Introduction to supervised learning

Walid Hadri
walid.hadri@student-cs.fr

March 14, 2021

Contents

1 Introduction

This article will be an introduction to supervised learning, the algorithms will be discussed briefly as they will be seen in details in the next lectures. The goal is to have a clear idea about what we will be dealing with when we are solving a problem that needs a supervised learning algorithm.

Let us keep in mind that this most used subbranch of Machine Learning. Every new machine learning practitioners begin their journey with supervised learning algorithms.

2 What is Supervised Learning?

Supervised learning is the machine learning task of learning a function that maps an input (X) to an output (Y) based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples afterwards. So we have a set of input-output pairs $\{(x_1, y_1), (x_2, y_2)...., (x_n, y_n)\}$ and some other inputs $x_1^*, ...x_m^*$ and we are looking for a function f to map the inputs x_i to the outputs y_i and predicts the outputs of the x_i^*. The objective of a supervised learning model is to predict the correct label for newly presented input data.

Supervised machine learning algorithms are designed to learn by example. The name "supervised" learning originates from the idea that training this type of algorithm is like having a teacher supervise the whole process.

The major steps for a supervised learning problem:

- Determine the type of the training examples
- Gather a training set that should be representative of the real-world use of the function
- Determine the input feature representation of the learned function. The number of features should not be too large, because of the **curse of dimensionality**; but should contain enough information to accurately predict the output.
- Determine the structure of the learned function and corresponding learning algorithm
- Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters.
- Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

The curse of dimensionality: it refers to when your data has too many features. Some issues arise, when we have so many features:

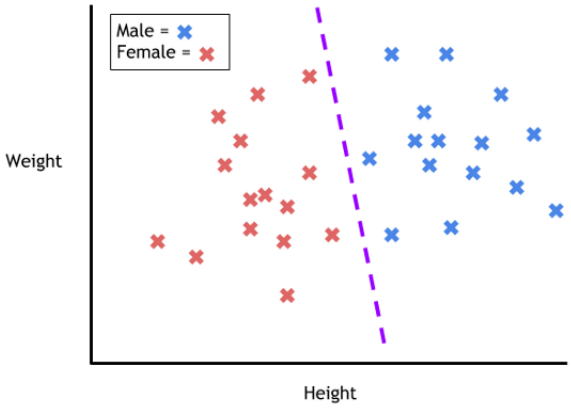
1. If we have more features than observations than we run the risk of massively overfitting our model — this would generally result in terrible out of sample performance.

2. When we have too many features, observations become harder to cluster, too many dimensions causes every observation in your dataset to appear equidistant from all the others. And because clustering uses a distance measure such as Euclidean distance to quantify the similarity between observations, this is a big problem. If the distances are all approximately equal, then all the observations appear equally alike (as well as equally different), and no meaningful clusters can be formed.

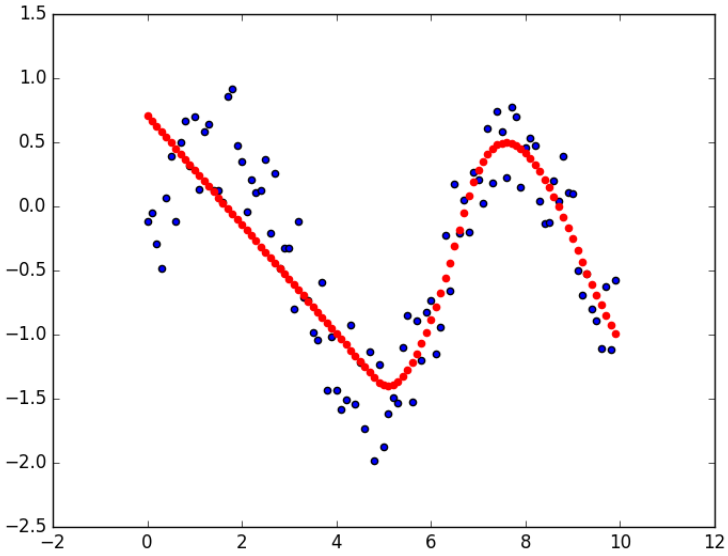
I recommend seeing this article, if you want to know more about this problem https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e

3 Classification and Regression

When the range of possible values for Y is **discrete**, we are dealing with a **classification** problem. An example: classification of a population to males and females.



When Y has a **continuous** domain, we are dealing with a **regression** problem. An example: build a polynomial model to predict sales or prices.



4 Which supervised algorithm is better?

A straight answer is that there is no such an algorithm for all situations. A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems. According to the No Free Lunch Theorem that states that all optimization algorithms perform equally well when their performance is averaged across all possible problems. Because of the close relationship between optimization, search, and machine learning, it also implies that there is no single best machine learning algorithm for predictive modeling problems such as classification and regression. One model may work very well on one dataset, but may perform badly on another. One of the challenges is to find the "best" model for some given problem.

5 Dealing with supervised problems

Let's recall that the main purpose of the supervised problems is learning a mapping function from X to Y to be able to make prediction on unseen/new data, so the main purpose is prediction not inference. Thus, the aim is a good performance on the unseen data not on the known data, for example suppose I want to make a prediction of the future stock of a market using last months data, even if I find a good function that maps well my features to my stock's quantity, it doesn't mean that using this function I will be able to do some good

predictions. The model has a performance on the known data, called training data, and the unseen data called test data. The latter is the important one. In this section, I will introduce briefly some of the issues that we will be dealing with when we are solving a supervised learning problem, these issues will be seen in details in the next lectures.

5.1 Bias-Variance trade-off

The bias-variance trade-off is a central problem in supervised learning. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data. Unfortunately, it is typically impossible to do both simultaneously.

So the goal is to find a target function that maps some inputs X to outputs Y. And then make some prediction on new inputs X^*. The prediction error for any machine learning algorithm can be broken down into three parts:

- Bias Error: The difference between the average of the predicted values and the true value. It is basically the deviation between the expected output of our model and the real value, so it indicates the fit of our model. A model with high bias pays very little attention to the training data, it is not capturing the underlying behaviour of the true functional form well and oversimplifies the model (imagine trying to fit a linear model to a sinus distribution). When we have a high bias, we are said to be underfitting. With a low bias, we are said to be overfitting.

- Variance Error: The variance of the predicted values. It determines how much the average model prediction deviates as different training data is tried. A model with high variance is suggestive that it is overfit to the training data, pays a lot of attention to training data and does not generalize.

- Irreducible Error: it cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable. In other words, it arises when X doesn't completely determine Y. That is, there are variables outside of X — and independent of X— that still have some small effect on Y. The only way to improve prediction error related to irreducible error is to identify these outside influences and incorporate them as predictors. Very important to keep in mind, specially if you are given a dataset to predict some output, and the information given is not enough to describe the output and there others unknown/not-given inputs that also influence the output.

5.2 Function complexity and amount of training data

The second issue is the amount of training data available relative to the complexity of the "true" function (classifier or regression function). If the true function is simple, then an "inflexible" learning algorithm with high bias and low variance will be able to learn it from a small amount of data. But if the true function is highly complex (e.g., because it involves complex interactions among many different input features and behaves differently in different parts of the input space), then the function will only be able to learn from a very large amount of training data and using a "flexible" learning algorithm with low bias and high variance.There is a clear demarcation between the input and the desired output.

5.3 Dimensionality of the input space

A third issue is the dimensionality of the input space, that leads us back to the curse of dimensionality seen before in this lecture. If the input feature vectors have very high dimension, the learning problem can be difficult even if the true function only depends on a small number of those features. This is because the many "extra" dimensions can confuse the learning algorithm and cause it to have high variance. Hence, high input dimensional typically requires tuning the classifier to have low variance and high bias. In practice, if the engineer can manually remove irrelevant features from the input data, this is likely to improve the accuracy of the learned function. In addition, there are many algorithms for feature selection that seek to identify the relevant features and discard the irrelevant ones. This is an instance of the more general strategy of dimensionality reduction, which seeks to map the input data into a lower-dimensional space prior to running the supervised learning algorithm.

5.4 Noise in the output values

A fourth issue is the degree of noise in the desired output values (the supervisory target variables). If the desired output values are often incorrect (because of human error or sensor errors), then the learning algorithm should not attempt to find a function that exactly matches the training examples. Attempting to fit the data too carefully leads to overfitting. You can overfit even when there are no measurement errors (stochastic noise) if the function you are trying to learn is too complex for your learning model. In such a situation, the part of the target function that cannot be modeled "corrupts" your training data - this phenomenon has been called deterministic noise. When either type of noise is present, it is better to go with a higher bias, lower variance estimator.

In practice, there are several approaches to alleviate noise in the output values such as early stopping to prevent overfitting as well as detecting and removing the noisy training examples prior to training the supervised learning algorithm. There are several algorithms that identify noisy training examples

and removing the suspected noisy training examples prior to training has decreased generalization error with statistical significance.

5.5 Other factors

- Heterogeneity of the data. If the feature vectors include features of many different kinds (discrete, discrete ordered, counts, continuous values), some algorithms are easier to apply than others. Many algorithms require that the input features be numerical and scaled to similar ranges (e.g., to the [-1,1] interval). Methods that employ a distance function are particularly sensitive to this. Other algorithms (like decision trees) easily handle heterogeneous data.

- Redundancy in the data. If the input features contain redundant information (e.g., highly correlated features), some learning algorithms will perform poorly because of numerical instabilities. These problems can often be solved by imposing some form of regularization.

- Presence of interactions and non-linearities. If each of the features makes an independent contribution to the output, then algorithms based on linear functions and distance functions generally perform well. However, if there are complex interactions among features, then algorithms such as decision trees and neural networks work better, because they are specifically designed to discover these interactions. Linear methods can also be applied, but the engineer must manually specify the interactions when using them.

When considering a new application, the engineer can compare multiple learning algorithms and experimentally determine which one works best on the problem at hand (see cross validation). Tuning the performance of a learning algorithm can be very time-consuming. Given fixed resources, it is often better to spend more time collecting additional training data and more informative features than it is to spend extra time tuning the learning algorithms.

6 Types of supervised learning algorithms

6.1 Parametric and Non-parametric models

- **A parametric algorithm** simply maps the data to a known/predefined functional form. It has a **fixed number of parameters** independent of the number of training examples, no matter how data you throw at a parametric model, it won't change it mind about the number of parameters it needs. A parametric algorithm is computationally faster, but makes stronger assumptions about the data. The algorithm may work well if the assumptions turn out to be correct, but it may perform badly if the assumptions are wrong. Examples: Linear regression, perceptron, LDA... they will be seen in the next lectures.

Benefits: Simpler (easier to understand and interpret), faster and they need less data.

Limitations: Constrained (by choosing a predefined form), limited to simple problems and they are more likely to fit poorly the true function.

- **A non-parametric algorithm** might sound a bit confusing at first: non-parametric does not mean that they have NO parameters! In contrary, a non-parametric algorithm uses a flexible number of parameters, and the number often grows as it learn from data. It is computationally slower but makers fewer assumptions about the data. They are good to use when you have a large amount of data and no prior knowledge and when you don't to worry about choosing the right features. Some examples: KNN, Decision Trees, Neural Networks...

  Benefits: Flexibility (Capable of fitting a large number of functional forms), power (no strong assumptions about the data) and they could have a higher performance for prediction.

  Limitations:they require a lot more training data to estimate the mapping function, they are slower as they often have far more parameters to train and they have more risk to overfit the training data.

6.2 Generative and Discriminative models

- **Generative models** are models where the focus is the distribution of individual classes in a dataset and the learning algorithms tend to model the underlying patterns/distribution of the data points. These models use the intuition of joint probability in theory, creating instances where a given feature (x)/input and the desired output/label (y) exist at the same time. Think of them as providing a model of how the data is actually generated. They actually learn P(X,Y).

  Generative models use probability estimates and likelihood to model data points and distinguish between different class labels in a dataset. These models are capable of generating new data instances. However, they also have a major drawback. The presence of outliers affects these models to a significant extent.

  Examples: Naive Bayes classifier, Linear Discriminant Analysis...

- **Discriminative models** tend to learn the boundary between classes/labels in a dataset. Unlike generative models, the goal here is to find the decision boundary separating one class from another. So while a generative model will tend to model the joint probability of data points and is capable of creating new instances using probability estimates and maximum likelihood, discriminative models (just as in the literal meaning) separate classes by rather modeling the conditional probability and do not make any assumptions about the data point. They are also not capable of generating new data instances. They are more robust to outliers.

Classifiers computed without using a probability model are also referred to loosely as "discriminative".

Examples: Logistic regression, Support vector machine, Decision Trees...

A special lecture will be dedicated to these two types.

6.3 Others

Tree-based machine learning methods are among the most commonly used supervised learning methods. They are constructed by two entities; branches and nodes. Tree-based ML methods are built by recursively splitting a training sample, using different features from a dataset at each node that splits the data most effectively. The splitting is based on learning simple decision rules inferred from the training data. Methods like decision trees, random forest, gradient boosting are being popularly used in all kinds of data science problems. We have regression and classification trees.

Kernel Based methods: are using a linear classifier to solve a non-linear problem, this is done by transforming a linearly inseparable data to a linearly separable one.

We will get to these types in details in the next lectures. This is just introduction to have a large view about supervised learning.