Machine Learning, Spring 2020

Instance Based Learning (KNN Classification)

Reading Assignment: Chapter 6 & 7

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

PyTorch tutorial: https://pytorch.org/tutorials/

Acknowledge: The slides are partially referred to coursera online machine learning course by Prof. Andrew Ng, BYU Data Mining Lab, CS-478-Machine learning, and NYU machine learning course.

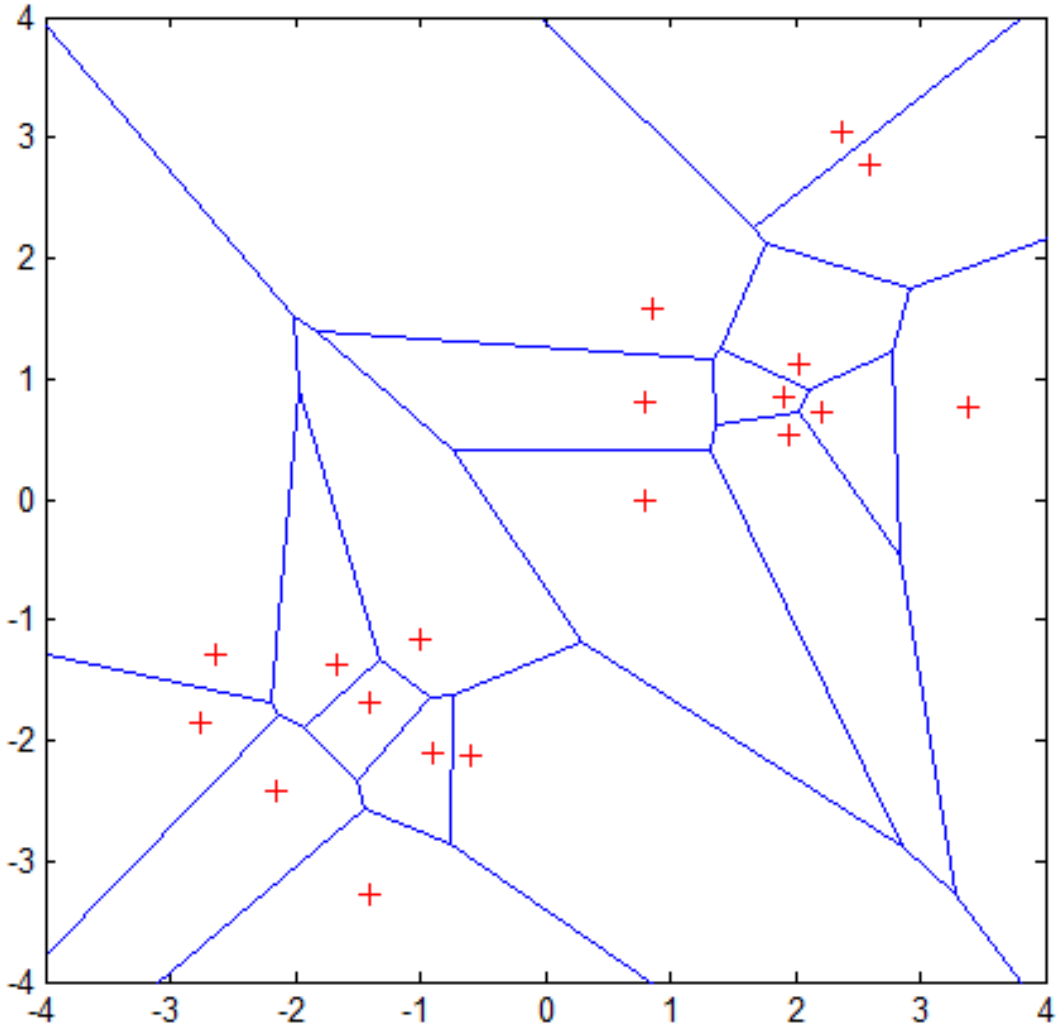
Introduction

- Instance-based learning is often termed *lazy* learning, as there is typically no “transformation” of training instances into more general “statements”
- Instead, the presented training data is simply stored and, when a new query instance is encountered, a set of similar, related instances is retrieved from memory and used to classify the new query instance
- Hence, instance-based learners never form an explicit general hypothesis regarding the target function. They simply compute the classification of each new query instance as needed

k-NN Approach

- The simplest, most used instance-based learning algorithm is the *k*-NN algorithm
- *k*-NN assumes that all instances are points in some *n*-dimensional space and defines neighbors in terms of distance (usually Euclidean in R-space)
- *k* is the number of neighbors considered

Graphic Depiction



Properties:

1) All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample
2) For any sample, the nearest sample is determined by the closest Voronoi cell edge

Basic Idea

- Using the second property, the *k*-NN classification rule is to assign to a test sample the majority category label of its *k* nearest training samples
- In practice, *k* is usually chosen to be odd, so as to avoid ties
- The *k* = 1 rule is generally called the nearest-neighbor classification rule

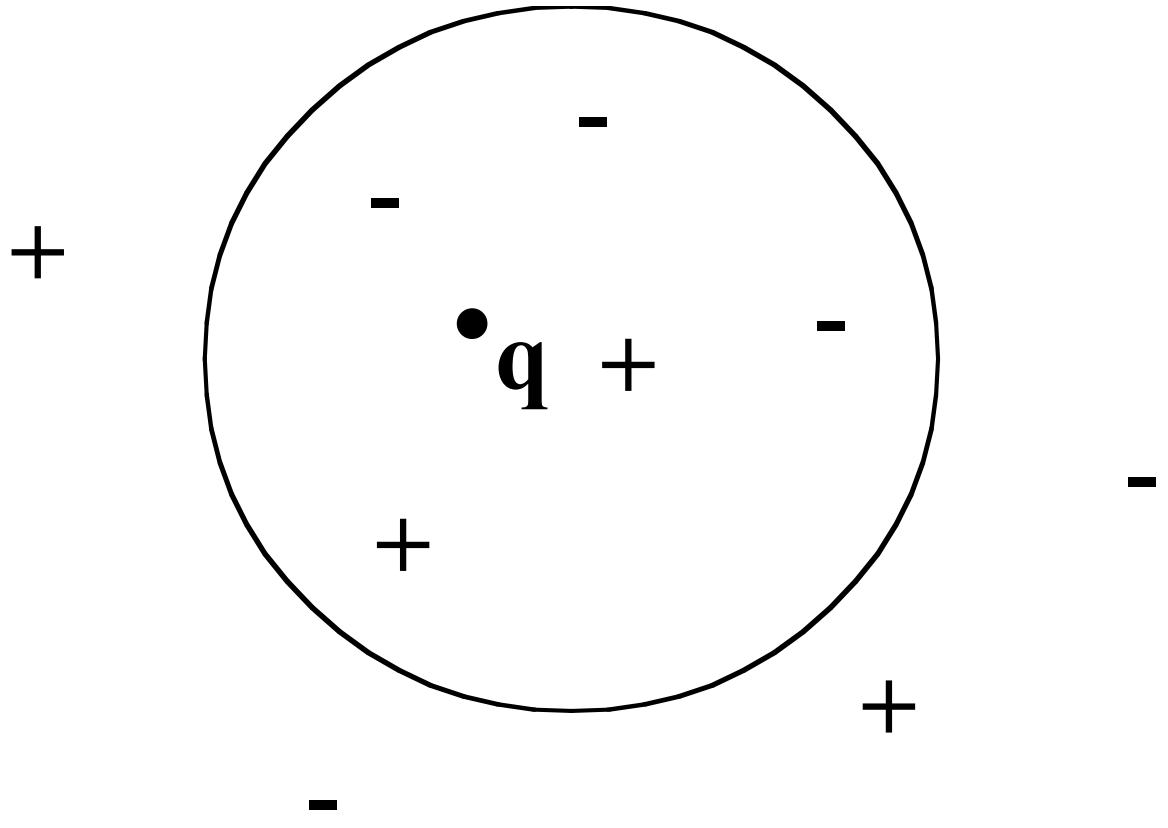
k-NN Algorithm

- For each training instance $t=(x, f(x))$
  - Add t to the set *Tr_instances*
- Given a query instance q to be classified
  - Let $x_1, \ldots, x_k$ be the k training instances in *Tr_instances* nearest to q
  - Return

$$\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

- Where V is the finite set of target class values, and $\delta(a,b)=1$ if $a=b$, and 0 otherwise (Kronecker function)
- Intuitively, the *k*-NN algorithm assigns to each new query instance the majority class among its k nearest neighbors

Simple Illustration



q is + under 1-NN, but – under 5-NN

Distance-weighted *k*-NN

- Replace $\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$ by:

$$\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} \frac{1}{d\left(x_i, x_q\right)^2} \delta(v, f(x_i))$$

Scale Effects

- Different features may have different measurement scales
  - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
  - Patient weight will have a much greater influence on the distance between samples
  - May bias the performance of the classifier

Standardization

- Transform raw feature values into z-scores

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

  - x_{ij} is the value for the i^{th} sample and j^{th} feature
  - μ_j is the average of all x_{ij} for feature j
  - σ_j is the standard deviation of all x_{ij} over all input samples
- Range and scale of z-scores should be similar (providing distributions of raw feature values are alike)

Predicting Continuous Values

- Replace $\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$ by:

$$\hat{f}(q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

- Note: unweighted corresponds to w_i=1 for all i