Table of contents

1. Introduction

Most of this material is derived from the text, Han, Kamber and Pei, Chapter 9, or the corresponding powerpoint slides made available by the publisher. Where a source other than the text or its slides was used for the material, attribution is given. Unless otherwise stated, images are copyright of the publisher, Elsevier.

This topic concludes our work on the data mining problems of classification and prediction with a look at the very popular Support Vector Machine kernel method as well as lazy learning algorithms including k-nearest neighbour, and finishing up with a brief summary of some variants of the classification and prediction data mining problem.

2. Support Vector Machine (Text: 9.3)

Support vector machines (SVMs)

- One of the most successful *classification* methods for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** (“essential” training tuples) and **margins** (defined by the support vectors)

History and Applications

- Vapnik and colleagues (1992): Support Vector Machine
  - groundwork from Vapnik & Chervonenkis’ statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximisation)
- Used for: classification and numeric prediction(regression)
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

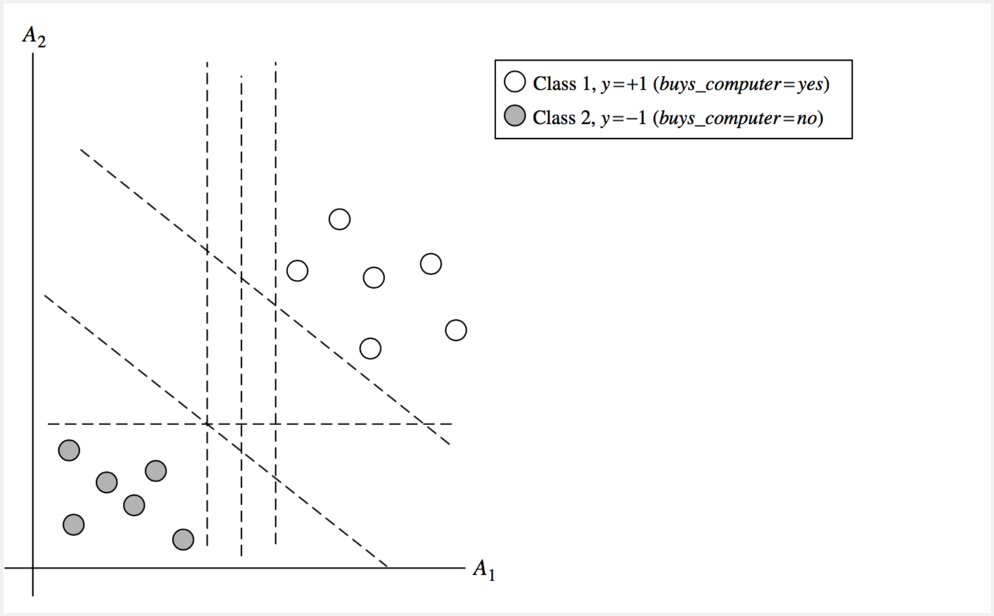
Algorithms

- Training of SVM can be distinguished by
  - Linearly separable case
  - Non-linearly separable case

2.1. Linearly Separable Case

SVM with Linearly Separable Training Data

- Let's consider buys_computer example with two input attributes A_1 and A_2. If the training tuples can be plotted as follows (x-axis and y-axis represent A_1 and A_2, respectively), then the dataset is linearly separable:



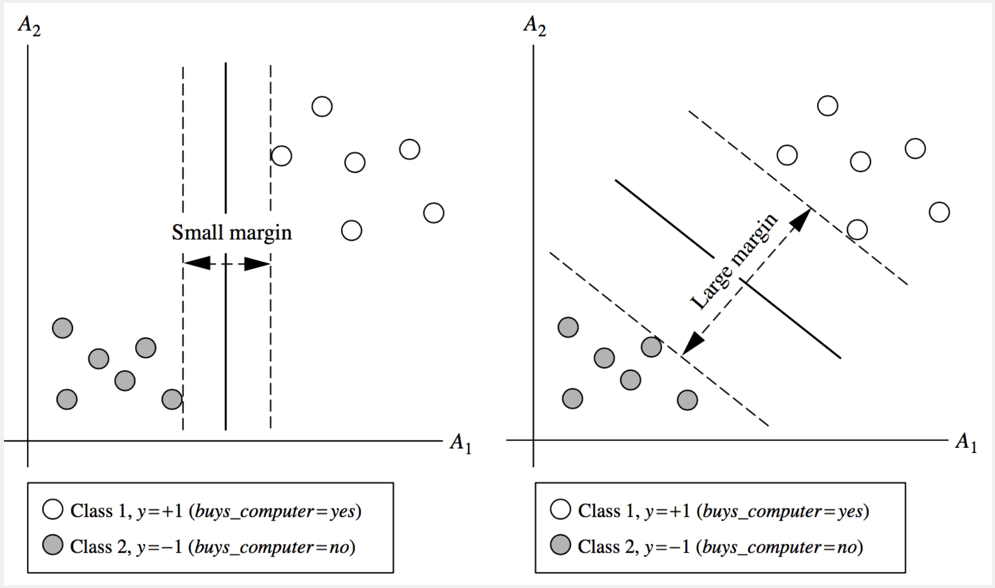
A_2
A_1
Class 1, $y=+1$ (*buys_computer=yes*)
Class 2, $y=-1$ (*buys_computer=no*)

- Because **a straight line (hyperplane) can be drawn to separate all the tuples** of class +1 from all the tuples of class -1.
- There are **infinite lines** (hyperplanes) separating the two classes
  - e.g., all of the dotted lines separate the training tuples exactly the same in the above example.
- We want to find the best one (the one that minimises classification error on unseen data)

Maximum marginal hyperplane

SVM searches for the hyperplane with **the largest margin**, i.e., maximum marginal hyperplane (**MMH**)

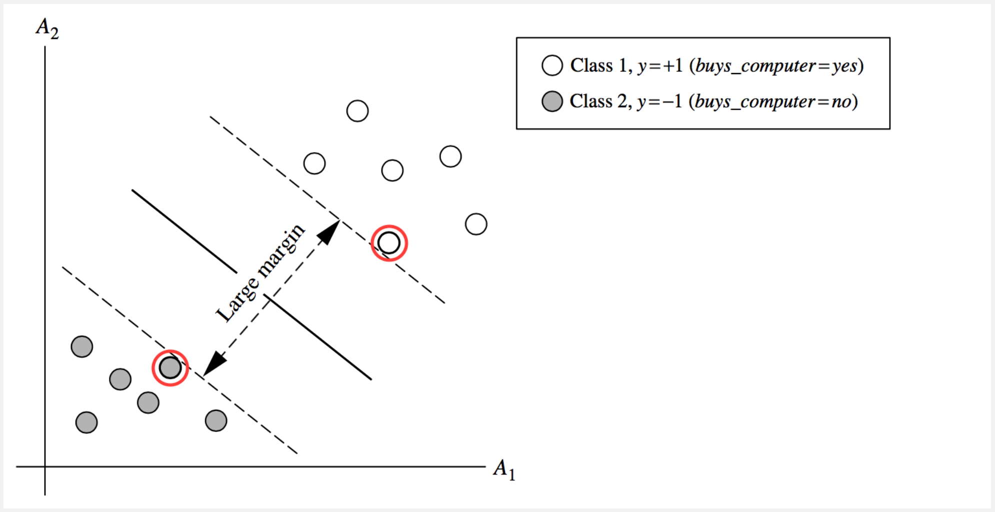
- **Margin**: Draw a perpendicular line from the hyperplane to a tuple. The distance between the hyperplane and the tuple is the margin of that hyperplane.



A_2
Small margin
A_1
Class 1, $y=+1$ (*buys_computer=yes*)
Class 2, $y=-1$ (*buys_computer=no*)
A_2
Large margin
A_1
Class 1, $y=+1$ (*buys_computer=yes*)
Class 2, $y=-1$ (*buys_computer=no*)

In this example, the hyperplane on the right figure has a larger margin than the one on the left.

Support Vectors:



A_2
A_1
Class 1, y=+1 (*buys_computer*=*yes*)
Class 2, y=−1 (*buys_computer*=*no*)
Large margin

- **Support vectors:** the training tuples that determine the largest margin hyperplane. In the above example, red-circled tuples are the support vectors of the hyperplane.

Formal definition of hyperplanes and support vectors:

Two dimensional training tuple case:

- In two dimensional space ($A_1 - A_2$ plane), a hyperplane corresponds to a line, and every hyperplane can be written as:
  - $A_2 = a \times A_1 + b$
- For a more general representation, if we replace A_1 and A_2 by x_1 and x_2, then the above hyperplane can be rewritten as:
  - $0 = w_1 \times x_1 + w_2 \times x_2 + w_0$,
  - where $w_1 = a, w_2 = -1, w_0 = b$.
  - We can represent any hyperplane(line) in two dimensional space with w_1, w_2, and w_0.
- In the linearly separable case, every training tuple satisfies the following condition:
  - H1 (positive class)
    - If $w_1 \times x_1 + w_2 \times x_2 + w_0 \geq +1$
  - H2 (negative class):
    - If $w_1 \times x_1 + w_2 \times x_2 + w_0 \leq -1$
- Support vector: Therefore, every training tuple that satisfies $w_1 \times x_1 + w_2 \times x_2 + w_0 = \pm 1$ is a support vector.

N-dimensional training tuple case:

- Let $X = (x_1, x_2, x_3, \ldots, x_n)$ be a training tuple with class label y_i then a separating hyperplane can be written as
  - $WX^\top + w_0 = 0$
  - where $W = w_1, w_2, \ldots, w_n$ is a weight vector and w_0 a scalar (bias)
  - $WX^\top = W \cdot X = \sum_{i=1}^{n} w_i \times x_i$
- The hyperplane defining the sides of the margin:
  - H1: $w_0 + WX^\top \geq 1$ for $y_i = +1$, and
  - H2: $w_0 + WX^\top \leq -1$ for $y_i = -1$
- These two equations can be combined into one equation:
  - $y_i(w_0 + WX^\top) \geq 1 \quad \forall i$
  - This equation can be solved as a constrained (convex) quadratic optimisation problem that maximises the margins to estimate the weights W from the training set, and is the SVM version of *training the model.*

Classify test tuple using trained model:

During the testing phase, the trained model classifies a new tuple X using the rules:

- Using hyperplane
  - H1 (positive class)
    - If $w_0 + WX^\top \geq 0$
    - Then X will be classified as a positive class
  - H2 (negative class):
    - If $w_0 + WX^\top < 0$
    - Then X will be classified as a negative class
- **Alternatively**, we can use the support vectors X_i, each with class label y_i, to classify test tuples. For test tuple X,
  - $d(X) = \sum_{i=1}^{\ell} y_i \alpha_i X_i X^\top + b_0$
  - where ℓ is the number of support vectors, and α and b_0 are automatically determined by the optimisation/training algorithm.
  - If the sign of $d(X)$ is positive then X is classified as H1, otherwise H2.
  - Note that we need to **keep only the support vectors** for testing
    - This fact will be used in the non-linearly separable case

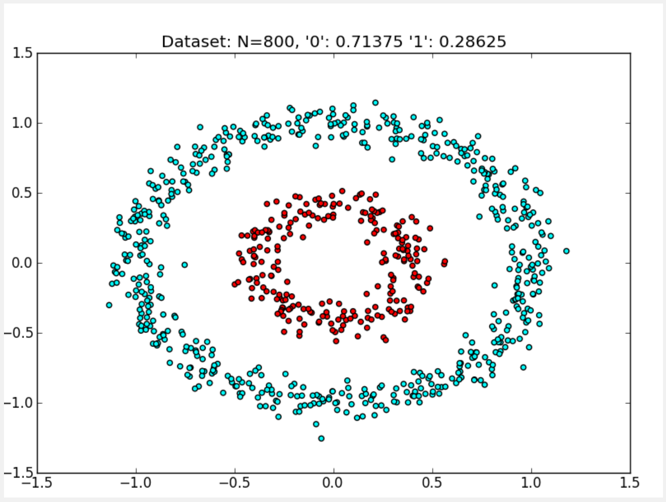
Why Is SVM effective on high-dimensional data?

- The **complexity** of a trained classifier is characterised by the number of support vectors rather than the dimensionality of the data
- The **support vectors** are the essential or critical training examples —they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found from the support vectors alone
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalisation, even when the dimensionality of the data is high

2.2. Linearly Inseparable Case

So far we've discussed the case where there is a straight line that separates two classes in the training dataset. Now, we will discuss the case when the data are not linearly separable.

Example of linearly inseparable data

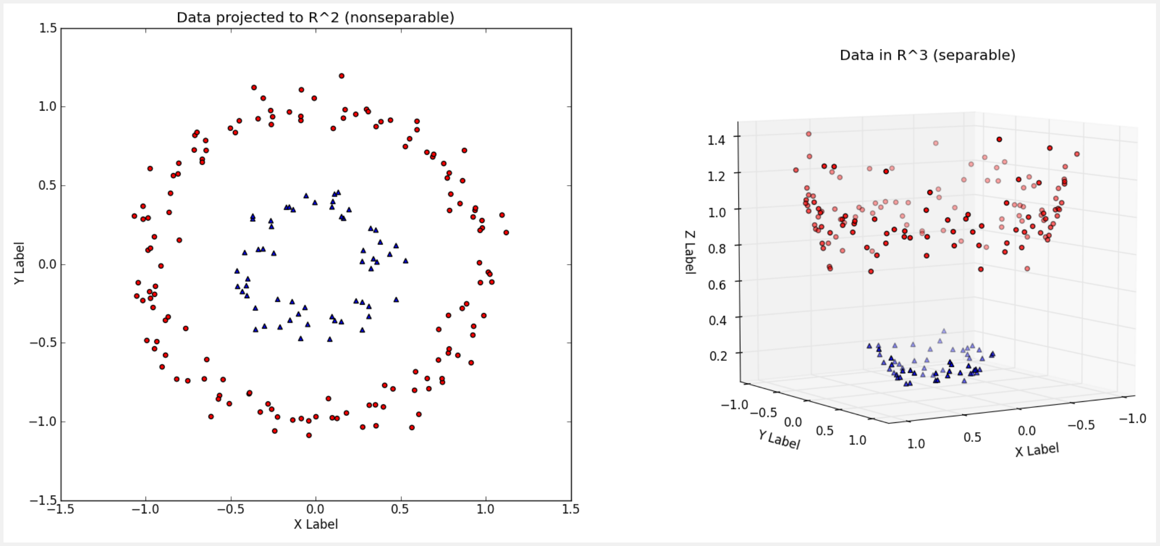


> A two-class dataset that is not linearly separable. The outer ring (cyan) is class '0', while the inner ring (red) is class '1'

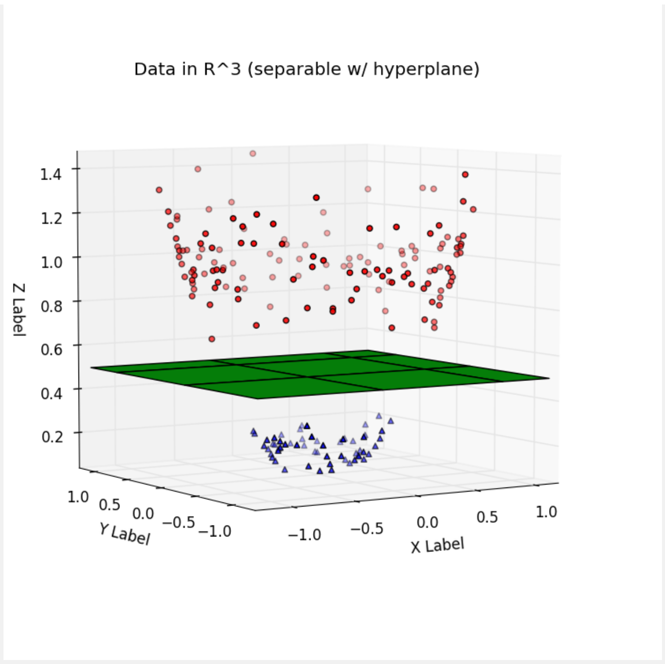
- Linearly inseparable training data. Unlike the linearly separable data, it is not possible to draw a **straight (linear) line** to separate the classes.
- Basic SVM would not be able to find a feasible solution here.
- But there is a way to extend the linear approach in this case!

Kernel Trick

The idea is to obtain linear separation by **mapping the data to a higher dimensional space**. Let's see the example first:



> (Left) A dataset in $\mathbb{R}^2$, not linearly separable. (Right) The same dataset transformed by the transformation:
$[x_1, x_2] = [x_1, x_2, x_1^2 + x_2^2]$



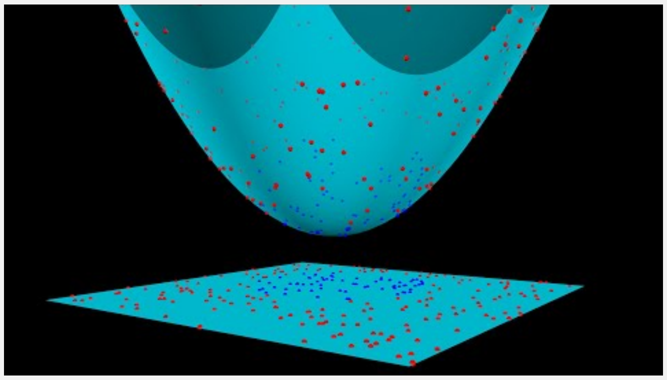
> Hyperplane (green plane) that **linearly separates two classes** in the higher dimensional space.

In the above example, we can train a linear SVM classifier that **successfully finds a good decision boundary in** $\mathbb{R}$.

However, we are given the dataset in $\mathbb{R}^2$. The challenge is to **find a transformation** $\phi : \mathbb{R}^2 \to \mathbb{R}^3$, such that the transformed dataset is linearly separable in $\mathbb{R}^3$.

$\phi([x_1, x_2]) = [x_1, x_2, x_1^2 + x_2^2]$, which after applied to every point in the original tuples yields the linearly separable dataset.

ACTION: Watch this video visualisation of polynomial kernel.



Assuming we have such a transformation ϕ , the new classification pipeline is as follows.

- First transform the training set X to X' with ϕ .
- Train a linear SVM on X' to get a new SVM.
- At test time, a new example $\bar{x}$ will first be transformed to $\bar{x}' = \phi(\bar{x}')$ during the testing time.

This is exactly the same as the train/test procedure for regular linear SVMs, but with an added data transformation via ϕ.

We have improved the **expressiveness** of the Linear SVM classifier by working in a higher-dimensional space.

Kernels

- The decision rule used in the linearly separable case, for test vector X was
  - $d(X) = \sum_{i=1}^{\ell} y_i \alpha_i X_i X^\top + b_0$
- Now it is converted, in the higher-dimensional space to
  - $d(X) = \sum_{i=1}^{\ell} y_i \alpha_i \phi(X_i) \cdot \phi(X)^\top + b_0$

- But all these dot-products over potentially very high-dimensional vectors are expensive.

- We define a kernel function K(.,.) on the data in the original lower-dimensional space:
  - $K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j)^\top$
- Now, every $\phi(X_i) \cdot \phi(X_j)^\top$ can be replaced by $K(X_i, X_j)$ in the training algorithm and decision rule. We do not need to make calculations over the higher-dimensional transformed vectors. We do not even need a formulation of ϕ, as a kernel function is enough for determining the SVM. But we do need the kernel function to have certain properties, so that not just any function will do.
- Here are some widely used kernel functions. They do result in different classifiers, although they are commonly of similar accuracy and the only way to find the best one is to try it.

  - Polynomial kernel of degree h: $K(X_i, X_j) = (X_i \cdot X_j^\top + 1)^h$
  - Gaussian radial basis function kernel: $K(X_i, X_j) = e^{-dist(X_i, X_j)^2/2\sigma^2}$
  - Sigmoid Kernel: $K(X_i, X_j) = tanh(\kappa X_i \cdot X_j^\top - \delta)$
  - Linear Kernel: $K(X_i, X_j) = X_i \cdot X_j^\top$

ACTION: Watch this video if you'd like to know more details about learning SVM

Support Vector Machines by Patrick Winston(MIT)

Some of the examples used in this note are from http://www.eric-kim.net/

2.3. Neural Network vs. SVM

Comparison between neural network and SVM

Decision hyperplanes found by non-linear SVM are similar to those found by neural network classifiers. e.g. SVM with Gaussian radial basis function is the same as a radial basis function neural network. e.g. A sigmoid kernel SVM is the same as a two-layer neural network (i.e with no hidden layers).

Neural Network	SVM
Nondeterministic algorithm which finds local minima Generalises well but doesn't have strong mathematical foundation Can easily be learned in incremental fashion To learn complex function, use multi-layer neural network, but adding more layers adds more training time Gets more complex with higher dimensions	Deterministic algorithm that finds the global minimum Nice generalisation property Hard to learn - learned in batch mode using quadratic programming techniques Using kernels can learn very complex functions Well suited to high-dimensional data

3. Practical Exercises: Support Vector Machines in Rattle

ACTION: Attempt these practical exercises with Rattle. There is a video showing the mechanics to get you started, written instructions for you to work through, and separately some suggested solutions.

Lab: Support Vector Machines in Rattle

Solution to Lab: Support Vector Machines in Rattle

4. Lazy Learners (Text: 9.5)

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 discussed methods so far): 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

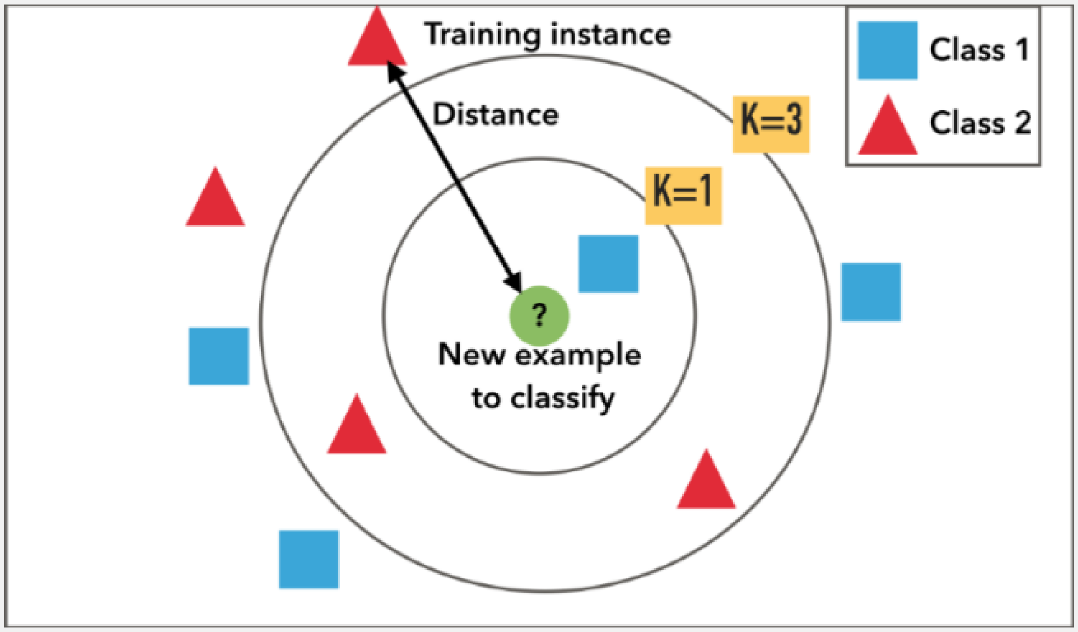
Lazy Learner: Instance based method

- 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 (KNN)
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

4.1. kNN-Classifier (Text: 9.5.1)

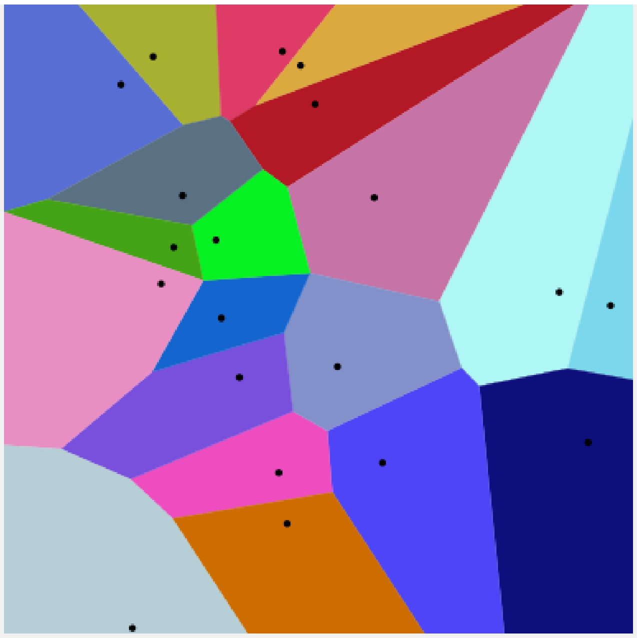
K-Nearest Neighbourhood (k-NN)

- kNN classifiers are based on learning by analogy
- A training tuple described by *n* attributes represents a point in the n-dimensional space
    - All training tuples are stored in an n-dimensional space
- Given a tuple of unknown class or unknown target value, KNN classifier searches k nearest neighbours of the unknown tuple.
    - The nearest neighbours are defined in terms of Euclidean distance or other metrics, $dist(X_1, X_2)$



- Two ways of classifying the unknown tuple in kNN
    - Discrete method (discrete-valued method)
        - k-NN returns the most common value among the k training examples nearest to X_q (test tuple)
        - Decision function:
            - $D(X_q) = \sum_{i=1}^{k} y_i$
            - where $y_i \in \{+1, -1\}$ is the class of ith nearest neighbour
            - If $D(X_q) > 0$ then X_q is positive class otherwise negative class

        - Voronoi diagram: the decision surface induced by 1-NN for a typical set of training examples


Example of Voronoi diagram with Euclidean distance. The lines represent a decision surface induced by 1-NN. from Wikipedia

- Continuous method (real-valued prediction):
    - Returns the mean values of the k nearest neighbours
    - Distance-weighted nearest neighbour algorithm
        - Weight the contribution of each of the k neighbours according to their distance to the query X_q
        - Give greater weight to closer neighbours
          $w_i = \frac{1}{dist(X_q, X_i)^2}$
    - Decision function:
        - $D(X) = \sum_{i=1}^{k} \frac{w_i}{\sum_{j=1}^{k} w_j} \cdot y_i$
        - where $y_i \in \{+1, -1\}$ is the target value of ith nearest neighbour

Characteristics of KNN

- **Robust** to noisy data by averaging k-nearest neighbours
- **Extremely slow** when classifying test tuples
    - With $|D|$ training tuples, $|D|$ comparisons are required to find k-nearest neighbourhood
    - For example, SVM only requires ℓ comparisons where ℓ is the number of support vectors
    - Partial distance method:
        - Compute a distance on a subset of n attributes.
            - If the distance exceeds a threshold, further distance computation will be halted
            - Otherwise keep computing the distance on the remaining attributes

4.2. Exercise: kNN Classifier

ACTION: Try the following exercise.

Example: KNN Classifier

Solution:

Solution to Exercise: KNN-Classifier

4.3. Case-Based Reasoning (CBR) (Text: 9.5.2)

- CBR(Case-Based Reasoning): Uses a database of problem solutions to solve new problems
- Store symbolic description (tuples or cases)—not points in a Euclidean space
- Applications: Customer-service (product-related diagnosis), legal ruling
- Methodology
    - Instances represented by rich symbolic descriptions
    - If there is an identical training case, given a test case, the solution of the training case will be returned
    - If not, search for similar cases, multiple retrieved cases may be combined
    - Tight coupling between case retrieval, knowledge-based reasoning, and problem solving
- Challenges
    - Find a good similarity metric
    - Indexing based on syntactic similarity measure, and when failure, backtracking, and adapting to additional cases

5. Variants of Classification (Text 9.7)

Multiclass-Classification

- Classification involving more than two classes (i.e., > 2 Classes)
- Method 1. **One-vs.-all** (OVA): Learn a classifier one at a time
    - Given m classes, train m classifiers: one for each class
    - Classifier j:treats tuples in class j as positive & all others as negative
    - To classify a tuple X, the set of classifiers vote as an ensemble. If classifier j predicts the positive class, then class j gets one vote. If classifier j predicts the negative class then all non-j classes get one vote.
- Method 2. **All-vs.-all** (AVA): Learn a classifier for each pair of classes
    - Given m classes, construct $m(m-1)/2$ binary classifiers
    - A classifier is trained using tuples of the two classes
    - To classify a tuple X, each classifier votes. X is assigned to the class with maximal vote
- Comparison
    - All-vs.-all tends to be superior to one-vs.-all
    - Problem: Binary classifier is sensitive to errors, and errors affect vote count

Semi-supervised Classification

- Semi-supervised: Uses labeled and unlabeled data to build a classifier
- **Self-training**:
    - Build a classifier using the labeled data
    - Use it to label the unlabeled data, and those with the most confident label prediction are added to the set of labeled data
    - Repeat the above process
    - Advantage: easy to understand; disadvantage: may **reinforce errors**
- **Co-training**: Use two or more classifiers to teach each other
    - Use two disjoint and independent selections of attributes of each tuple to train two good classifiers, say f_1 and f_2
    - Then f_1 and f_2 are used to predict the class label for unlabeled data tuples X
    - Teach each other: The tuples in X having the most confident prediction from f_1 are added to the set of labeled training data for f_2, & vice versa
    - Retrain two classifiers using the extended training sets, using the same disjoint attribute selections

Active-Learning

- Class labels are expensive to obtain
- Active learner: **query human (oracle) for labels**
- Pool-based approach: Uses a pool of unlabeled data
    - L: a small subset of D is labeled, U: a pool of unlabeled data in D
    - Use a query function to carefully select one or more tuples from U and request labels from an oracle (a human annotator)
    - The newly labeled samples are added to L, and learn a model
    - Goal: Achieve high accuracy using as few labeled data as possible
- Evaluated using learning curves: Accuracy as a function of the number of instances queried (# of tuples to be queried should be small)
- Research issue: How to choose the data tuples to be queried?
    - Uncertainty sampling: choose the least certain ones
    - Reduce version space, the subset of hypotheses consistent with the training data
    - Reduce expected entropy over U: Find the greatest reduction in the total number of unlabelled data

Transfer-Learning

- Transfer learning: Build classifiers for one or more similar source tasks and apply to a target task
- vs Traditional learning: Build a new classifier for each new task

Want to know more?

COMP8420/**COMP4660** **Neural Networks, Deep Learning and Bio-inspired Computing** covers, in much more depth, neural, deep learning, fuzzy, evolutionary and hybrid methods.

COMP8600/**COMP4670** **Statistical Machine Learning** covers, in much more mathematical depth, Bayesian learning, regression, neural networks, and support vector machines.