K-Nearest Neighbours Classification

Instructor: Farzaneh Mirzazadeh

Department of Computer Science, UCSC, Winter 2017

Slide contents from slides of *Machine Learning* courses of

- Prof David Helmbold
- Prof Kevin Murphy
- Prof Richard Zemel and Prof Raquel Urtasun

Thanks to all of them.

Introduction

Topic of today: K-nearest neighbors (KNN) classification

Belongs to a number of families of machine learning methods:

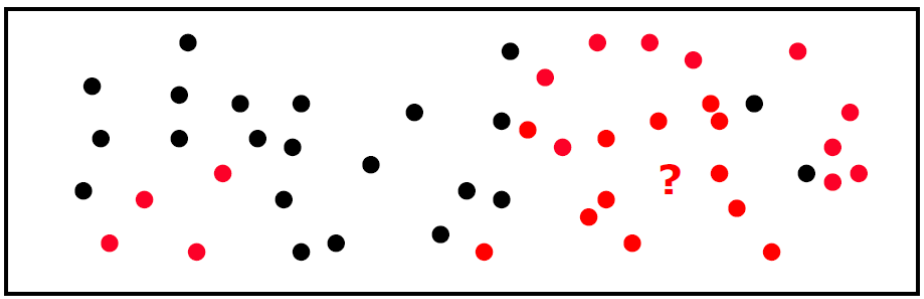
1. Non-parametric as opposed to parametric method
   - **Parametric methods**: Fit a parametric model. Summarize the model by a number of parameters (eg feature weights in linear and logistic regression)
   - **Non-parametric methods**: Do not fit a parametric model. Memorize training data and make prediction for a new data point based on similar training example

2. Lazy as opposed to eager method
   - **Lazy methods**: Generalization is delayed until a query comes to the system. No training phase.
   - **Eager methods**: Constructs a model based on data at training time before query comes.

3. Instance based method: instead of explicit generalization, compares new problem instances with training instances, stored in memory.

K Nearest Neighbors (KNN) classification

- One of the simplest methods for supervised learning
- Works surprisingly well from time to time, and if tuned well
- Views data as points in a Euclidean space, then deals with distances from them
- Learning step amounts to memorizing (storing) training data only



Rationale behind this type of classification methods

- The features that are used to describe and represent the data are relevant to their targets, in a way that close-by points have the same labels
- Targets vary smoothly with input

K-NN with $K = 1$: **Nearest Neighbor Classifier**

I. Definition

Nearest Neighbor Method

- **Training time procedure**: Store training examples
- **Testing time procedure**: Given a new test example $\mathbf{x}_\circ$, find the training example $(\mathbf{x}_i, y_i) \in S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t)\}$ whose Euclidean (i.e. L_2) distance $\sqrt{\|\mathbf{x}_\circ - \mathbf{x}_i\|_2^2}$ from $\mathbf{x}_\circ$ is smallest. Return its target $\hat{y}(\mathbf{x}_\circ) = y_i$ as the prediction.

Note:

- We do not need to compute square root to find the smallest distance. Why?
- Because if we raise them all to power 2, still the closest examples are the same. Ordering does not change.
- Note: Nearest neighbor algorithm above works for multiclass classification too.

I. Definition

Nearest Neighbor Method

- **Training time procedure**: Store training examples
- **Testing time procedure**: Given a new test example $\mathbf{x}_\circ$, find the training example $(\mathbf{x}_i, y_i) \in S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t)\}$ whose Euclidean (i.e. L_2) distance $\sqrt{\|\mathbf{x}_\circ - \mathbf{x}_i\|_2^2}$ from $\mathbf{x}_\circ$ is smallest. Return its target $\hat{y}(\mathbf{x}_\circ) = y_i$ as the prediction.

Note:

- We do not need to compute square root to find the smallest distance. Why?
- Because if we raise them all to power 2, still the closest examples are the same. Ordering does not change.
- Note: Nearest neighbor algorithm above works for multiclass classification too.

I. Definition

Nearest Neighbor Method

- **Training time procedure**: Store training examples
- **Testing time procedure**: Given a new test example $\mathbf{x}_\circ$, find the training example $(\mathbf{x}_i, y_i) \in S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_t, y_t)\}$ whose Euclidean (i.e. L_2) distance $\sqrt{\|\mathbf{x}_\circ - \mathbf{x}_i\|_2^2}$ from $\mathbf{x}_\circ$ is smallest. Return its target $\hat{y}(\mathbf{x}_\circ) = y_i$ as the prediction.

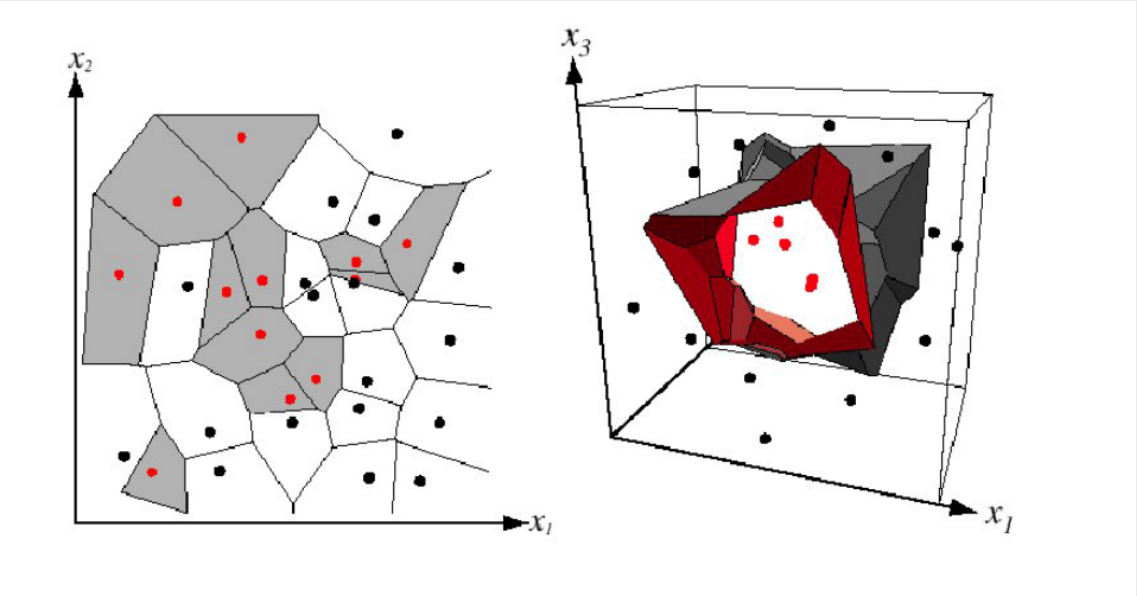
Note:

- We do not need to compute square root to find the smallest distance. Why?
- Because if we raise them all to power 2, still the closest examples are the same. Ordering does not change.
- Note: Nearest neighbor algorithm above works for multiclass classification too.

II. Decision boundary

Voronoi Diagram

- It is possible to infer regions with prediction value 0 and regions with prediction value 1 from training data
- Visualization of such decision regions is called a Voronoi diagram.
- Not explicitly computed during classification but can be inferred.
- Useful for studying properties of the method
- Sample Voronoi diagrams (See Figure below, 2D in LHS, 3D in RHS)



From Richard Zemel and Raquel Urtasun's slide

III. Prediction time complexity

Brute force algorithm

- **Compute the distance from every training example, find the minimum**
- $O(t \times n)$ time, where $t = \#$training examples and $n = \#$ features
- Pretty expensive test time. (Compared to parametric methods that have very cheap test time.)

Any idea for speed-up?

Prediction time speed-up?

Improved algorithms

1. Use **advanced data structures** to store training data that facilitate more efficient retrieval of the smallest distance.
   Examples:
   - kd tree
   - Ball tree
   - Cover tree
2. Use **locality sensitive hashing**: A randomized algorithm that with high probability hashes close-by points into the same bin.
3. Find approximately nearest neighbors instead of exact, in the above two.

What is the price?

- Increases training time, used for construction of the data structures.
- Is it bad? Not actually. Training is performed once and is offline. But prediction price has to be paid every time for each query.

Prediction time speed-up?

Improved algorithms

1. Use **advanced data structures** to store training data that facilitate more efficient retrieval of the smallest distance.
   Examples:
   - kd tree
   - Ball tree
   - Cover tree
2. Use **locality sensitive hashing**: A randomized algorithm that with high probability hashes close-by points into the same bin.
3. Find approximately nearest neighbors instead of exact, in the above two.

What is the price?

- Increases training time, used for construction of the data structures.
- Is it bad? Not actually. Training is performed once and is offline. But prediction price has to be paid every time for each query.

Prediction time speed-up?

Improved algorithms

1. Use **advanced data structures** to store training data that facilitate more efficient retrieval of the smallest distance.
   Examples:
   - kd tree
   - Ball tree
   - Cover tree
2. Use **locality sensitive hashing**: A randomized algorithm that with high probability hashes close-by points into the same bin.
3. Find approximately nearest neighbors instead of exact, in the above two.

What is the price?

- Increases training time, used for construction of the data structures.
- Is it bad? Not actually. Training is performed once and is offline. But prediction price has to be paid every time for each query.

Prediction time speed-up?

Improved algorithms

1. Use **advanced data structures** to store training data that facilitate more efficient retrieval of the smallest distance.
   Examples:
   - kd tree
   - Ball tree
   - Cover tree
2. Use **locality sensitive hashing**: A randomized algorithm that with high probability hashes close-by points into the same bin.
3. Find approximately nearest neighbors instead of exact, in the above two.

What is the price?

- Increases training time, used for construction of the data structures.
- Is it bad? Not actually. Training is performed once and is offline. But prediction price has to be paid every time for each query.

Prediction time speed-up?

Improved algorithms

1. Use **advanced data structures** to store training data that facilitate more efficient retrieval of the smallest distance.
   Examples:
   - kd tree
   - Ball tree
   - Cover tree
2. Use **locality sensitive hashing**: A randomized algorithm that with high probability hashes close-by points into the same bin.
3. Find approximately nearest neighbors instead of exact, in the above two.

What is the price?

- Increases training time, used for construction of the data structures.
- Is it bad? Not actually. Training is performed once and is offline. But prediction price has to be paid every time for each query.

IV. Space complexity

$O(t \times n)$ space
All training data needs to be stored. Pretty expensive.

Improvement could happen if we can

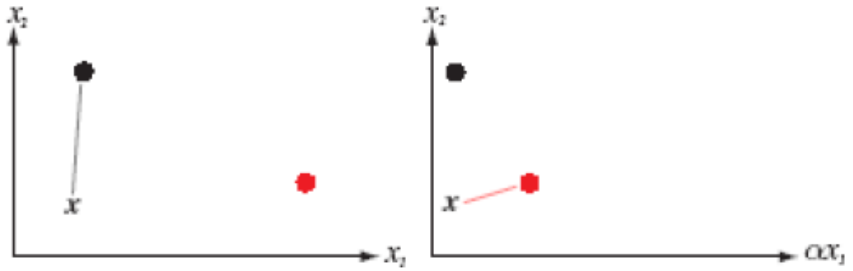
- Remove some data points (like the ones inside Voronoi decision regions as opposed to the boundary)
- Remove some of features

Details out of scope

V. Feature importance and scale

- Features with larger range will have more influence on the distance.

If we scale x1 by 1/3, NN changes!



From Kevin Murphy's slide

- Irrelevant and redundant features affect distance.

Tricks to solve

- Standardize features (Dividing each feature by the feature range works here)
- Weight features according to their importance in distance, then compute the distance
- Use Mahalanobis distance: Not only weight single features, but tweak interaction between features too. (See next slide for definition)

What is a Mahalanobis distance?

- It is scaling interactions between features by using a matrix Σ

$$dist^2(u,v) = (u-v)^\top \Sigma (u-v)$$
$$= \sum_i \sum_j (u_i - v_i)\Sigma_{ij}(u_i - v_j)$$

- Σ is a positive semidefinite matrix, i.e. $\Sigma \succeq 0$
- Since $\Sigma = A^\top A$ for some A, you could substitute:

$$dist^2(u,v) = (u-v)^\top A^\top A (u-v)$$

  Meaning: points are first mapped to a new space with matrix A, and then Euclidean distance is computed in that space.
- For Euclidean distance set $\Sigma = I$, where I is the identity matrix.

From Kevin Murphy's slide

VI. Sensitivity to class noise

- Nearest neighbor predictions very sensitive to mislabeled training examples, aka class noise.
- Has non-smooth predictions. 1 s causes the prediction to be 0.

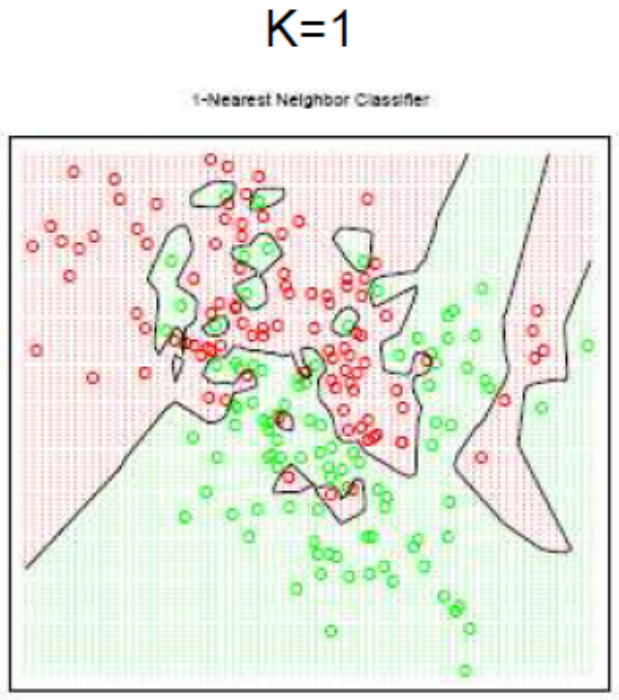


Fig 2.2, 2.3 of HTF01

- Tiny islands.
- Any idea to fix this? See next slides.

From Kevin Murphy's slide

VI. Sensitivity to class noise

- Nearest neighbor predictions very sensitive to mislabeled training examples, aka class noise.
- Has non-smooth predictions. 1 s causes the prediction to be 0.

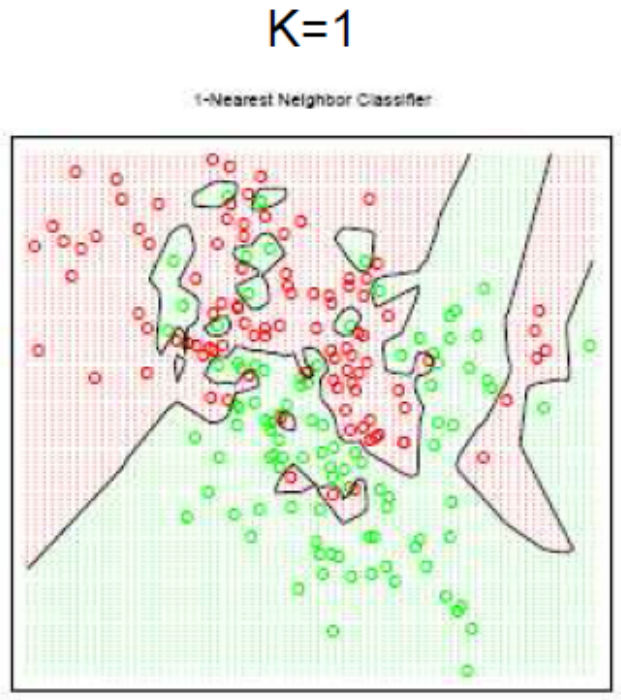


Fig 2.2, 2.3 of HTF01

- Tiny islands.
- Any idea to fix this? See next slides.

From Kevin Murphy's slide

How to make more robust to class noise?

- Idea: Smooth by having K nearest neighbors vote
- Increasing K yields smoother predictions, since we average over more data points (See Figure in right)

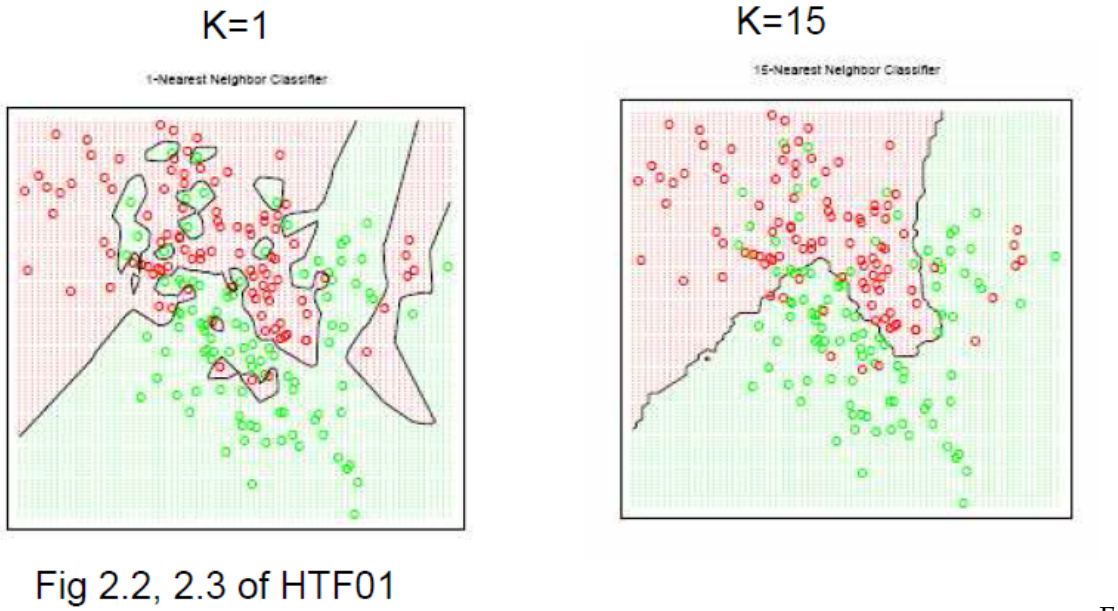


Fig 2.2, 2.3 of HTF01

From Kevin Murphy's slide

- Two extremes for value of K
  1. $K = 1$ yields piecewise constant labels
  2. $K = t$ yields globally constant (majority) label

K Nearest Neighbors Classifier

I. Definition

K Nearest Neighbors classification

- Training time procedure: Store training examples
- Testing time procedure: Given a new test example $\mathbf{x}_\circ$, find thw K closest neighbors of it in the training set, then for each class count the number of K nearest neighbors that have it as their target, finally find the the majority class, return it as the prediction for $\mathbf{x}_\circ$
- The mathematical formulation for prediction (test time):

$$\hat{y}(\mathbf{x}) = C^* \quad \text{where} \quad C^* = \arg\max_C \sum_{\mathbf{x} \in K\text{NN}(\mathbf{x}_\circ)} \mathbf{I}(y(\mathbf{x}) == C),$$

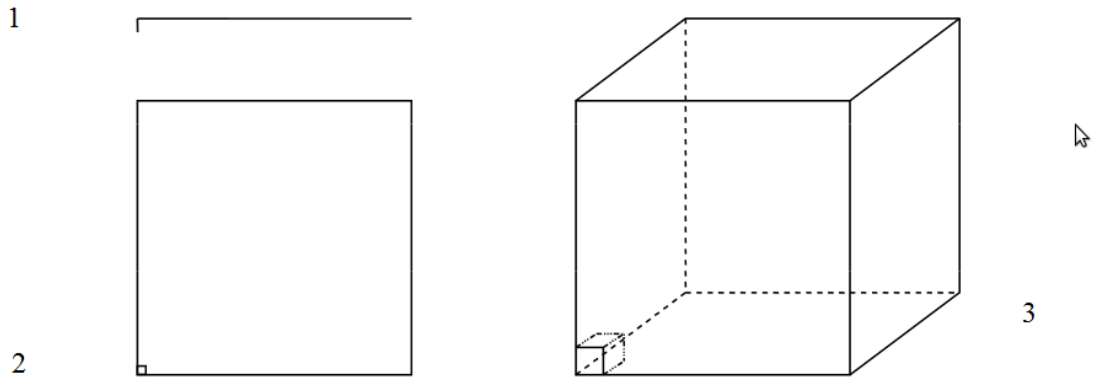
where $\mathbf{I}$ is the indicator function and is one when input is true, and $K\text{NN}(\mathbf{x}_\circ)$ is the set of K nearest neighbors of $\mathbf{x}_\circ$ in training examples. Note that by computing sum of indicator functions, we perform counting.

III. Curse of dimensionality and its effect in KNN

Curse of dimensionality refers to the issue that same problems that could be easily solved in low dimension (i.e. when number of features is small), get very (even exponentially) harder to solve in high dimension (i.e. when number of features is large).

Curse of dimensionality in K-NN

- *k*NN breaks down in high-dimensional space
  - "Neighborhood" becomes very large.
- Assume 5000 points uniformly distributed in the unit hypercube and we want to apply 5-nn. Suppose our query point is at the origin.
  - In 1-dimension, we must go a distance of 5/5000 = 0.001 on the average to capture 5 nearest neighbors
  - In 2 dimensions, we must go $\sqrt{0.001}$ to get a square that contains 0.001 of the volume.
  - In d dimensions, we must go $(0.001)^{1/d}$



How to tune hyper-parameter K of KNN?

Question: How many neighbors should we consider?

Hyper-parameter setting

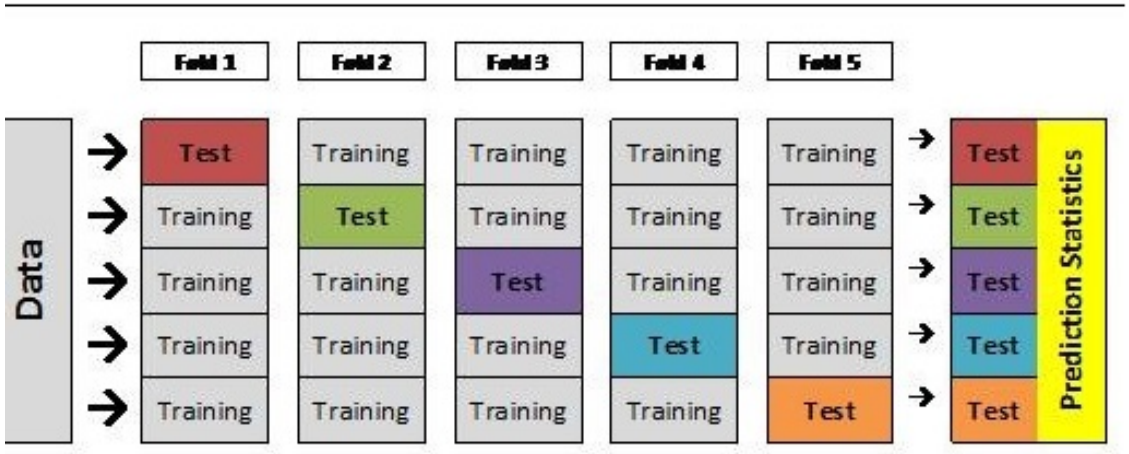
- A similar task as setting regularization parameter. Hyper-parameter setting
- Can use part of training data as validation set. Decide which of candidates are better based on prediction accuracy on validation set. Problem: In small sample size, we could get lucky that the validation section happens to be an easier to predict part of our sample.
- Alternatively: Can use cross-validation to decide which of candidates are better.
- The problem of tunning K is an example of model selection.

Cross-validation

f fold cross-validation

1. Divide training data to f folds with (almost) equal size.
2. Repeat i = 1 to f
   - Train on all except the the i th fold.
   - Validate on fold i
3. Combine validation errors

Demonstration of 5 fold cross validation:



Cross-validation

- Economic in the use of training data.
- The general procedure could be used both for model selection and for final model evaluation to announce the test error.

Popular f fold cross-validation

- 5 fold cross-validation
- 10 fold cross-validation
- Leave-one out: Size of each fold in one.

Summary

In this lecture we talked about

- Nearest neighbor (NN) method for classification
- KNN method for classification
- Cross validation

Note: KNN could be used for regression too. Any idea how?