K nearest neighbor

Lazy vs. Eager Learning

- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the previous discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

Lazy Learner: Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing (“lazy evaluation”) until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

K Nearest Neighbor

- K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).
- KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.
- The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

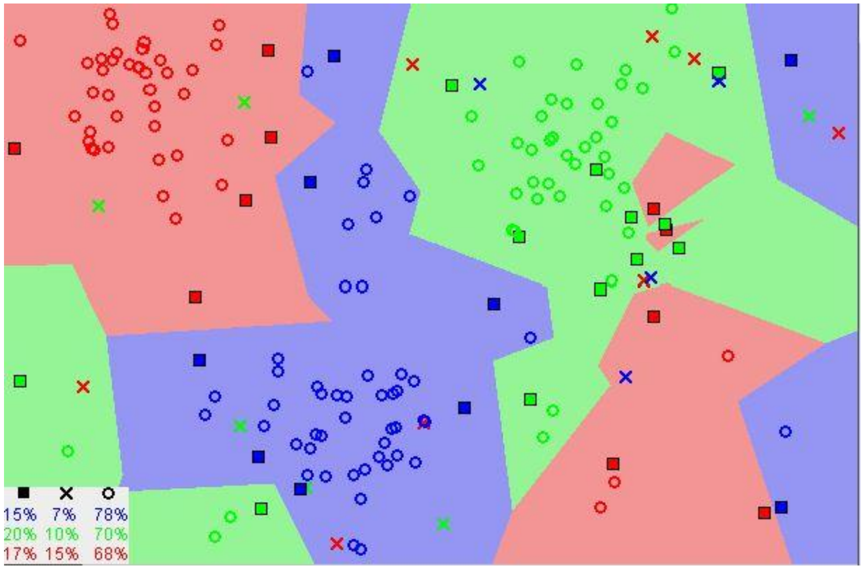
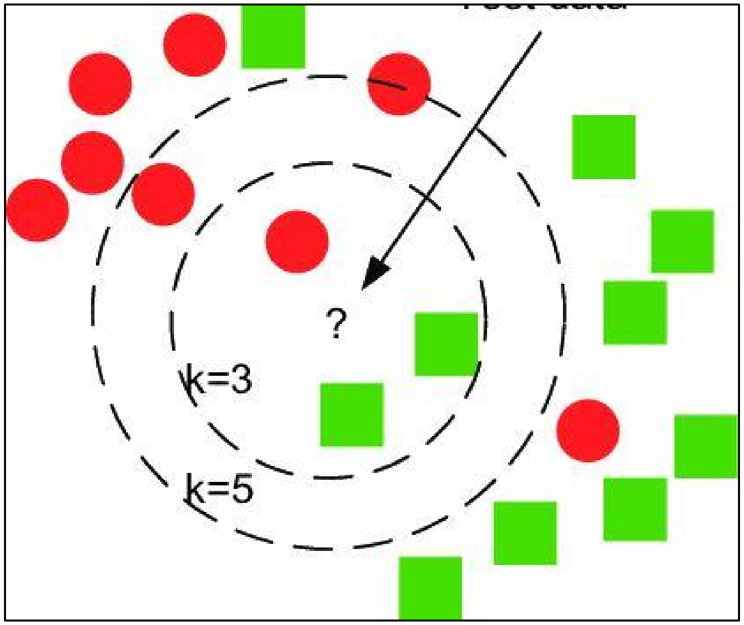


Image showing how similar data points typically exist close to each other

The *k*-Nearest Neighbor Algorithm

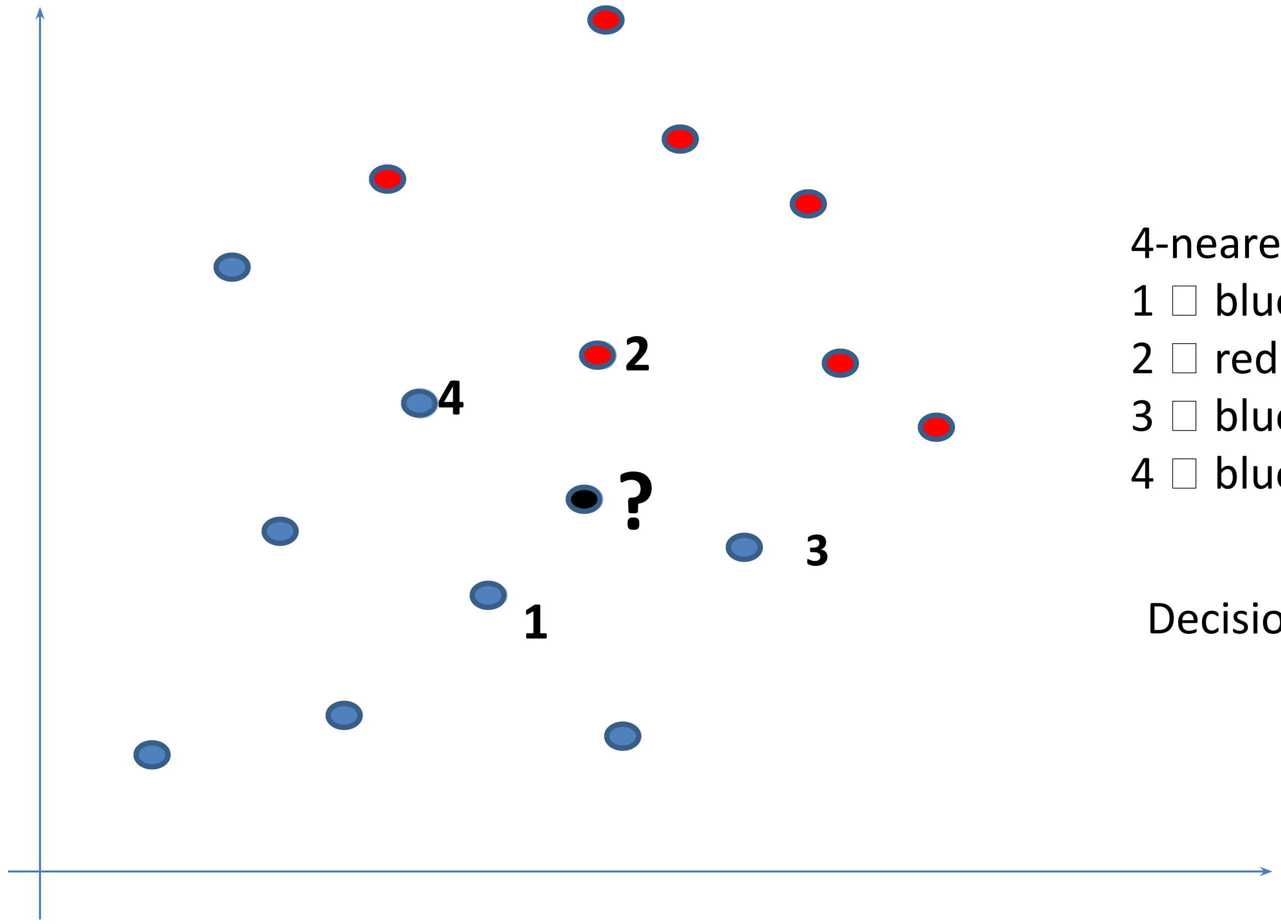
- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of distance, dist($\mathbf{X}_1$, $\mathbf{X}_2$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q



The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data
    - Calculate the distance between the query example and the current example from the data.
    - Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

Illustration of KNN



4-nearest neigbor :

1 □ blue

2 □ red

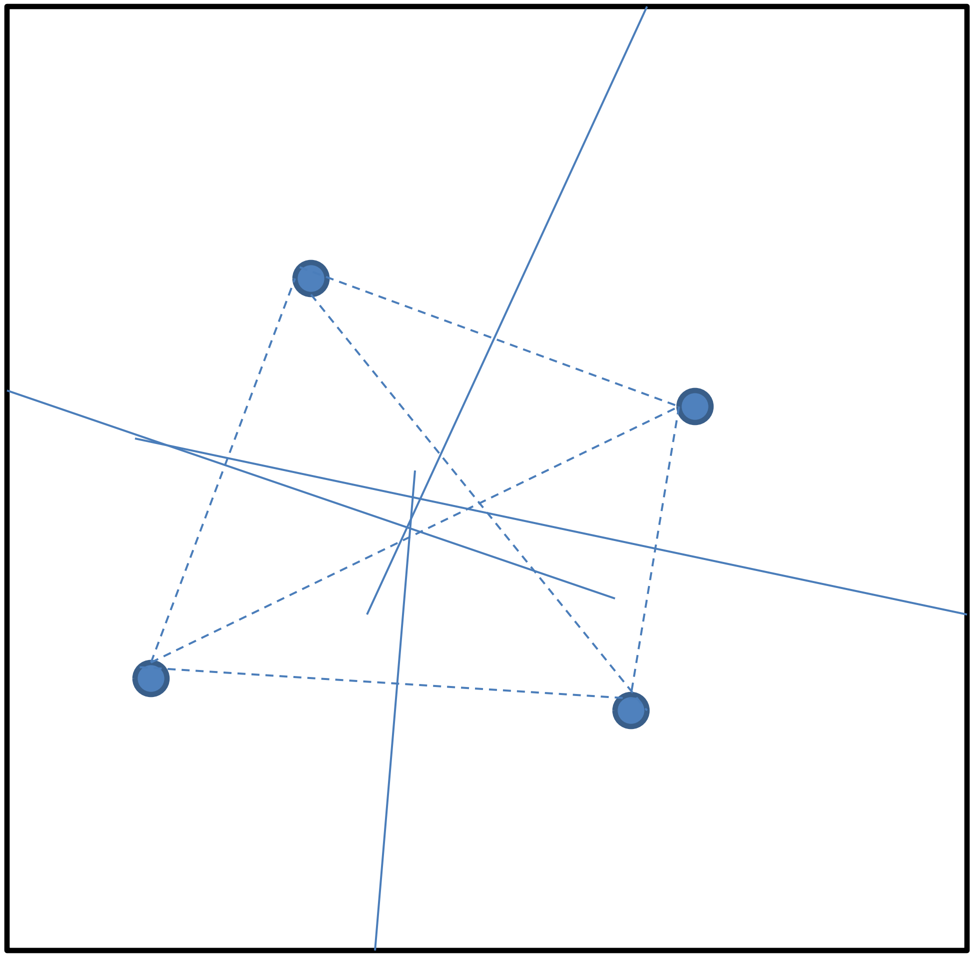
3 □ blue

4 □ blue

Decision is blue

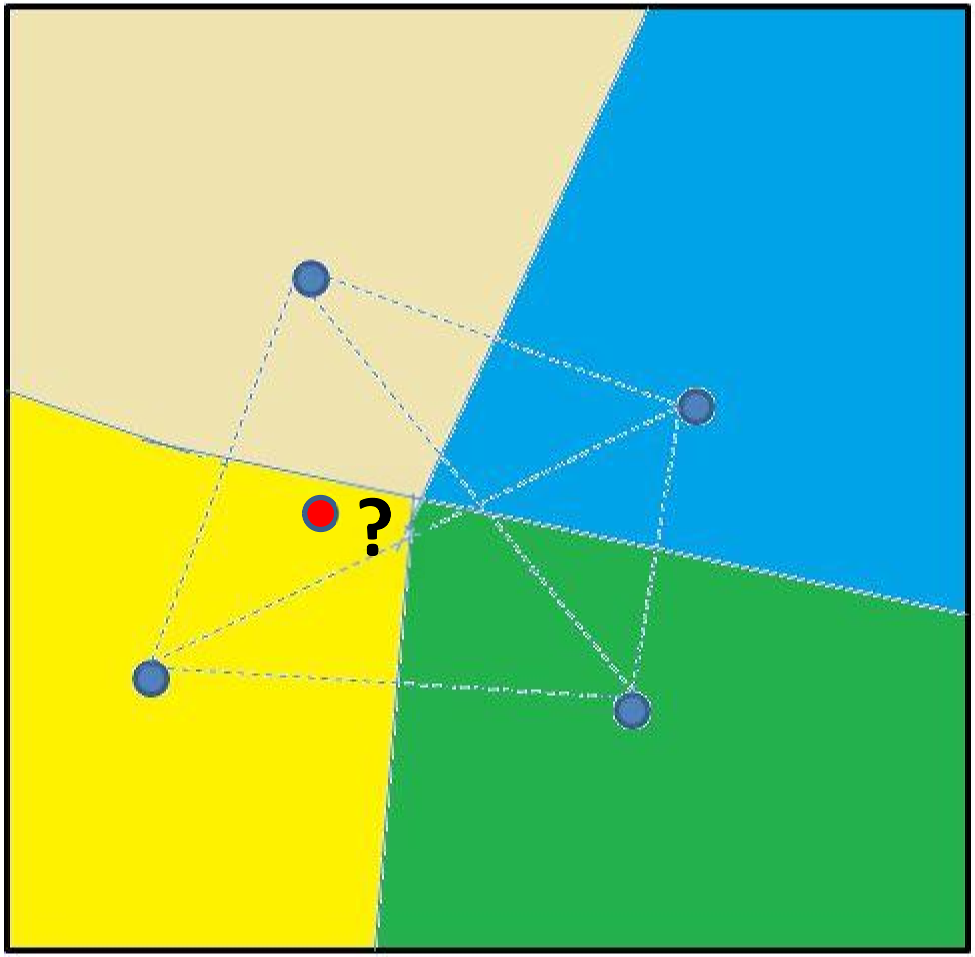
Vonoroi Diagram

- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples
- The set of polygons created have a unique feature — edges of the polygons are the perpendicular bisectors of the lines joining the neighboring points.



Vonoroi Diagram

- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples
- The set of polygons created have a unique feature — edges of the polygons are the perpendicular bisectors of the lines joining the neighboring points.



Choosing the right value for K

Run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors.

Here are some things to keep in mind:

- As we decrease the value of K to 1, our predictions become less stable.
- Inversely, as we increase the value of K, our predictions become more stable (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
- In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Advantages and disadvantages of KNN

Advantages

- The algorithm is simple and easy to implement.
- There’s no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).

Disadvantages

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

Discussion on the *k*-NN Algorithm

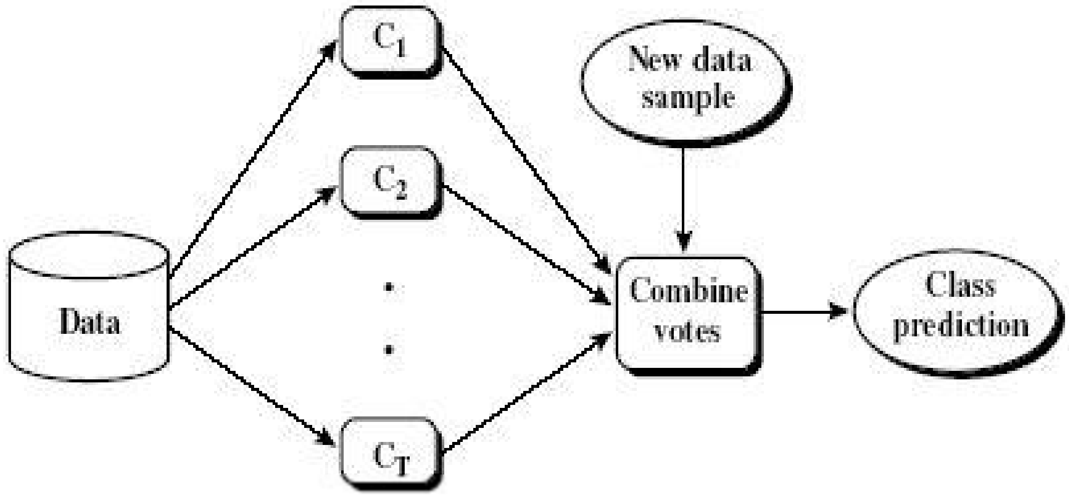
- *k*-NN for real-valued prediction for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query x_q
    - Give greater weight to closer neighbors

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

- Robust to noisy data by averaging *k*-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

TECHNIQUES TO IMPROVE CLASSIFICATION ACCURACY: ENSEMBLE METHODS

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors’ majority vote
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified** by M_i
  - The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

TERIMA KASIH