CLASSIFICATION ALGORITHMS ON MULTIDIMENSIONAL DATASET

July 6, 2019

Georgios Michas (sdi1400109)
Maria - Despoina Siampou (sdi1600151)

National and Kapodistrian Universiry of Athens
Department of Informatics and Telecommunications

Contents

0.1 Introduction

Having Wisconsin Breast Cancer Dataset, we want to create and validate three different classification models:

1. K-Nearest Neighbor (Knn)
2. Decision Tree
3. Support Vector Machine (SVM)

The dataset has 699 entries and 2 classes:

1. Benign (person is not sick, 458/699 (65.5%))
2. Malignant (person is sick, 241/699 (34.5%))

Each entry in this dataset is a 11d - vector with the following format :

Attribute	Domain
Sample code number	id number
Clump Thickness	1 - 10
Uniformity of Cell Size	1 - 10
Uniformity of Cell Shape	1 - 10
Marginal Adhesion	1 - 10
Single Epithelial Cell Size	1 - 10
Bare Nuclei	1 - 10
Bland Chromatin	1 - 10
Normal Nucleoli	1 - 10
Mitoses	1 - 10
Class	(2 for Benign, 4 for Malignant)

This report will be an introduction to the classification algorithms used as well as a presenation of our results.

0.2 K - Nearest Neighbours (KNN)

1. **Definition**
   The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification. The input consists of the k closest training examples in the feature

space. The output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k=1, then the object is simply assigned to the class of that single nearest neighbor. k-NN is a type of lazy learning, where the function is only approximated locally and all computation is deferred until classification. A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

2. **Algorithm**
   The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables Hamming distance can be used.

3. **Complexity**
   KNN search has O(N) time complexity for each query where N = Number of data points. For KNN with K neighbor search, the time complexity will be O(log(K)*N), if we maintain a priority queue to return the closest K observations.

4. **K-NN and KD-Trees**
   The KD tree is a binary tree structure which recursively partitions the parameter space along the data axes, dividing it into nested orthotropic regions into which data points are filed. The construction of a KD tree is very fast: because partitioning is performed only along the data axes, no D-dimensional distances need to be computed. Once constructed, the nearest neighbor of a query point can be determined with only O(log(N)) distance computations. This method, mainly, attempts to reduce the required number of distance calculations by efficiently encoding aggregate distance information for the sample. The basic idea is that if point A is very distant from point B, and point B is very close to point C, then we know that points A and C are very distant, without having to explicitly calculate their distance. In this way, the computational cost of a nearest neighbors search can be reduced being proportional to the dimensions.

5. **Curse of dimensionality**
   Though the KD tree approach is very fast for low-dimensional neighbors searches (D $<$ 20), it becomes inefficient as D grows very large. In this case running a fast approximate k-NN search using locality sensitive hashing (LSH) might be the only feasible option.

0.3 Decision Trees

1. **Definition**
   Decision trees are the most developed methods for partitioning sets of items into classes. A decision tree classifies data items into a finite number of predefined classes. The tree nodes are labeled with the names of attributes, the arcs are labeled with the possible values of the attribute, and the leaves are labeled with the different classes.

2. **Algorithm**
   Given a set C of items, grow a decision tree is grown using the divide-and-conquer algorithm as follows:

   (a) If all the items in C belong to the same class or C is small, the tree is a leaf labeled with the most frequent class in C.

   (b) Otherwise, a test based on a single attribute with two or more outcomes is chosen. Make this test the root of the tree with one branch for each outcome of the test.The same procedure is applied recursively for each subset of C $(C_1, C_2, ..., C_n)$.

   (c) In last step two heuristic criteria are used to rank possible tests: information gain, which minimizes the total entropy of the subsets C_i, and the default gain ratio that divides information gain by the information provided by the test outcomes.

3. **Entropy**
   A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous). If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one. Entropy's type by using the frequency table of one

attribute is : E(X) = $\sum_{i=1}^{c} -p_i log(p_i)$. Furthermore, entropy's type by using the frequency table of two attributes is: E(T,X) = $\sum_{c \in X} P(c)E(c)$.

4. **Information Gain**
   Constructing a decision tree is all about finding attribute that returns the highest information gain. The information gain is based on the decrease in entropy after a dataset is split on an attribute and it comes of the relationship: $Gain(T,X) = Entropy(T) - Entropy(T,X)$.

5. **Decision Trees and Dimensionality**
   Decision Tree operates by rejecting sequentially classes until the correct class is found. In other words, the correct class corresponding to a feature vector, is determined by searching a tree-based decision system. The feature space is divided into regions corresponding to the different classes. For example, a 2-d (binary) Decision Tree divides the search space into hyperrectangles with sides parallel to the axis. The tree is searched in a sequential manner and a decision of the form $x_i \leq \alpha$, with x_i being a feature and α a threshold value, is made at each node for individual features. This processing scheme is an essential part of many tree-based vector quantization algorithms.

6. **Curse of dimensionality**
   Decision tree suffers from the curse of dimensionality, too. As the sample space increases, the distances between data points increases, making it much harder to find an optimal split.

0.4 Support Vector Machine (SVM)

1. **Definition**
   Support Vector Machine (SVM) is widely used in classification objectives. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

2. **Linear Classification**

   Support the training data point each belongs to two classes and the objective is to decide which class is responsible for a new data point. In support vector machine a data point is viewed as a p dimensional vector and separate data point with a $(p-1)$ dimensional hyper plane is known a linear classification. The steps of linear classification are described below:

   (a) Assume the training data of D with a set of n points of calculating by $D = \{(x_i, y_i) | x_i \in R\ p, y_i \in \{-1,1\}\} i = 1n,$ where the y_i is either 1 or -1, indicating the class to which the point x_i belongs. Each x_i is a P-dimensional real vector. We want to find the maximum-margin hyper plane that divides the points having y_i=1 from those having y_i=-1.

   (b) Any hyper plane can be written as the set of points X satisfying following $W \cdot X - b = 0$, where $\cdot$ denotes the dot product and W denotes the normal vector of hyper plane. The parameter $\frac{b}{\|W\|}$ determines the offset of the hyper plane from the origin along the normal vector W.

   (c) For maximization distance in point for linearly separable data the margin is set. The set margin hyper plane is described as $W \cdot X - b = 1$ and $W \cdot X - b = -1$. Geometrically, the distance between these two hyper planes is $\frac{2}{\|W\|}$, so to maximize the distance between the planes we want to minimize $\|W\|$.

   (d) To prevent the data points to fall into the margin, the following constraint is added: for each i either $W \cdot X - b \geq 1$ for x_i of first class or $W \cdot X - b \leq 1$ for x_i of second class.

   (e) The final equation is written as $y_i(w \cdot x_i - b) \geq 1,\ \forall 1 \leq i \leq n.$

   (f) The extended extension of maximum-margin classifier which provides a solution to the above mentioned problem is given as $min \frac{1}{2} \|W\|^2,$ $s.t.\ y_i(w \cdot x_i - b) \geq 1,\ \forall 1 \leq i \leq n.$

3. **Non-Linear Classification**

   Non-linear classification is based on kernel function. The resulting algorithm and linear classification are formally same, except that every dot product is replaced by a non-linear kernel function. This allows the algorithm to fit the maximum-margin hyper plane in a transformed feature space. The transformation may be non-linear and the transformed space high dimensional; thus though the classifier is a hyper plane in the high-dimensional feature space, it may be non-linear in the original

input space.
If the Gaussian radial kernel function used as the corresponding feature space is a Hilbert space for infinite dimensional. This classification shown that higher dimension increases the generalization error, although the amount of feature space is bounded.

4. **Kernel Functions**
   The idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. The kernel function plays a critical role in SVM and its performance: $K(x,x') = \langle \phi(x), \phi(x') \rangle$
   Kernel function defines inner product in the transformed space, depending on K. Some Kernel Functions are the following:
   (a) Gaussian or Radial Basis Function (RBF) kernel, default for one-class learning: G(x_i, x_j) = exp(-$\|x_i - x_j\|^2$)
   (b) Linear kernel, default for two-class learning: G(x_i, x_j) = $x_i' x_j$
   (c) Polynomial kernel: G(x_i, x_j) = $(1+\mathrm{x}_i' x_j)^p$

0.5 Results

To test how well our models are trained upon a given data and test it on unseen data, we used k-fold cross validation. The procedure of k-fold, has a single parameter called k that refers to the number of groups that a given data sample is to be split into. So, k-fold randomly divides the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fitted on the remaining k-1 folds. This procedure is repeated k times; each time, a different group of observations is treated as a validation set.

Regarding the following computations, high cost was assigned to the misidentification of malignant as benign and low cost to the opposite situation. These assignments were made in order to achieve almost 100% sensitivity.

As a result, running the above algorithms with the same k-fold validation split and costs [0 1; 4 0] and no cost respectively, gave us the following results:

Algorithm	Accuracy	Sensitivity	Specificity	Exec Time (secs)
KNN	97.43%	99.59%	96.29%	0.09811
SVM	97.00%	99.59%	95.63%	0.1259
DTree	94.56%	95.85%	93.89%	0.0883

Algorithm	Accuracy	Sensitivity	Specificity	Exec Time (secs)
KNN	96.71%	95.43%	97.38%	0.0994
SVM	97.00%	97.93%	96.50%	0.1215
DTree	93.84%	90.46%	95.63%	0.0956

0.6 Conclusion

Regarding to above results, it can be easily seen that KNN and SVM are more suitable for this problem. KNN performs really well due to the use of KD Tree. SVM ,also, is very accurate, more robust due to optimal margin gap between separating hyper planes, - which helps making better predictions with test data - and more efficient when using Kernel functions. Thus it is faster in training, better in accuracy with stability/robustness. On the other hand, Decision Tree may runs fast but it lacks performance.