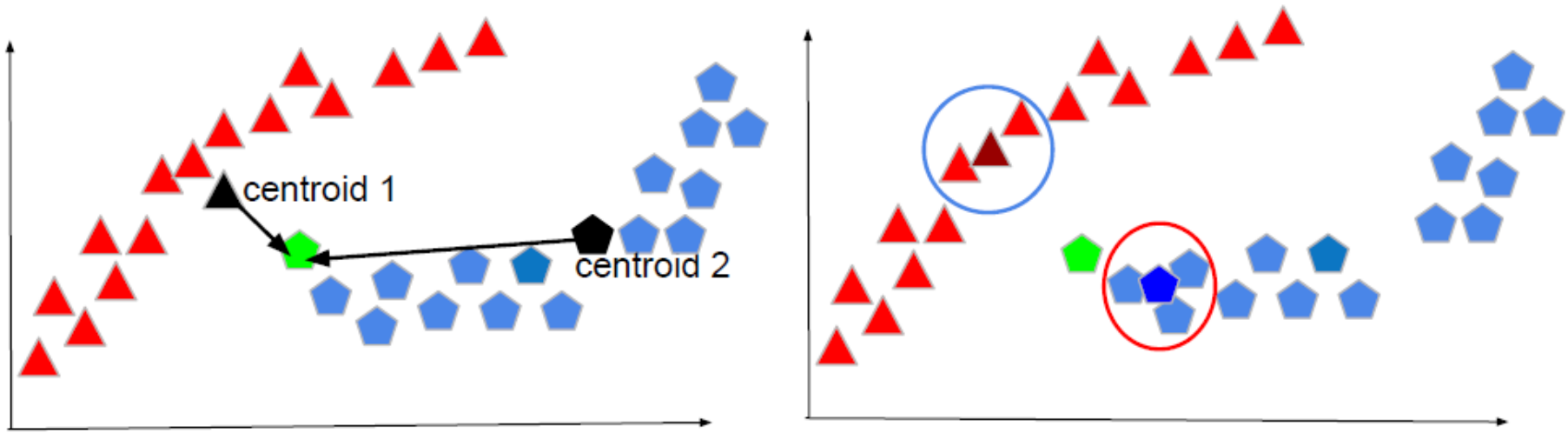
Smaller test error rates may be achieved for larger k.

The minimum error rate with linear weights is 2.76% (when $k = 7$).

Comments on kNN classification

- Instance-based learning (or lazy learning)
- Model free (nonparametric)
- Simple to implement, but powerful
- Localized and nonlinear (trying to approximate the Bayes classifier)
- Algorithmic complexity only depends nearest neighbors search (memory and CPU cost)
- The choice of k is critical (need to use cross validation)
- Lots of variations in the literature

Fixing the nearest centroid classifier

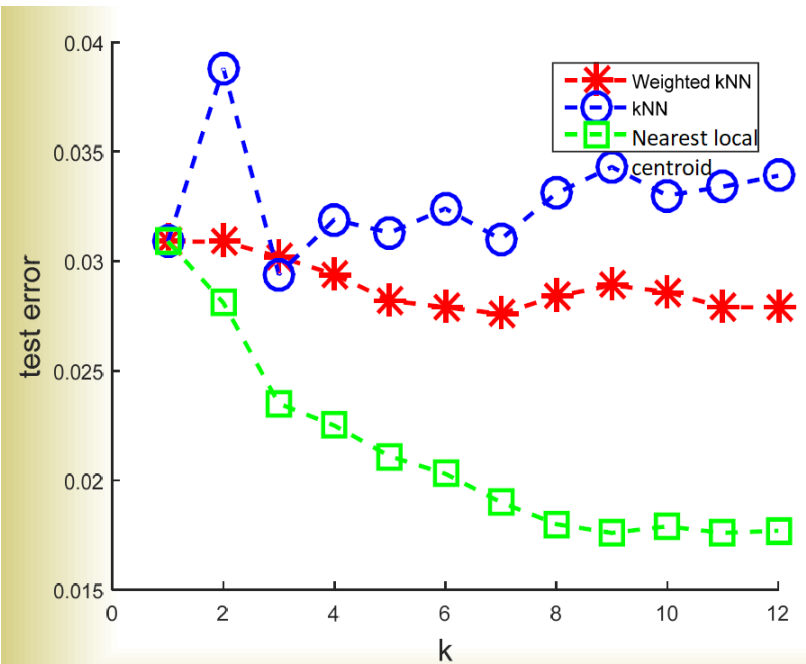


The global centers are often insufficient to summarize the training classes.

A fix is to only look at the most "relevant region" of each class (with respect to the given test point), and use their local centroids for prediction.

Nearest local centroid (NLC) classifier

Main idea: For each test point, find its k nearest neighbors inside each training class and assign the label for the test point based on the nearest local centroid.



The lowest test error rate is 1.76% (when $k = 9$).

But it may take a long time to compute the nearest local centroids.

What is the complexity of classifying one test point?

How to implement the NLC classifier

This classifier is not implemented by any software, so you will have to implement it from scratch (this is an excellent chance to hone your coding skills).

The following two MATLAB functions (and their counterparts in Python) will be very useful to you:

- $pdist2$: Pairwise distances between two sets of observations
- $knnsearch$: k nearest neighbor search

Assignment 1

Data set needed by this assignment:

Fashion-MNIST (both training and test data)[4]

- Benchmark results[5]
- Paper reference[6]: "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms"

[4] https://github.com/zalandoresearch/fashion-mnist
[5] http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/#
[6] https://arxiv.org/pdf/1708.07747.pdf

This assignment consists of the following questions:

1. For each $k = 1, \ldots, 12$, apply the plain kNN classifier with 6-fold cross validation to the training set and plot the validation error as a function of k. Similarly, for each k in the same range, apply the plain kNN classifier directly to the test set and plot the test errors in the same plot. Do the two error curves seem to be consistent (especially in terms of the optimal value of k for each error)? For the optimal k in terms of test error, can you identify a few misclassified images and show the k nearest training examples to understand why they were wrong?

2. For each $1 \leq k \leq 12$, perform kNN classification with each of the four distance metrics: Euclidean, Cityblock, Cosine and Correlation. Report the test error rates by plotting four curves together (one corresponding to a metric) against k. Which distance metric seems to work best on this data set and what is the lowest error rate you got overall?

3. For each $k = 1, \ldots, 12$, apply the weighted kNN classifier with the inverse weights to the data set. Repeat the experiment with squared inverse and linear weights. Plot the test errors corresponding to the three different weights against k and interpret the results.

4. Implement the nearest local centroid classifier to apply it to the data for each $k = 1, \ldots, 12$. Plot the test error curve and compare with the plain kNN classifier. What is the confusion matrix of the nearest local centroid classifier corresponding to the optimal k?

Read the following instructions carefully:

- Please prepare your results in presentation format (using Powerpoint or LaTex), and submit the slides to Canvas.
  - You should report the experimental results (such as accuracy and running time) through meaningful visuals such as figures, tables, etc. Please also briefly discuss your results.
  - For each question, you must clearly indicate all implementation details such as parameter values used.

- You must submit all your scripts separately to support your results. Make sure that your code is clearly labeled and organized, and readily executable.
- Your work will be graded based on correctness, completeness and clarity.

Reminder: Some of the programming assignments might take very long time (hours) to finish. Therefore, it is advisable to start doing the homework as early as possible, to allow enough time for the experiments to finish.

Also, use the *usps* data set (which is much smaller) for testing/debugging your code. Once it runs well, replace *usps* by *fasion-MNIST*.