LEC1: Instance-based classifiers

Dr. Guangliang Chen

February 2, 2016

Outline

- Having data ready
- kNN
- $k$means
- Summary

Downloading data

- Save all the scripts (from course webpage) and raw files (from LeCun's page) in the same folder
- Run `script_processing_raw_data.m` in MATLAB (from that folder)
- Then you should see two MATLAB data files: `MNIST_data_train.mat` and `MNIST_data_test.mat`

Data format

- Training data:
  trainImages 784x60000
  trainLabels 60000x1

- Test data:
  testImages 784x10000
  testLabels 10000x1

In both cases, images are stored as column vectors.

How to get matrix format?

Use MATLAB *reshape* function:

```
index = 2;
X = reshape(trainImages(:, index), 28, 28); % 28 x 28
figure; imagesc(X); colormap gray
```

What about a few images?

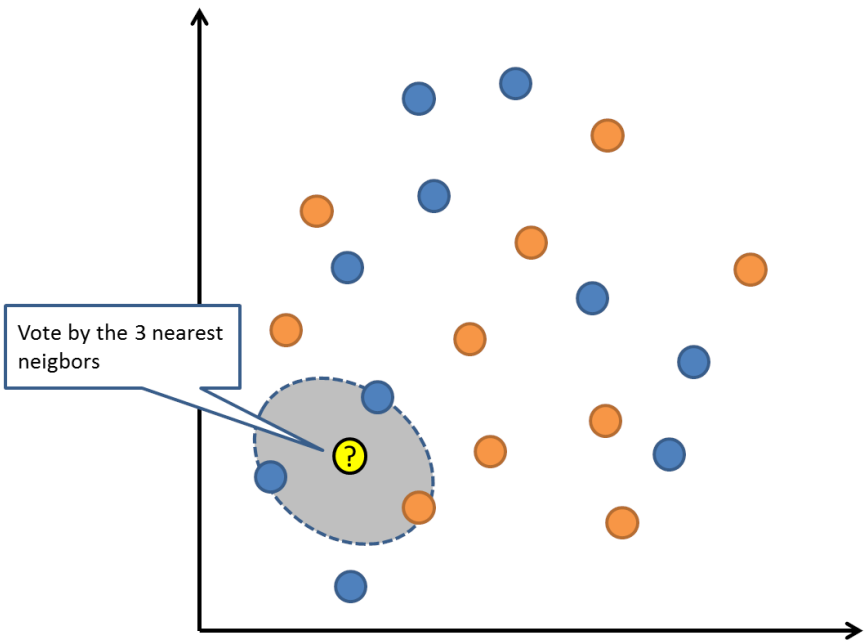
```
index = 1:10;
X = reshape(trainImages(:, index), 28, []); % 28 x 280
figure; imagesc(X); colormap gray
axis equal
```

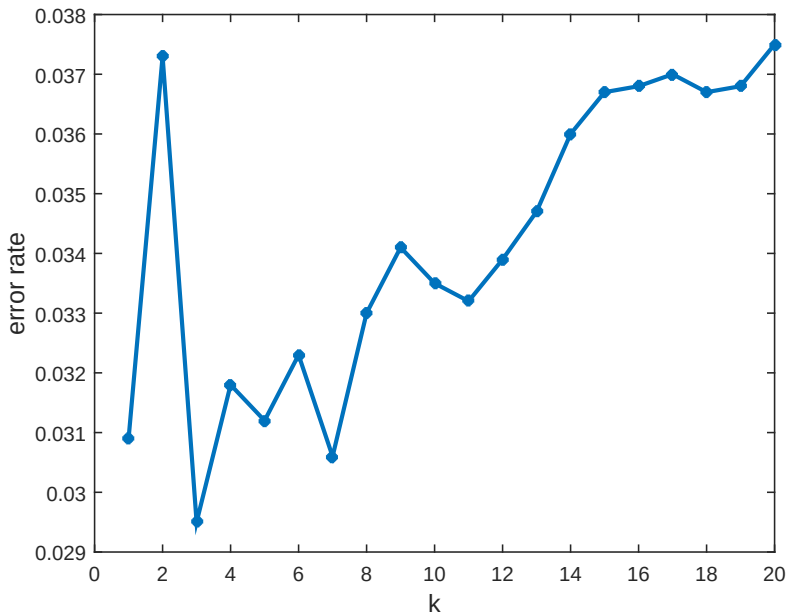
What about many images?

```
index = 1:100;
X = reshape(trainImages(:, index), 28, 28, []); % 28 x 28 x 100
DisplayImageCollection(X);
```

The k nearest neighbors (kNN) classifier

Determines the label of a test point based on those of k **closest training points**.





Critical questions in kNN classification

- How large should k be?
- What do we mean by "close"?
- How can we quickly find the nearest neighbors of a given point (especially in a large training set)?
- What are meaningful ways to evaluate the method?

Answers to previous questions

- Choose best k using *cross validation*
- Consider different distance metrics
- Use fast nearest neighbor search algorithms (e.g., kdtree)
- Test error (if given truth), decision map, etc.

m-fold cross validation

For each value of k do the following:

- Partition the training set randomly into m equal sized subsets
- Of the m subsets, one is retained as validation data and the remaining $m-1$ are used as training data
- The above process is repeated m times (the folds)

Note that every observation is used for validation exactly once. A combined error may thus be computed to evaluate the k value.

Common distance metrics (for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$)

- 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: $1 - \left\langle \frac{\mathbf{x}}{\|\mathbf{x}\|_2}, \frac{\mathbf{y}}{\|\mathbf{y}\|_2} \right\rangle$
- Correlation coefficient: $1 - \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}}$

The MATLAB $knnsearch$ function

IDX = knnsearch(X,Y);

This is the most basic way of using this function which finds the nearest neighbor in X for each point in Y. IDX is a column vector. Each row in IDX contains the index of nearest neighbor in X for the corresponding row in Y.

To find more than one nearest neighbor using a metric other than Euclidean, use

IDX = knnsearch(X,Y, 'k', 8, 'Distance', 'cityblock');

For full usage, type '**help knnsearch**' in MATLAB command window.

Another useful function $pdist2$

Pairwise distance between two sets of observations.

- For full usage, type '**help pdist2**' in MATLAB command window.
- **D = pdist2(X,Y);**
  Returns a matrix D containing the Euclidean distances between each pair of observations in the MX-by-N data matrix X and MY-by-N data matrix Y. Rows of X and Y correspond to observations, and columns correspond to variables. D is an MX-by-MY matrix, with the (I,J) entry equal to distance between observation I in X and observation J in Y.

- **D = pdist2(X,Y,DISTANCE)**. Choices of DISTANCE are:

  'euclidean' - Euclidean distance (default)

  'cityblock' - City Block distance

  'minkowski' - Minkowski distance. The default exponent is 2. To specify a different exponent, use D = pdist2(X,Y,'minkowski',P), where the exponent P is a scalar positive value.

  'chebychev' - Chebychev distance (maximum coordinate difference)

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

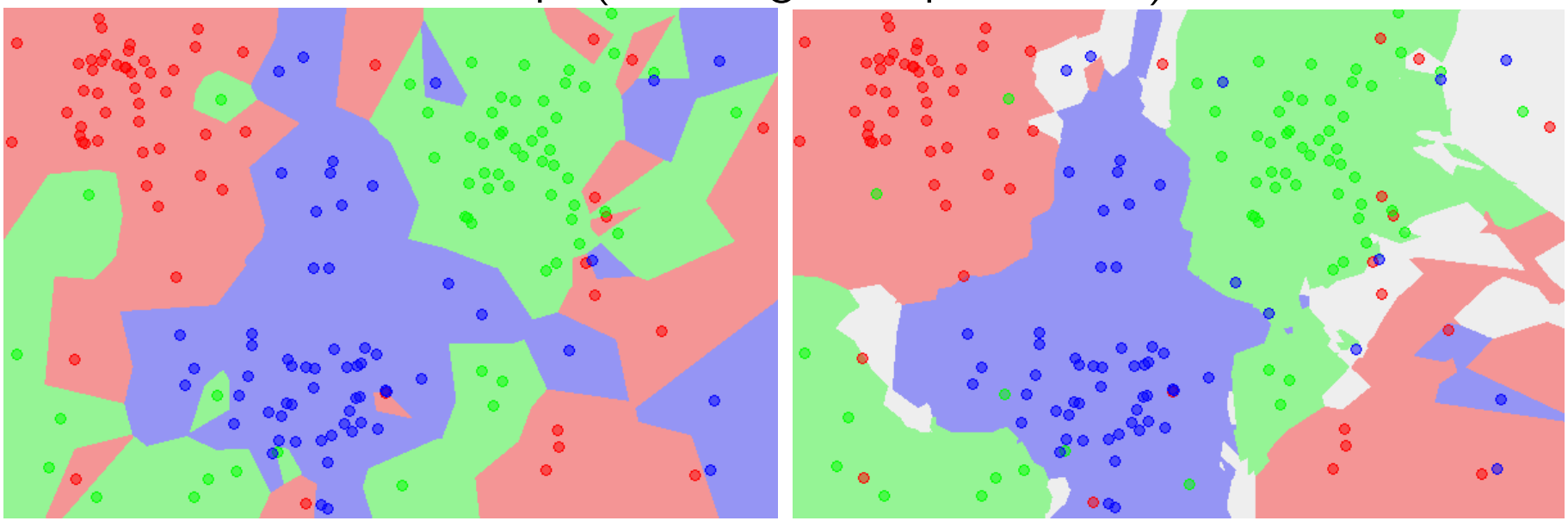
  'correlation' - One minus the sample linear correlation between observations (treated as sequences of values).

Ways to evaluate kNN classification

- Test error (if truth known)
- Validation error (for training data)
- Resubstitution error (for training data)
- Confusion matrix
- Decision map

Decision maps

1NN and 5NN decision maps (white regions represent ties)



(Pictures taken from `wikipedia.org`)

Matlab commands for kNN classification

% fit a kNN classification model
mdl = fitcknn(trainX, trainLabels, 'NumNeighbors', 3);

% apply the model to test data and compute test error
pred = predict(mdl, testX);
testError = sum(testLabels =pred)/numel(testLabels)

% 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)

Weighted kNN

The motivation is to give closer neighbors more voting weights.

Let $\mathbf{x}$ be a test point, whose k nearest training examples are $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(k)}$ (with labels $y_1, \ldots, y_k$ respectively).

The plain kNN classification rule can be denoted as follows:

$$\hat{y} = \arg\max_y \sum 1_{y_i = y}$$

We introduce weights into the rule:

$$\hat{y} = \arg\max_y \sum w_i \, 1_{y_i = y}$$

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:

- Linearly decaying weights
- (Squared) inverse weights
- Exponentially decaying 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);

Weighted kNN on MNIST Digits

Recall that the smallest error obtained by a plain kNN classifier on the MNIST handwritten digits is 2.94% (when $k = 3$).

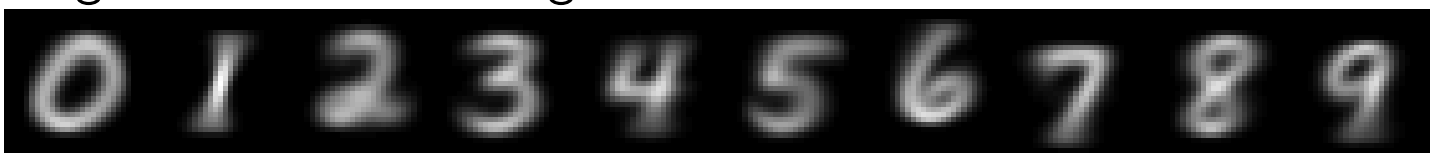
We fix $k = 3$ but experiment with the following kinds of weights:

- Inverse: 2.83%
- Squared inverse: 2.83%
- Linear: Q4 of HW1 (due Friday noon).
- Normal: ???

Note: Slightly bigger values of k should be more suitable for weighted kNN.

The global $k$means classifier

Assign labels to test images based on the most similar centers.



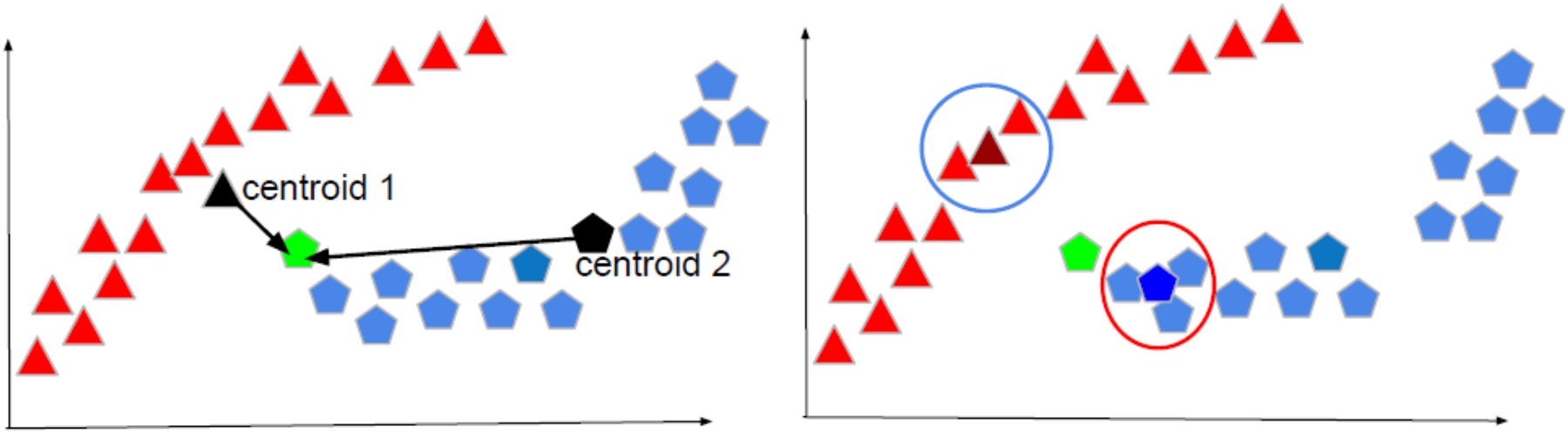
Recall that it has a 18.0% error rate.

It is an example of "nearest subset" classifiers

$$\hat{y} = \text{argmin}_y \ \text{dist}(\mathbf{x}, \mathcal{C}_y)$$

in which $\text{dist}(\mathbf{x}, \mathcal{C}_y)$ represents the distance from a test point $\mathbf{x}$ to a training class $\mathcal{C}_y$.

The local $k$means classifier



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

An alternative is to only look at the most "relevant region" of each class (with respect to the given test point).

The local $k$means classifier (cont'd)

We may use the k nearest points in each class to represent the corresponding class, and then perform $k$means classification based on the reduced data. We call this approach *local $k$means*.

One thing to note: local 1means $=$ 1NN.
This implies that local 1means has a 3.1% error rate from previous plot.

What about larger k?

Comments on kNN and $k$means

- Instance-based learning (or lazy learning)
- Model free (nonparametric)
- Simple to implement, yet quite powerful
- It is a localized and nonlinear approach
- Algorithmic complexity only depends nearest neighbors search (memory and CPU cost)
- The choice of k is critical (may need cross validation)
- Lots of variations (weighted kNN, local $k$means)

Some learning resources

- MATLAB Tutorials (`http://www.mathworks.com/support/learn-with-matlab-tutorials.html`)
- MATLAB Statistics and Machine Learning Toolbox Documentation (`http://www.mathworks.com/help/stats/index.html`)
- kNN in Python (`http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/`)
- Google's Python class (`https://developers.google.com/edu/python/`)
- If you still don't know how to do something, just google.

Assignment 1 (due 2/9, Tuesday)

This homework only uses the MNIST Digits dataset (both training and test data).

1. Implement the plain kNN classifier and apply it with 6-fold cross validation to the training set to select the best k in the range 1:10. Be sure to plot the curve of validation error versus k. Is it consistent with the test error curve that I obtained?

2. For the optimal k found above, perform kNN classification on the test set. What is the error rate? Display also the confusion matrix.

3. For each $1 \leq k \leq 10$ apply the kNN classifier with the city-block distance metric to the test set. Show the test errors curve corresponding to the different k. How does it compare with the Euclidean distance?

4. Implement the weighted kNN classifier with linearly decreasing weights, for each $k = 3, \ldots, 12$ and apply it directly to the test set. Show the test errors curve and interpret the results.

5. Implement the local $k$means classifier for $k = 2, \ldots, 10$ and apply it to the test data directly. What is the result like now?

Reminder: Some of the programming assignments might take very long time (could be days) to finish. Therefore, it is advisable to start doing the homework as early as possible, definitely before this weekend.