PROJECT ML

University of Hamburg

Department of Mathematics

Jonas Eckhoff - Timo Greve

Max Lewerenz - Giulia Satiko Maesaka - John-Robert Wrage

Machine Learning Methods
Group 5

Contents

1 Introduction

These notes are the documentation of the research on machine learning methods lead during the development of an engine for facial emotion recognition for the course on Machine Learning at the University of Hamburg. This document aims to give an overview on the current context of Machine Learning and then it focuses on presenting the algorithms implemented in the engine (KNN, SVM, AdaBoost and CNN); it explains how they work, their advantages and disadvantages.

2 What is ML?

Machine learning is the field of study responsible for developing and understanding algorithms that allow computer systems to perform certain tasks independently. The type of tasks considered are these for which any rule-based approach is unfeasible, either because a precise set of rules is unclear, for example in pattern recognition, or because computing according to the rules is too complex, for example in playing Go.

Given a data set, the goal of a machine learning method can be of two types, one is to use the data to find a function, which will then match new input data to a value; another is to find a subdivision of the data set based on similarities. The methods used for the first type of goal are named *supervised learning* methods and the given data set, called *training set*, must be structured in pairs input-output. For the second type of goal, one uses *unsupervised learning* methods.

In a supervised learning method, the output value is either quantitative or qualitative. For the problem of predicting the price of a house given its area and location, the output is a quantitative value; these are called *regression* problems. On the other hand, if the task is to address an emotion to a given face expression, the output is a qualitative value encoding a certain emotion; these are called *classification* problems.

Supervised learning methods are structured in two parts. Firstly it consists of a predetermined set of functions, called *hypothesis space*, from which a final function will be chosen. Secondly the method needs a predetermined way of searching through the hypothesis space, therefore it requires a predetermined *loss function* and *optimization algorithm*. The choice of hypothesis space, loss function and optimization algorithm depends on the task and different methods need to be implemented and evaluated. For the task of recognizing facial emotion, it is known that between the methods so far developed, the best results are achieved by convolutional neural networks. Still, this project tests different methods one more time, for the sake of experience.

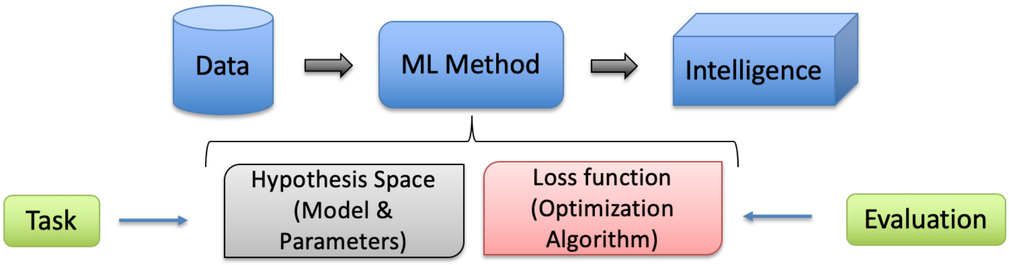

Data
ML Method
Intelligence
Task
Hypothesis Space (Model & Parameters)
Loss function (Optimization Algorithm)
Evaluation

Figure 1: The flow of developing an engine based on machine learning.

2.1 A first approach to Machine Learning methods

Machine Learning is not a new paradigm for developing algorithms, however the field's growing pace increased drastically in the 1990s. This was when the computational power and volume of available data finally allowed practical research. Due to this fast and recent development, multiple methods emerged without an established theory and organization as a base. For that reason, giving a brief overview on the most common algorithms can be a debatable and overwhelming task.

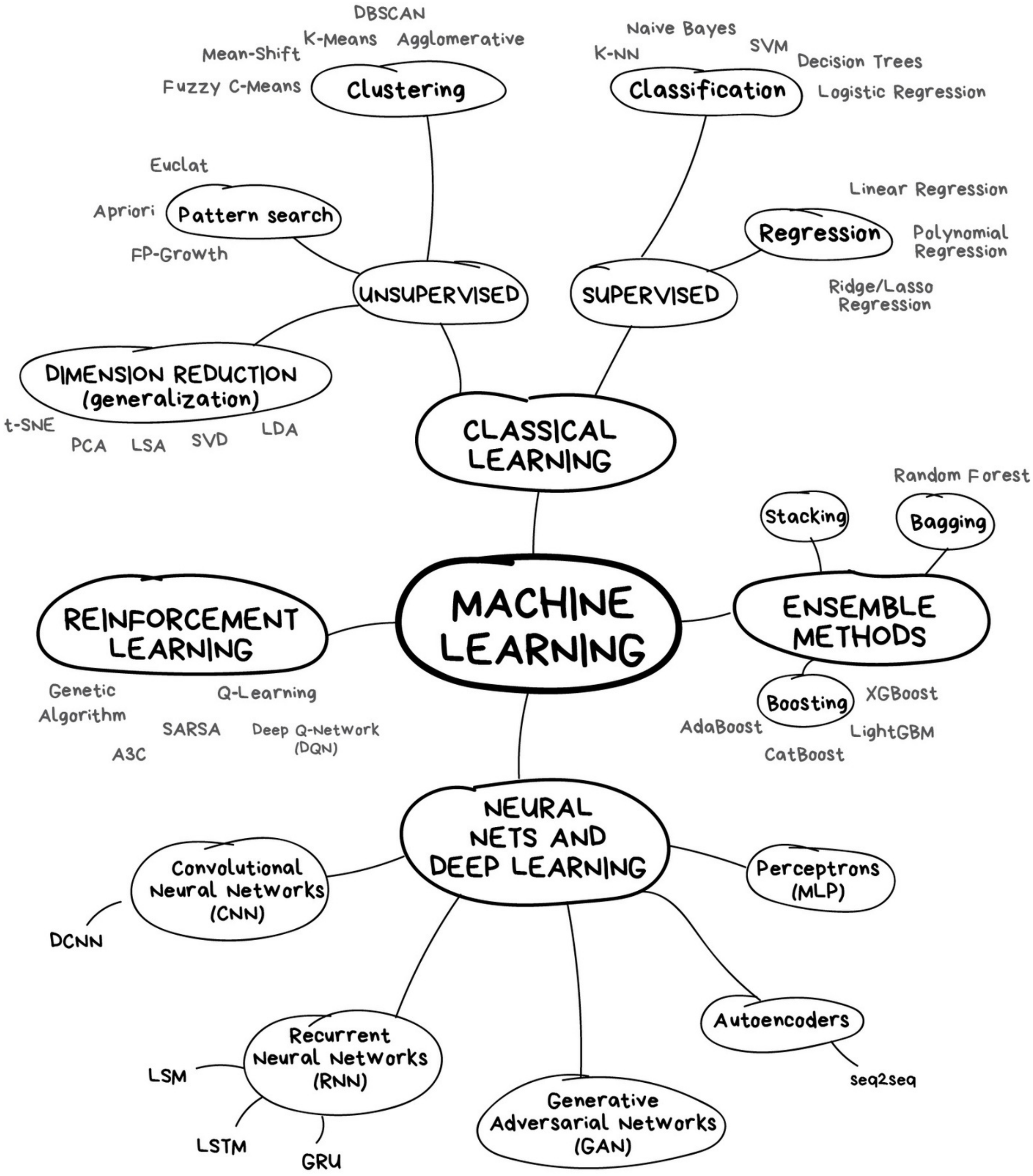


Figure 2: ML-Overview

Nevertheless, giving the idea of some machine learning methods is still an interesting way to stablish a more concrete understanding of the field. In this section we give a brief overview on Linear Regression, as an introductory example.

In the following chapters we discuss K-Nearest Neighbours, Support Vector Machines, Decision Trees, AdaBoost and Neural Networks.

2.1.1 Linear Regression

In a linear regression algorithm, the hypothesis space $\mathcal{H}$ are functions $h : \mathcal{X} \to \mathcal{Y}$ of shape

$$h(x_1, \dots, x_d) = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d,$$

where the values β_i are parameters to be chosen.

Given a data set $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$ of input-output pairs, a common choice of loss-function in regression problems is the square error given by

$$L(h(x), y) = (h(x) - y)^2.$$

This is a way of measuring the mismatch between the predicted output $h(x^i)$ and the correct output y^i.

For the choice of a final function, it's best to consider the whole set of training data, therefore define the empirical risk

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x^i), y^i),$$

which averages the total mismatch.

Finally, an optimization algorithm is used to solve the problem

$$\underset{h \in \mathcal{H}}{\text{minimize}} \hat{R}(h).$$

In this step, it is important that the empirical risk is a differentiable (or at least continuous) function.

2.1.2 Methods in General

This same approach can be used for a hypothesis space $\mathcal{H}$ consisting of another type of (non-linear) functions, for example polynomials. If the functions in $\mathcal{H}$ are also determined by few parameters, a minimizer might be found in closed form.

Some other times, the functions in $\mathcal{H}$ may be a composition of many simpler functions, for example in decision trees and neural networks. In these situations, the number of parameters defining these functions might be huge, but here the focus is in the simplicity of computing these functions. Finding a minimizer in these cases require methods from numerical optimization for example, the algorithm of gradient-descent and for the composition of functions, methods such as backpropagation.

There are also various choices for the loss-function and it depends on the type of task to be solved. A desirable loss-function could be the indicator

$$L(h(x), y) = \mathbb{1}_{\{h(x) \neq y\}} = \begin{cases} 1, & \text{if } h(x) \neq y \\ 0, & \text{if } h(x) = y \end{cases}.$$

Nevertheless, this function is not continuous. Therefore one might look for ways of approximating it. In the case of a binary classifier, i.e., $\mathcal{Y} = \{0, 1\}$, a continuous approximation is the log-loss or cross-entropy function

$$L(h(x), y) = \begin{cases} -\log(h(x)), & \text{if } y = 1 \\ -\log(1 - h(x)), & \text{if } y = 0 \end{cases},$$

this gives large values when $h(x)$ and y mismatch.

2.2 Evaluating a method

If the machine learning method learns a function h which performs perfectly in the training set (the empirical error is 0), but does not generalize to a new input, then the engine built cannot be considered successful. The problem of a method only memorizing the training set is called *overfitting*.

A way of measuring the generalization capacity of a classifier is by modelling the training set as samples from a pairs of random variables (X_i, Y_i) that are independently identically distributed. Then the empirical risk becomes a random variable, whose expected value is called *generalization risk*.

The generalization capacity of an empirical risk minimizer h given by the machine learning method can be measured by the *estimation error*, which compares the generalization risk of h to the minimum generalization risk achieved by some function in $\mathcal{H}$. An absolute way of measuring the generalization capacity of $\mathcal{H}$ is given by the *approximation error*, which measures the smallest possible generalization risk achieved by a function in $\mathcal{H}$.

The estimation error and approximation error have a trade-off. Whenever the set $\mathcal{H}$ is too large (compared to the training set, for example), it is likely that a function with very small generalization risk can be found and therefore the approximation error is small. But at the same time, it is more likely that the function h chosen by the method overfits, and therefore has big generalization risk. In this case the estimation error is likely to be large. Whenever the set $\mathcal{H}$ is too small, the approximation error might be big. But the function h is likely to be close to the function in $\mathcal{H}$ minimizing the approximation error and, therefore, the estimation error is small.

The study of the relationship between the generalization risk, the empirical risk, the capacity of a hypothesis space and the number of samples, is within the field of Statistical Learning Theory.

For our practical purpose of good training of facial emotion classifiers, we prevented overfitting by working on the data set, for example mirroring the images, by dropping out badly labelled pairs and by stopping the training whenever stagnation in the validation accuracy was observed. Our data set, made available from a Kaggle competition, consisted of 28.709 training pairs and 7.178 testing pairs. The input were 48×48 grayscale images of faces and the output were within 7 emotions: angry, disgusted, frightened, sad, surprised, happy and neutral. We also trained the methods on fewer emotions to achieve better accuracy.

3 Classical learning

In this Chapter we are going to explain the Methods k-nearest-neighbours and support vector machine. Also we are going to briefly introduce decision trees because we are using them in Chapter 5 for AdaBoost.

3.1 KNN

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

The straight-line distance (Euclidean distance) is a popular and familiar choice to measure the distance between two samples. Note that there are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. Other possibilities to measure the distance could be e.g. Manhattan or Hamming. In the following picture the black dot could be classified as - if we only compare with the next neighbour. But if one chooses to compare with the three nearest neighbours it the dot would be classified as +, because two dots would be + and only one would be -. It is important to select a "good" amount of neighbours to classify. An example for this can be found in the next picture where the black dot is near to an (probably) outlier. If k would be chosen as 1 then the black dot would be classified as + but it would probably be -.

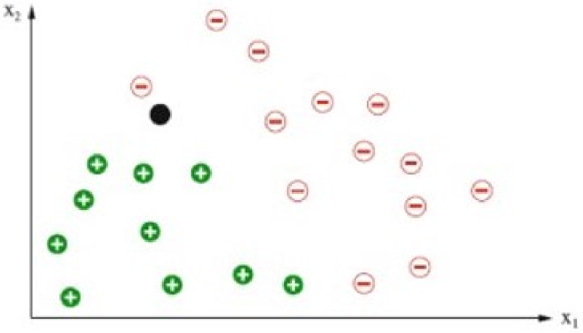


Figure 3: K-Nearest Neighbors

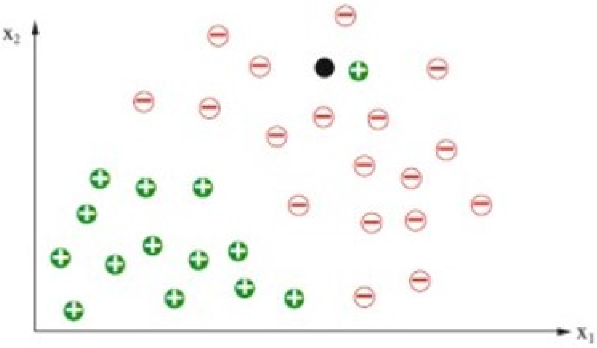


Figure 4: K-Nearest Neighbors

Sources:

- scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html https://scikit-learn.org/stable/modules/neighbors.html

- Wikipedia: Nächste-Nachbarn-Klassifikation https://de.wikipedia.org/wiki/Nächste-Nachbarn-Klassifikation

3.2 Decision Trees

(Classification-) Decision tree

The data will be divided by yes/no questions. This is done by selecting the question such that the best information gain is created(spliting the data via a feature which can seperate the different lables best). There are different methods to measure the entrophy from which the information gain can be calculated. A common approach is to calculate the Gini-index. For more information please have a look at the scources. In the following picture one can see a decision tree with depth 8. Trees tend to overfit if one let them grow to their maximum depth, so there are methods to prevent this from happening(e.g. pruning).

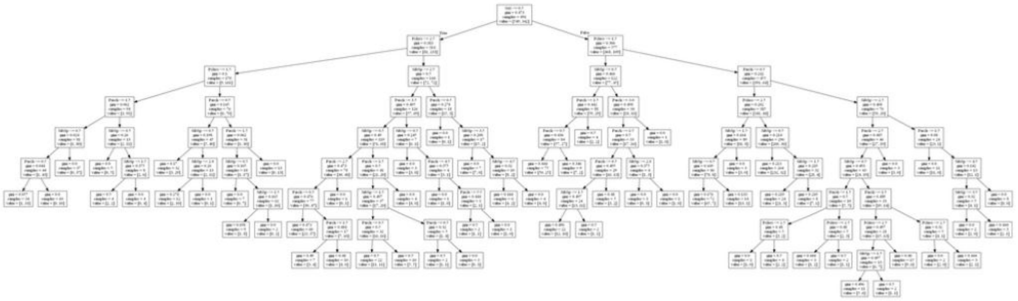


Figure 5: Decision Tree

Decision Trees can used for ensemble methods (Random forest, adaboost,...). We refere to Chapter 5.

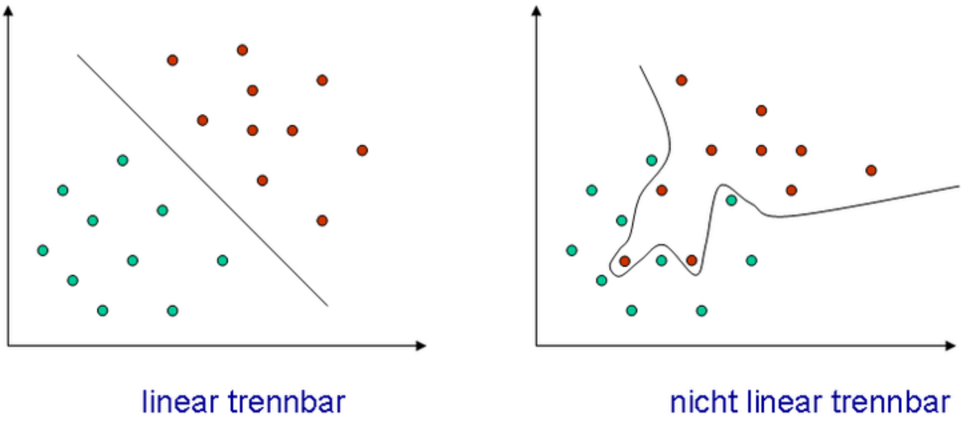
Sources:

- scikit-learn: https://scikit-learn.org/stable/modules/tree.html https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- Wikipedia: Entscheidungsbaum https://de.wikipedia.org/wiki/Entscheidungsbaum

3.3 Support Vector Machine

3.3.1 Introduction

Support Vector Machines (SVM) are supervised learning algorithms for classification. They are one of the most widely used supervised learning algorithms. SVMs offer a high accuracy when it comes to classification. The idea behind a SVM is to find a hyperplane that seperates classes of datapoints with a large margin. Where the margin is the smallest distance between the closest datapoint x of one class and the hyperplane (therefore the name 'support vector'). Because of this feature the SVM is sometimes also called **large margin classifier**. The so called kernel trick can be applied when non linear data has to be classified.



3.3.2 Linear Support Vector Machines

For a short introduction, assume the Support Vector Machine algorithm is being used for a binary classification problem $\mathcal{Y} = \{-1, 1\}$, that the training set has $\mathcal{X} \subset \mathbb{R}^d$ and the subsets $X_{-1} = \{x^i \in \mathcal{X} \colon y^i = -1\}$, $X_1 = \{x^i \in \mathcal{X} \colon y^i = 1\}$ can be the separated by an affine hyperplane. Under these assumptions, the hypothesis space is the set of affine hyperplanes in $\mathbb{R}^d$ that separate the training data and not all possible affine hyperplanes, as in Linear regression.

While in Linear Regression the empirical risk adds an error for each element in the data set, in SVMs the error of a given hyperplane is the sum $\delta_- + \delta_+$, where δ_- is the minimum distance between the hyperplane and an input point which would be classified as -1 and δ_+ is defined analogously, for an input point which would be classified as 1.

3.3.3 Mathematics behind SVMs

For given input data $X = \{x_1, ..., x_m\}$, with $x_i \in \mathbb{R}^n$ and matching result vector $Y = \{y_1, ..., y_m\}$, with $y_i \in \{-1, 1\}$. We want a classifier

$h : \mathbb{R}^n \to \{-1, 1\}$ such that $h(x) = \begin{cases} 1 & \text{,if } w^T x + b > 0, \\ -1 & \text{,if } w^T x + b < 0. \end{cases}$ with $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$. We want h to be correct for most samples. This formulation leads to the following primal problem:

$min_{w,b,s} \frac{1}{2} w w^T + C \sum_{i=1}^{m} s_i,$

s.t.: $y_i(w^T x_i + b) \geq 1 - s_i,\ s_i \geq 0.$

Where s_i are the so called slack variables that should denote the distance from the correct margin if a point is misclassified.

Intuitively we want to maximize the margin what results in minimizing $||w||^2$ including a penalty when something is misclassified. C controlls the strength of the misclassification penaly. One could view it as a inverse regularization parameter. A large C results in overfitting and a smaller C results in a "smoother" fit.

The dual problem to the above Primal problem is:

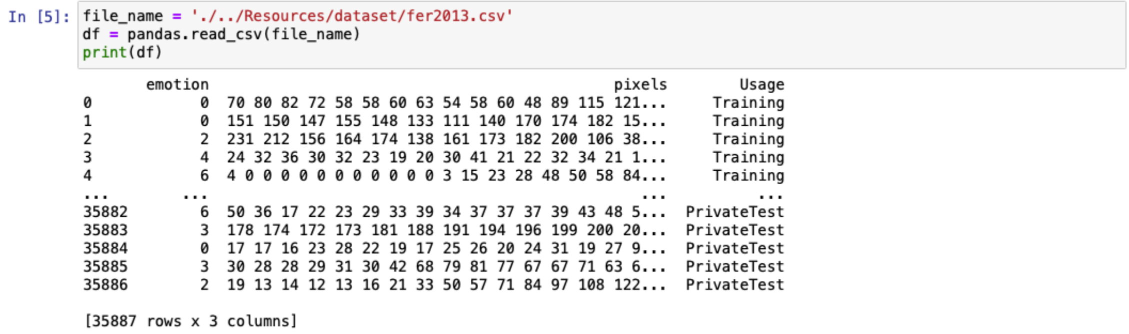
$min_\lambda \frac{1}{2} \lambda^T Q \lambda - e^T \lambda$

s.t.: $y^T \lambda = 0$ and $0 \leq \lambda_i \leq C$ for all $i = 1, ..., m$. The entries of the matrix $Q \in \mathbb{R}^{m \times m}$ are given by $Q_{i,j} = y_i y_j \langle x_i, x_j \rangle$.

This shows that the minimization only depends on the scalar product of x_i and x_j and we can apply the kernel trick. Instead of the standard scalar product we can use a kernel function $K(x_i, x_j)$.

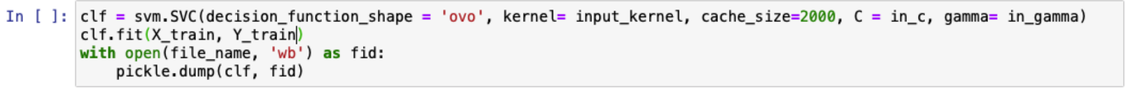
3.3.4 In our case

We use the python library sklearn, which can easily be installed via pip. First we get the training data and split it into training and test data.



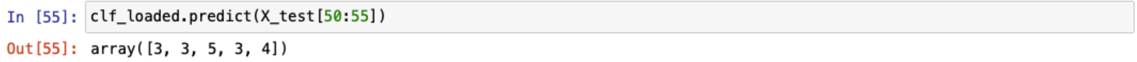
```
In [5]: file_name = './../Resources/dataset/fer2013.csv'
        df = pandas.read_csv(file_name)
        print(df)

               emotion                                             pixels        Usage
        0            0  70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...     Training
        1            0  151 150 147 155 148 133 111 140 170 174 182 15...     Training
        2            2  231 212 156 164 174 138 161 173 182 200 106 38...     Training
        3            4  24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...     Training
        4            6  4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...     Training
        ...        ...                                                ...          ...
        35882        6  50 36 17 22 23 29 33 39 34 37 37 37 39 43 48 5...  PrivateTest
        35883        3  178 174 172 173 181 188 191 194 196 199 200 20...  PrivateTest
        35884        0  17 17 16 23 28 22 19 17 25 26 20 24 31 19 27 9...  PrivateTest
        35885        3  30 28 28 29 31 30 42 68 79 81 77 67 67 71 63 6...  PrivateTest
        35886        2  19 13 14 12 13 16 21 33 50 57 71 84 97 108 122...  PrivateTest

        [35887 rows x 3 columns]
```

Next we fit the model to the training data for the chosen parameters and safe it with the pickle model.



```
In [ ]: clf = svm.SVC(decision_function_shape = 'ovo', kernel= input_kernel, cache_size=2000, C = in_c, gamma= in_gamma)
        clf.fit(X_train, Y_train)
        with open(file_name, 'wb') as fid:
            pickle.dump(clf, fid)
```

Now the model can give a prediction on new images.



```
In [55]: clf_loaded.predict(X_test[50:55])
Out[55]: array([3, 3, 5, 3, 4])
```

Sources:

- Martin Lotz: *Mathematics of Machine Learning*. Lecture Notes, Warwick (UK), 2020.
- scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
  https://scikit-learn.org/stable/modules/svm.html
- Coursera: Machine Learning, by Stanford University
  https://www.coursera.org/learn/machine-learning/home/welcome
- Wikipedia: Support vector machine.https://en.wikipedia.org/wiki/Support_vector_machine

4 Neural Networks and Deep Learning

4.1 Convolutional Neural Networks CNN

A convolutional neural network is a deep learning algorithm which is primarily used to classify images. It can more easily capture spatial (and temporal) dependencies since the convolution filters use a linear combination of neighboring pixels to calculate an output value. After a convolution filter is applied, a max-pooling filter can be applied, which lowers the amount of nodes in the network. A 2x2 max-pooling filter, for example, replaces every 2x2 square with the maximal value in that square, resulting in lowering the amount of nodes after that layer to a quarter. At the end the multidimensional layers are flattened to a one dimensional vector that is connected to the output via a fully connected layer.

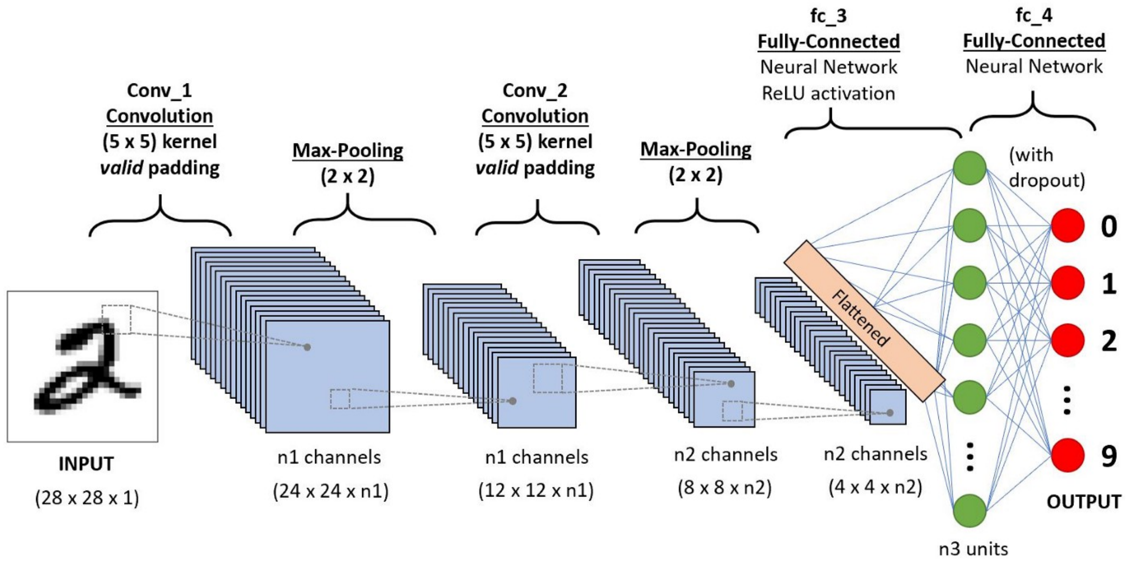


Figure 6: Example of a CNN architecture

Helpful links:
A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-net

Since we want to detect emotions in images, a convolutional neural network would be a good choice. It is the go-to type of network for these kind of computer vision problems.

4.2 Recurrent Neural Networks RNN

A recurrent neural network (RNN) is a type of artificial neural network commonly used in speech recognition and natural language processing. It uses feedback loops, that allow information to persist (the network has memory), to process sequences of data.

RNNs are not useful to analyse facial expression in an image.

4.3 Generative adversarial networks GAN

Generative adversarial networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other in order to generate new, synthetic instances of data that can pass for real data. They are used widely in image generation, video generation and voice generation.
One neural network, called the generator, generates new data instances, while the other, the discriminator, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data that it reviews belongs to the actual training dataset or not.

Helpful links:
A Beginner's Guide to Generative Adversarial Networks (GANs)
`https://pathmind.com/wiki/generative-adversarial-network-gan`

GANs can be used to generate images and videos. We could use this type of network to expand training data. Having a well trained GAN would allow us to create new images for specific emotions that look convincingly real in order to prevent overfitting by creating an almost endless stream of new images. It may also be used to create video training data that can be used to train a neural network with memory to better capture changes in emotion in a video stream. The training of a GAN however takes a very long time compared to a simple CNN which makes it probably unfeasible for us even though it would create interesting possibilities to explore.

4.4 Autoencoders

Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact "summary" or "compression" of the input. An autoencoder consists of 3 components: encoder, code and decoder. The encoder compresses the input and produces the code, the decoder then reconstructs the input only using this code.

Helpful links:
Applied Deep Learning - Part 3: Autoencoders
`https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af`

An autoencoder is used to reduce the dimensionality. Since there is a lot of redundancy in images, especially if they all contain more or less centered faces, it can be expected that an autoencoder could shrink the input size considerably before using a different neural network. This could potentially make the training process a lot quicker. We would however not be able to use a convolutional neural network anymore because the 2 dimensional image structure is lost by removing the redundancy with the autoencoder. Moreover some information may be lost in the process which could affect the overall accuracy of the neural network.

The effectiveness of an autoencoder could possibly be explored if there are given time constraints for the training process. It can also be compared to principal component analysis (PCA) which removes redundancy in a more controlled way.

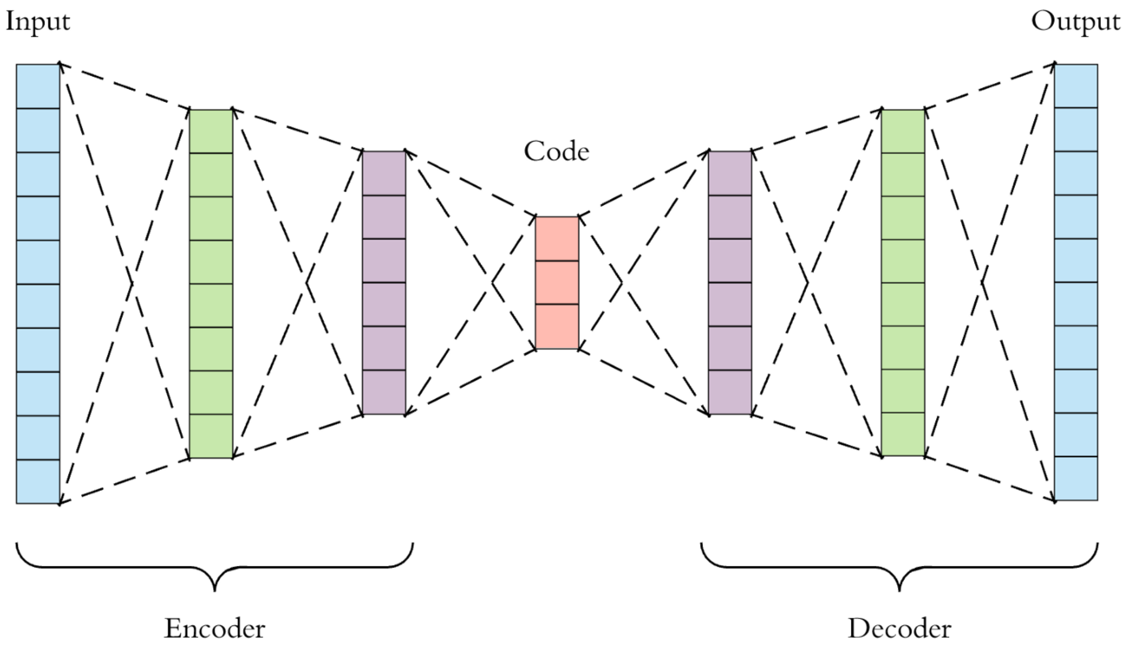


Figure 7: Example of an autoencoder architecture

4.5 Perceptrons MLP

5 Ensemble Methods

The main idea of ensemble is to combine algorithms to achieve a better accuracy than with a single modell. Ensemble methods can be split into heterogene and homogeneous ensembles. In homogenous ensembles only one classifier is used, while in heterogen ensembles different algorithms can be combined. We implemented a homogenous method called adaboost with decision tree stumps(depth=1). This is a boosting method. There are other fields like Bagging(Bootstraping) or Random Forest which are not explained here. For a first introduction with Desicion Trees you can look at https://medium.com/analytics-vidhya/ensemble-methods-for-decision-trees-f4a658af754d.

If we consider decision trees, we can observe that they tend to overfit if they grow to their maximum depth.

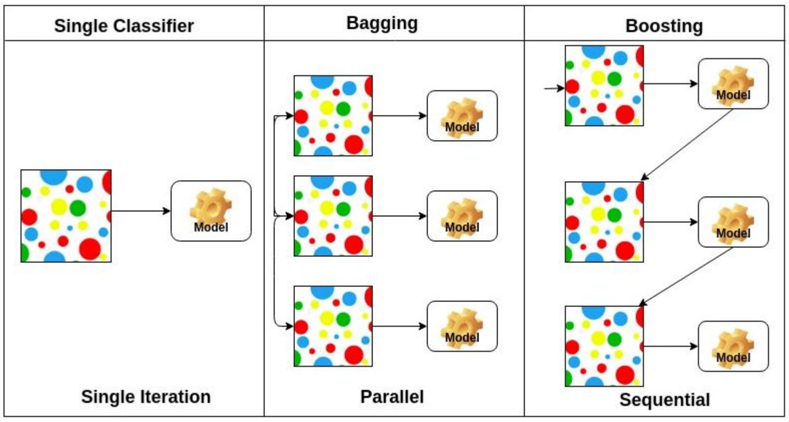


Figure 8: Bagging/Boosting

Also in every node there is uncertainty. Ensemble methods try to build a bunch of weak learner(like the mentioned stumps) to let them predict together and try to decrease variance (and bias).

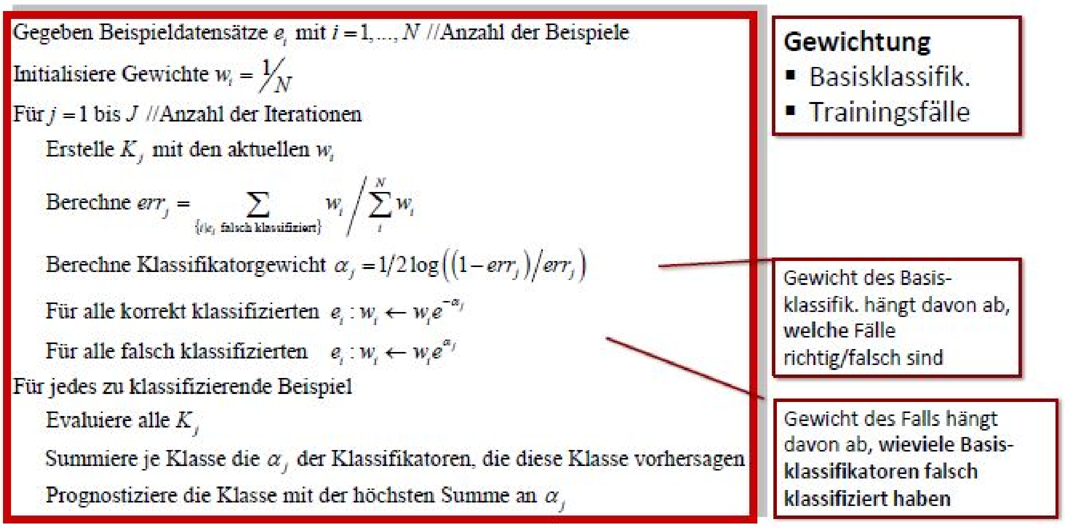
5.1 AdaBoost

For our problem we implemented an algorithm called AdaBoost. The main idea behind AdaBoost is building a series of trees, each of those being an updated version of the previous one. At the end of the process, when all the classifiers built during the iterations will be asked to vote for the target of a new observation, there will be trees with a heavier vote than others. Those are the trees that performed the best during all the iterations (so, they showed very few misclassifications).

- AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.

Like mentioned we implemented Adaboost with Stumps(depth=1). Regarding the functionality of Decision Trees we refere to Chapter 3 Classical learning. The input was the same for like for the other algorithms(2304 pixels) and the emotions. The process of the Algorithm can be seen in the following graphic.

In summary it can be said that Adaboost had a surprisingly good accuracy compared with SVM. But the algorithm tends to classify a lot of pictures as happy and is weak at classifing Angry, Disgust and Fear. One advantage of Adaboost is that it is easily implemented and the training is rather simple. It is also possible to use different weak learner(Trees, KNN, etc.) But if the weak learner selected doesnt fit the prob-



```
Gegeben Beispieldatensätze $e_i$ mit $i = 1, \dots, N$ //Anzahl der Beispiele
Initialisiere Gewichte $w_i = 1/N$
Für $j = 1$ bis $J$ //Anzahl der Iterationen
    Erstelle $K_j$ mit den aktuellen $w_i$
    Berechne $err_j = \sum_{\{i|e_i \text{ falsch klassifiziert}\}} w_i \Big/ \sum_i^N w_i$
    Berechne Klassifikatorgewicht $\alpha_j = 1/2 \log\left((1 - err_j)/err_j\right)$
    Für alle korrekt klassifizierten $e_i : w_i \leftarrow w_i e^{-\alpha_j}$
    Für alle falsch klassifizierten $e_i : w_i \leftarrow w_i e^{\alpha_j}$
Für jedes zu klassifizierende Beispiel
    Evaluiere alle $K_j$
    Summiere je Klasse die $\alpha_j$ der Klassifikatoren, die diese Klasse vorhersagen
    Prognostiziere die Klasse mit der höchsten Summe an $\alpha_j$
```

Gewichtung
- Basisklassifik.
- Trainingsfälle

Gewicht des Basis-klassifik. hängt davon ab, **welche Fälle** richtig/falsch sind

Gewicht des Falls hängt davon ab, **wieviele Basis-klassifikatoren falsch klassifiziert haben**

lem then AdaBoost also struggles solving it. Also with growing iterations Adaboost can overfit. Another disadvantage is that due to the iterative calculation its difficult to parallelize.

Sources:

- Introduction to AdaBoost https://towardsdatascience.com/understanding-adaboost-for-decision-tree-ff8f07d2851
- scikit-learn: https://scikit-learn.org/stable/modules/ensemble.html https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
- Video with an example https://www.youtube.com/watch?v=LsK-xG1cLYA
- Wikipedia: Boosting https://de.wikipedia.org/wiki/Boosting