Summary

Introduction to Machine Learning (WBCS032-05)

Bjorn Pijnacker

2021–2022 | 1a

Contents

Chapter 1

Introduction & Supervised and Unsupervised Learning

1.1 Machine Intelligence

How do we define when a machine is intelligent or when it thinks? Intelligence has many different definitions. In computing, it deals with perception, decision making, and learning. Machine learning, in essence, is adaptive decision-making based on analysis of data. The three main branches are unsupervised learning, supervised learning, and reinforcement learning. We focus on the prior two.

In unsupervised learning we aim to define a cost function that we can optimize in the process of learning. There is no feedback from the environment into the learning process in unsupervised learning. The challenges in unsupervised learning are coming up with a criterion, deciding the number of clusters or groups, and defining similarity and dissimilarity for datapoints.

Data-driven supervised learning can use regression, where we try to predict some (continuous) quantity. If we have some function $\alpha x_1 + \beta x_2 > \gamma$ we can optimize α, β, γ using real-world data. This data may originate from observations or measurements. Some issues in supervised learning include how complex to make the hypothesis (how many variables in the function), how to parameterize our hypothesis, how to handle noisy data, the optimization of the generalization ability, and how to test and validate our solution.

Outside these three main branches of learning also exist many other forms of machine learning that are adaptions of the above.

Chapter 2

Linear Vector Quantization

We consider classification problems which in context of supervised learning. We have assignment of data to one of several classes (crisp/soft). We have reference data that we can compare to. The classification is done using some distance measure. Data is—if possible—always represented as vectors of numerical features.

2.1 K-Nearest Neighbor Classifier

We start with a simple distance-based classifier: K-nearest neighbor classifier. Given a set of points with known labels, the problem we wish to solve is the label of a new point with unknown (to us) label. Finding this label is called a query.

We classify a query according to the the label of the nearest neighbor or the majority of the K nearest neighbors. This yields a local decision boundary according to Euclidean distances.

This is conceptually very simple and requires no training at all. There is only a single parameter to optimize: K. This method is, however, very expensive on storage and computation, sensitive to outliers, and may result in overly complex decision boundaries.

2.1.1 Prototype based classification

We propose a solution: prototype based classification. In this case, we represent the data by one or several prototypes per class of data. We then classify a query according to the label of the nearest prototype. This is defined more formally later.

This method less sensitive to outliers and results in far less complex decision boundaries. It also has lower storage needs, and less computational effort when classifying. A downside to this method is that a training phase is required to construct the prototypes and we introduce a model selection problem: the number of prototypes per class.

Nearest Prototype Classifier (NPC) The NPC is defined as follows:
Given a set of prototypes

$$\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^K \qquad \mathbf{w}^k \in \mathbb{R}^N$$

which carry class labels

$$S^1, S^2, \ldots, S^K \qquad S^k = S(\mathbf{w}^k) \in \{1, 2, \ldots, C\}$$

and a datapoint $\mathbf{x} \in \mathbb{R}^N$ we determine the winner $\mathbf{w}^*$ to be

$$\mathbf{w}^* = \underset{\mathbf{w}^j}{\operatorname{argmin}} \{d[\mathbf{w}^j, \mathbf{x}]\}_{j=1}^K$$

where d is our distance function. The winner is more intuitively defined as the closest (based on d) prototype to our point $\mathbf{x}$.

2.2 Learning Vector Quantization

LVQ is an algorithm that allows us to determine prototypes for a dataset. The heuristic scheme used is called LVQ1. It works in a few steps: (1) initialize prototype vectors for different classes; (2) present a single example; (3) identify the winner; (4) move the winning prototype closer towards the data if the class of the data equals that of the prototype, and further away otherwise.

The initial prototypes are chosen by taking random datapoints. The winner is chosen using the nearest prototype classifier. The update step is then defined as

$$\mathbf{w}^* \leftarrow \mathbf{w}^* + \eta_w \, \psi(S^*, \sigma^m) \, (\mathbf{x}^m - \mathbf{w}^*)$$

where

$$\psi(S, \sigma) = \begin{cases} +1 & S = \sigma \\ -1 & \text{otherwise} \end{cases}$$

and η_w is the learning rate.

A variation of LVQ1 is Generalized LVQ. Generalized LVQ updates two prototypes at the same time: the closest wrong and the closest right one. The closest wrong prototype is moved away, and the closest right prototype is moved closer to the datapoint we are considering.

Chapter 3

Regression

Regression is another type of supervised learning we consider. In regression, we have some input parameters and some output that we would like to predict. We do this using a mathematical or statistical model (e.g. $m = \gamma(a^\alpha \cdot b^\beta)$ where we tune α, β, γ).

3.1 Linear Regression

Linear regression tries to fit a linear function to the parameters to produce a prediction of the output. We are given some $\mathbb{D} = \{\mathbf{x}^\mu, y^\mu\}_{\mu=1}^P$ where $\mathbf{x}^\mu \in \mathbb{R}^N$ are our N-dimensional input vectors, and $y^\mu \in \mathbb{R}$ are our target values. We define a hypothesis linear relations as $f_H(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x}$ where $\mathbf{w}$ is a vector of coefficients.

To evaluate how well our model works we need an error value E. In this case we define the mean squared error (MSE) as

$$E = \frac{1}{2}\sum_{\mu=1}^{P}\left[\mathbf{w}^\mathsf{T}\mathbf{x}^\mu - y^\mu\right]^2$$

To obtain the values of $\mathbf{w}$ we use the pseudo-inverse of a matrix. We first define matrices Y and X as follows:

$$Y = \left[y^1, y^2, \ldots, y^P\right]^\mathsf{T} \qquad \in \mathbb{R}^P$$
$$X = \left[\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^P\right]^\mathsf{T} \qquad (P \times N)$$

We then find

$$\mathbf{w}^* = [X^\mathsf{T}X]^{-1}X^\mathsf{T}Y$$

In this case, $[X^\mathsf{T}X]^{-1}X^\mathsf{T}$ is the pseudo-inverse of matrix X.

3.1.1 Machine learning "vs." statistical learning

The concepts of *Maximum Likelihood* and *Maximum a Posteriori* could be part of the exam, but I don't understand the statistical modeling personally.

3.1.2 Gradient Descent

Given our error function $E(\mathbf{w})$ as defined above, we have a gradient $\nabla_w E$ that gives us the gradient of $\mathbf{w}$ in E. We can use this to do numerical minimization using gradient descent. If we have the sequence $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla E|_{\mathbf{w}_t}$ we approach *some* minimum of E with the rate set by the learning rate η.

> **The learning rate η** The learning rate controls the step size of algorithms. We need it to be small enough to ensure convergence, but large enough to facilitate as-fast-as-possible learning.
>
> If our η is a good value, then we have smooth convergence. If it's an "intermediate" value we get oscillations around minimum, but convergence nonetheless. If it is too big, our vector $\mathbf{w}$ diverges and we get no solution.

Chapter 4

Validation

Key problems in machine learning include model selection, data representation, and determining algorithm parameters. An important aspect in this is testing the generalization performance of our trained model.

One way we propose to do this by splitting our dataset (randomly) into n disjoint set. All but one of these sets are then used for training, and the other is used for validation. We repeat this training n times, where we differ which set of data is used for validation every time. We calculate both the average training and test errors (and their variances).

In n-fold cross validation, the plots of the training/test errors w.r.t. some parameter can show us clearly which parameter value is a good choice. We will often see both the training and test error start off at some "bad" value, then both improve. Later on, while the training error will keep improving, the test error will worsen. This is due to overfitting, where our model is too specific to the data used to train, and it doesn't generalize well.

> **The Bias/Variance Dilemma** The bias/variance dilemma talks about competing aims in training: low bias, that is, a small deviation from the "true solution"—on average; vs. low variance, that is, weak dependence on the actual training set and robustness of the hypothesis.

Chapter 5

Unsupervised Vector Quantization

For unsupervised vector quantization, the aim is the same as that of learning vector quantization, that is, we wish to represent our data by a few prototype vectors while keeping the dimensionality.

The algorithm named unsupervised competitive learning is the same as LVQ1, except there is no labels. We end up with prototypes that represent the data density. One difference is how the error is calculated. With competitive VQ, the quantization error is defined as

$$H_{VQ} = \sum_{j=1}^{K} \sum_{\mu=1}^{P} \left(\xi^{\mu} - \mathbf{w}_j\right)^2 \prod_{k \neq j}^{K} \Theta\left(d_k^{\mu} - d_j^{\mu}\right)$$

that is, the sum of the distances (squared) to the winning prototypes for all points. The lower this is, the closer all data is to the prototypes.

This clustering metric H_{VQ} may not always lead to the best results. Sometimes we have "intuitive" clusters that yield a higher H_{VQ} than some other set of prototypes. Similarly, a higher number of prototypes will almost always lead to a lower H_{VQ} so we need to be wary of this. A good method for determining how many clusters are needed is by using the elbow method. We plot H_{VQ} against the number of clusters, and where we see an elbow in the plot is the amount of clusters that we want.

Another algorithm is the K-means algorithm. For this algorithm there is three steps:

(1) **Initialization:** place K vectors on random data points

(2) **Assignment of data to centers:** assign every data point to the nearest center (a.k.a. prototypes) and let this define the current clustering

(3) **Re-compute centers:** compute the means of the K current clusters as new prototypes

We iterate back and forth between step 2 and 3.

Chapter 6

Clustering & Outlier Detection

6.1 Clustering Analysis

Clustering analysis is to partition a set of data points into groups such that points within those groups are more similar to those inside the group than those in other groups. It is part of exploratory data analysis.

6.2 Hierarchical Clustering

Hierarchical clustering is a method of clustering with builds a hierarchy of clusters. We start with all data points in their own cluster. We then merge the closest two clusters into a single one. The distance between two clusters is defined later on. We keep merging clusters until we only have the desired number of clusters left.

We can visualize this clustering in a dendrogram. A dendrogram has points on the x-axis and the number of clusters on the y-axis. It shows an inverted U shape for each merging that happens, slowly building up the dendrogram.

The process described above is agglomerative (bottom up) cluster. Its inverse is divisive (top down) clustering, where the data points start in one cluster and it keeps splitting into subsets.

6.2.1 Linkage measures

Linkage measures define the distance between two clusters in a unique way. We consider 5:

Centroid linkage

The distance between the center of two clusters, defined as

$$D(C_1, C_2) = D\left(\left(\frac{1}{|C_1|}\sum_{\mathbf{x}_i \in C_1} \mathbf{x}_i\right), \left(\frac{1}{|C_2|}\sum_{\mathbf{x}_j \in C_2} \mathbf{x}_j\right)\right)$$

Single linkage

The distance between the closest two points in either cluster, defined as

$$D(C_1, C_2) = \min_{\mathbf{x}_i \in C_1, \mathbf{x}_j \in C_2} D(\mathbf{x}_i, \mathbf{x}_j)$$

Complete linkage

The distance between the closest two points in either cluster, defined as

$$D(C_1, C_2) = \max_{\mathbf{x}_i \in C_1, \mathbf{x}_j \in C_2} D(\mathbf{x}_i, \mathbf{x}_j)$$

Average linkage

The mean distance between all possible pairs of points in either cluster, defined as

$$D(C_1, C_2) = \frac{1}{|C_1|}\frac{1}{|C_2|}\sum_{\mathbf{x}_i \in C_1}\sum_{\mathbf{x}_j \in C_2} D(\mathbf{x}_i, \mathbf{x}_j)$$

Ward's (minimum variance) linkage

Doesn't return the distance but instead tells us which clusters to merge. It combines the clusters where the increase in within-cluster variance would be minimal after merging.

6.3 Evaluation metrics

After creating clusters we want to know how good they are. For this we have evaluation metrics. The first we consider is Cohesion vs. Separation. We have both the within-cluster sum of squares (WSS) and the between-cluster sum of squares (BSS). Using this we can compare different clusterings of the same data.

The second we consider is the silhouette score. This is defined using

$$S_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

$$a_i = \frac{1}{|C_i| - 1}\sum_{j \in C_i, i \neq j} d(i, j)$$

$$b_i = \min_{k \neq i} \frac{1}{C_k} \sum_{j \in C_k} \sum_{j \in C_k} d(i,j)$$

Intuitively we can see a_i as the average distance of i to the samples in its cluster, and b_i as the minimum nearest-cluster distance. If our data is badly matched, we get S_i near -1. If our data is well-separated we get S_i near 1. Summing up all the different S_i scores and dividing by the number of clusters gives us the global silhouette score.

Chapter 7

Dimensionality Reduction