Lukas Johannes Ruettgers, ML HW7, November 22, 2023

1 Introduction

Many aspects and attributes of our universe can be quantified in data. Our understanding of the world can hence improve through the understanding of data. However, our limited resources are not able to efficiently capture the abundance of our universe. For that reason, we can naturally collect only a tiny fraction of *samples* $\mathcal{X}$ of the entire data $\mathbf{X}$. In general, we are interested in the nature that governs the total *distribution* $\mathbf{X}$. To that end, the field of Machine Learning (ML) formalises methods that given some samples infer conclusions on the dataset from which the samples stem from.

These methods are often divided into subcategories by the type of conclusions they intend to infer. Firstly, a *regression* method could make conclusions on the relation between the values of several attributes – also referred to as *features* – of $\mathbf{X}$ e.g. in form of a deterministic function. Secondly, a *classification* method assumes that a dataset $\mathbf{X}$ is partitioned into multiple classes and intends to correctly discriminate each instance of $\mathbf{X}$ to their corresponding class. Lastly, one could also desire to recover the probability density function that describes the frequency of each feature value combination. This task is called *density estimation.*

1.1 Feature engineering

The imperfections of these methods arise from multiple error sources. Already in the data collection process we restrict ourselves to a certain measurement space that may neglect important attributes. In such a case, even the best methods can not obtain perfect conclusions since the features that are provided as input lack necessary attributes. To reduce this first error source, mostly a lot of attributes are captured in practice, even if some of them provide only few additional information. However, the size of the feature space and hence the complexity of inductive reasoning will grow exponentially with the amount of features. To ensure the feasibility of the ML algorithms, one might desire to keep the feature space as small as possible while retaining as much information as possible. When a large number of features is available, *feature selection* and *feature extraction* algorithms have been developed to find low-dimensional feature subspaces with minimal information loss. While feature selection methods only consider combinations of the given features, feature extraction methods also allow any linear transformations of the given features as feature subspace. Measuring the loss of information requires a metric to express the relevance of information to the specific task. For classification tasks, *class separability metrics* rate features by the additional information they provide about the membership to a specific class. Popular metrics seek to maximize the feature value distance between samples of different classes while keeping the distance for samples of the same class minimal. Some other metrics apply the Shannon entropy to the distribution of a feature value across classes. For example, a feature whose value distribution is identical for all classes would have a high entropy and does not provide any additional information. Conversely, if only one class would take on certain values of a feature, the entropy of that feature is small. Both feature selection and extraction are subsumed in *feature engineering*, which tackles the problem of obtaining optimal features in its entirety. The information loss in the feature extraction process constitutes the second error source.

1.2 Mathematical model and objective function

After the input feature space has been obtained, the last error source arises from the ML algorithms themselves. Even if we know the most relevant features, we still do not know their distribution. Because of the abundance of possible mathematical models, many algorithms make assumptions that simplify the problem but inevitably entail performance limits. The mathematical model directly defines the *hypothesis space* of possible solutions. To search an optimal solution inside that space, we need to define optimality first. That is, we require an *objective function* $J(\alpha)$ that scores each hypothesis α based on its quality to hold for the general dataset $\mathbf{X}$. However, as engineers of the algorithms we do normally not know $\mathbf{X}$ either and must regard the available samples as sufficiently representative for $\mathbf{X}$. For that reason, objective functions usually base their estimation of the adequacy of α on the available *training dataset* $\mathcal{X}$.

1.3 Learning Theory

On the contrary, there is also theoretical work on the ability of objective functions to yield solutions that generalize to $\mathbf{X}$. In the optimal case, one should not infer a conclusion that best fits the training dataset but a conclusion that will most likely hold for the entire data. In the terms of *Statistical Learning Theory* (SLT), the objective function J should exhibit a *minimal expected risk.* That is, the hypotheses α that optimize J should keep as much of their optimality for $\mathbf{X}$ as possible. To obtain general theoretical results, we must make the conservative assumption that we have no knowledge on the true model beyond the given samples. Therefore, the best that we can do is to avail to the law of large numbers and estimate the expected risk by the *empirical risk* R_{emp}. In order to converge to the smallest possible risk, R_{emp} must converges in probability to the true risk R for *any* α. This worst-case condition is necessary, as the central theorem of SLT states. On the other side, *Computational Learning Theory* (CLT) assesses the power of mathematical models to express patterns and hypotheses on $\mathbf{X}$. This way, it investigates the necessary complexity to optimally express the nature of $\mathbf{X}$. One of the most famous complexity measures is the *VC dimension*, which equates the maximum number of vectors that the given mathematical model can arbitrarily discriminate.

2 Regression estimation

The easiest and most well-studied instance of regression is the linear regression. As the name suggests, linear regression assumes that a fraction of the measured features Y that constitute $\mathbf{X}$ are linear dependent from another fraction X. We desire to obtain a function f that describes this linear dependence as $y = f(x) = w^T x$, where w are the *weights* of each feature in X. The optimal f is defined as the linear function that minimizes the squares distance between $f(x_i)$ and y_i for each $(x_i, y_i) \in \mathcal{X}$. To assess the model's accuracy despite noise in the data, a random variable ζ is mostly added to each term to subtract the aleatoric uncertainty from the epistemic uncertainty we are interested in.

If the feature Y takes on not continuous but only binary values, *logistic regression* was developed to obtain the *sigmoid function* $P(y = 1 \mid x) = \frac{e^{w^T x}}{1+e^{w^T x}}$ that most likely represents the probability of $y \in Y$ to take on value 1. In this case, the objective function is the negative logarithmic likelihood to estimate y accurately, where the logarithmic function retains critical points while ensuring numerically stable calculations.

3 Classification and Pattern Recognition

The logistic regression task above bridges the gap between regression and classification, since this method can be used for binary classification and also be extended to discriminate into multiple classes. In classification, we desire to learn a *classifier* $f : \mathbf{X} \to \{1, \dots c\}$ that assigns each instance of $\mathbf{X}$ to its corresponding class inside the c classes. Before we summarize the most important methods, we glimpse at metrics that assess classification performance.

3.1 Classification Performance Assessment

In the general case, we have $c \in \mathbb{N}$ classes and therefore c^2 of possible scenarios. There are c classes to which the classifier can assign x and there are c possible true memberships. The *confusion matrix* $A \in \mathbb{N}^{c \times c}$ accumulates the absolute frequency of each possible scenario in the training dataset $\mathcal{X}$, where A_{ij} equates the number of samples in $\mathcal{X}$ that truly belong to class j and have been assigned to class i by f. In the binary classification case, this simplifies to four possible scenarios, which are referred to as a False (F)/True (T) Positive (P)/Negative (N) respectively. While the performance of f can be simply assessed by the *accuracy* $\frac{TP+TN}{TP+FP+FN+TN}$, one could also give individual weights to different classification errors and find a minimal risk decision. To find such a threshold, the ROC curve plots the False Positive Rate $FPR = \frac{FP}{TN+FP}$ against the TPR and is commonly used to find a desirable trade-off.

3.2 Methods

Let us first consider the case where the samples in $\mathcal{X}$ are not labeled and hence any information about the classes beyond their mere existence is not known. In such a case, *Fisher's Linear Discriminant* is an early method to find a binary classification boundary. Here, the objective is to maximize the distance between two class means while minimizing the inner-class distance between each sample and the class mean.

On the other side, the task of *Supervised Learning* assumes that the class label of each sample in $\mathcal{X}$ is given. Then there is more information on the classes' shape and the goal is merely to find a discriminant that separates the samples of different classes. An early method for that task is the *Perceptron*, which seeks to perfectly separate the classes by a linear hyperplane. The *Perceptron Criterion* only considers whether each sample is located on the correct side of the hyperplane. If the classes are linearly separable, multiple solutions might satisfy this criterion. The *Optimal Hyperplane* in that solution region was found to be the hyperplane that maximizes the margin between the nearest sample vectors – called support vectors – on both sides. The mathematical formulation of this constrained optimization problem as a *Primal Problem* paved the way to *Support Vector Machines*, which find exactly that hyperplane. However, if the classes are not linearly separable, one might either relax the Perceptron Criterion to correctly assign as many samples as possible or extend the expressive power of the classifier. *Multilayer Perceptrons* seize the latter approach by extending the network structure of Perceptrons by non-linear *activation functions* in each layer. In contrast to the sign function used by Perceptrons, these activation functions are differentiable and therefore allow error backpropagation which was the missing component for a *gradient descent* optimization procedure to be applied to the Perceptron network.

Distance-based classifiers have also been extended to the case of non-linear separable classes. Instead of deciding membership in a class only on the distance to the mean of the class, the *k-Nearest-Neighbours* method assigns each $x \in \mathbf{X}$ to the class to which the most of the k closest neighbours belong. To tackle the computational cost, only essential samples that constitute the convex hull of the respective classes need to be stored with little loss in accuracy. Moreover, the data can be organized in tree structures to allow more efficient nearest-neighbour search.

Besides considering the entire vector space at once for classification, the idea of *Decision Trees* is to iteratively discriminate data by individual features until the resulting groups are so small that all $x \in \mathbf{X}$ that end up in that group belong to the same class. As the name suggests, this iterative discrimination is visualized as a tree, where the inner nodes comprise a feature by which incoming data is further split up to the subnodes. To construct effective trees, we avail to feature selection metrics like the feature entropy presented above to choose by which feature we shall discriminate at each node. Since the tree structure is not limited, the decision trees would overfit the training data $\mathcal{X}$ and refrain from making reasonable compromises that seem imperfect on $\mathcal{X}$ but generalize well to $\mathbf{X}$. To tackle this issue, *Random Forests* comprise multiple restricted decision trees and finally decide based upon the collection of decisions by each tree. The trees are usually restricted in the samples they are trained with, in the features they may choose at each node or the granularity of their leafs. The method hopes that the weak complexity of each tree combined with the randomized training samples and feature sets creates independent decision trees which may complement each other's weaknesses to form a stronger classifier. More generally, this idea of combining multiple weak classifiers to a strong one is subsumed in *Ensemble learning.* To ensure that the classifiers complement each other's classification weaknesses, the *Ada Boost* algorithm iteratively constructs a classifier and adjusts the weight of each sample based on the prediction accuracy of the past classifiers. Finally, the final classifier is a weighted combination of the weak classifiers where each weight corresponds to the accuracy of the respective weak classifier.

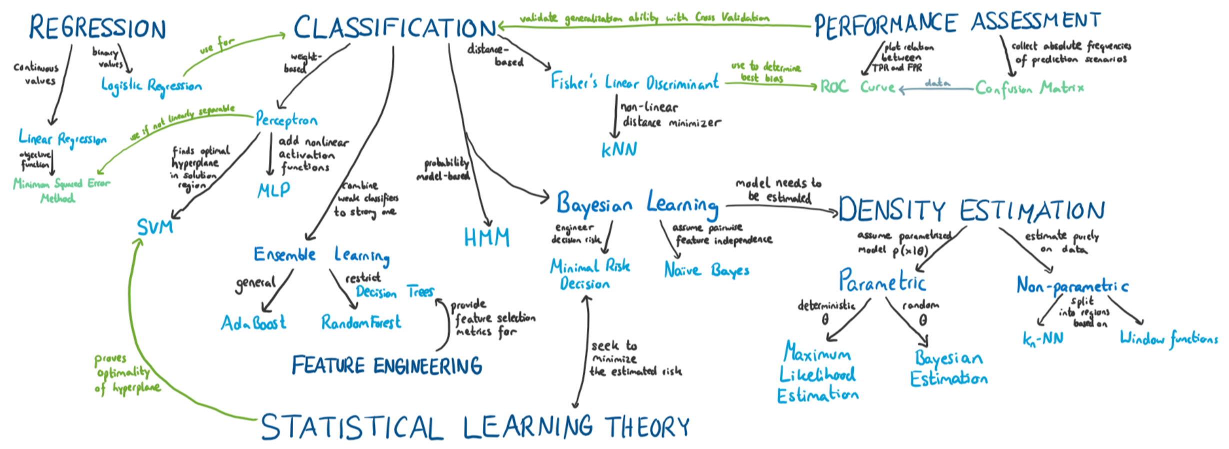
So far, the methods did not seriously consider $\mathbf{X}$ to constitute *random* vectors that emerge from a probabilistic distribution $p(x)$. For a fixed x, Bayes' rule yields that the *posterior probability* $p(\omega_i \mid x)$ that x belongs to class i is proportional to the *conditional density* $p(x \mid \omega_i)$ and the *prior probability* $p(\omega_i)$ of each class. If we make assumptions on the prevalence of each class and how the data of each class is distributed, we have fixed values for $p(\omega_i)$ and we can model $p(x \mid \omega_i)$ by a parametrized probability density function (p.d.f.) model. However, this will still leave us with a hard problem, since parametrized density functions can still take complex forms for a high-dimensional feature space. To ease this computational burden, the *Naïve Bayes Classifier* (NBC) pretends that each feature is independently distributed of each other. This allows the factorization of a complicated p.d.f. into the product of more simple densities. However, this assumption hardly holds in practice and the method will therefore only yield mediocre approximations of the true posterior probability.

4 Density estimation

We see above that the calculation of the posterior breaks down to a reasonable estimation of the p.d.f. of x. While parametric estimation assumes that the p.d.f. belongs to a parametrized family of densities, non-parametric estimations only use the given data to estimate the model. In the former case, one could either regard the parameters as deterministic and obtain the parameters which maximize the likelihood of the p.d.f. to generate the given data $\mathcal{X}$. This method is called *Maximum Likelihood Estimation*. When the parameters are regarded as random variables, we could avail to *Bayesian estimation* to obtain an estimation that minimizes the empirical risk as we defined above when we covered SLT. On the other hand, non-parametric estimation approximates the true density by the empirical density of the data in fixed regions, which are usually determined to encompass a fixed number of samples.

If one desires to study the temporal evolution of a certain variable, *Hidden State Markov Models* (HMMs) provide a simplified but still powerful density model where each random variable is a sequence $\{x_i\}_{i=1,...}$. Most importantly, *Markov chains* make the assumption that each state x_i depends only on its predecessor, which notably simplifies the computation of conditional probability $p(x_i \mid x_1, \ldots x_{i-1})$, which in general increases in complexity over time. In comparison to normal Markov chains, HMMs regard each x_i only as a measurement of a hidden state π_i, over which the transition probability is defined. While the *evaluation problem* requires to obtain the probability of a measurement sequence given all probabilities and hence requires an expectation over all possible hidden state sequences, the *decoding problem* requests only the most likely hidden state sequence to generate x. Similarly to parametric estimation, the *learning problem* seeks to estimate reasonable probability densities of the HMM given its state structure.

Below is a small graph to summarize the most important relations of the covered concepts.



REGRESSION
continuous values
binary values
Logistic Regression
use for
CLASSIFICATION
validate generalization ability with Cross Validation
PERFORMANCE ASSESSMENT
plot relation between TPR and FPR
collect absolute frequencies of prediction scenarios
weight-based
distance-based
Fisher's Linear Discriminant
use to determine best bias
ROC Curve
data
Confusion Matrix
non-linear distance minimizer
kNN
Linear Regression
objective function
Minimum Squared Error Method
use if not linearly separable
Perceptron
finds optimal hyperplane in solution region
add nonlinear activation functions
MLP
SVM
combine weak classifiers to strong one
probability model-based
HMM
Bayesian Learning
model needs to be estimated
DENSITY ESTIMATION
engineer decision risk
assume pairwise feature independence
Minimal Risk Decision
Naïve Bayes
assume parametrized model $p(x|\theta)$
estimate purely on data
Parametric
Non-parametric
Ensemble Learning
general
restrict
Decision Trees
AdaBoost
RandomForest
provide feature selection metrics for
FEATURE ENGINEERING
seek to minimize the estimated risk
deterministic θ
random θ
split into regions based on
k_n-NN
Window functions
Maximum Likelihood Estimation
Bayesian Estimation
proves optimality of hyperplane
STATISTICAL LEARNING THEORY