SAPIENZA
UNIVERSITÀ DI ROMA

Deception Detection Using Facial Action Units

Facoltà di Ingegneria dell'informazione, Informatica e Statistica

Corso di Laurea Magistrale in Informatica

Candidate

Emanuele Orfanelli
ID number 1383726

Thesis Advisor

Prof. Luigi Cinque

Co-Advisor

Dr. Danilo Avola

Academic Year 2017/2018

Deception Detection Using Facial Action Units
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: emanueleorfanelli@gmail.com

Dedicated to
my Family and Friends

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Luigi Cinque for the continuous support of my thesis and related research, for his patience, motivation, and knowledge.

Besides my advisor, I would like to thank Prof. Danilo Avola, Daniele Pannone, Marco Marini, Marco Bernardi, and Cristiano Massaroni, for their insightful comments and encouragement, and for giving me access to the lab and letting me be a part of their team. Without they support it would not have been possible to conduct this research.

I also want to thank my family for the unwavering support they gave me over the years, in my life and through this thesis.

A special thanks to all my friends who have always been there for me, especially Enrico de Francisci, Francesco Andreozzi, Cabio Ficcarelli, Giovanni Garufi, and Stefano Luzi.

Contents

Chapter 1

Introduction

In this chapter we give an overview of the work (par. 1.1). We start by examining how people perform at the task of detecting lies in order to have a comparison with other methods adopted in the filed of machine learning and computer vision (par. 1.2).
We then present a taxonomy of the current state of the art (par. 1.3) concerning deception detection, putting particular emphasis on computer vision research.
The last section is about the structure of the rest of this thesis, and finally our contribution to the field of deception detection (par. 1.4).

1.1 Overview of the work

Deception detection has always been a very interesting topic since it has numerous social and psychological implications in many different fields. In the past 40 years there has been a steady increase in researchers interested in studying deceptive behavior, initially from a sociological and psychological point of view, and in more recent years from a technological standpoint, aided by the advance of machine learning techniques, especially in the field of Computer Vision, an interdisciplinary field that aims to simulate the human visual system to automatically comprehend images or videos.

Our work aims to recognize whether a person is lying or telling the truth by performing a frame by frame analysis on a database of videos, with the goal of extracting a set of the different facial movements performed by the person in the video. After finishing the extraction, the data is submitted to analysis by using statistical and machine learning techniques.

The dataset we are using was provided by Perez-Rosas et al. [**Perez-Rosas:2015:DDU:2818346.2820758**] and is composed by 121 videos taken from real life trials and uploaded on YouTube (par. 4.2). Since those videos are taken from real life events, the illumination, pose, audio and video features are not homogeneous and often substandard. Substantial work was done to eliminate the parts that are not relevant, and as result some videos were trimmed or discarded.

After trimming and cleaning the videos, we split the data into two sets, called the train and test set, based on the subject ID and extracted all the facial movements. Then we utilized machine learning and statistical approaches to understand if the person in the video was lying, based on the results of analyzing each frame.

The result of this work, especially when improved by having a better and larger dataset, can be useful in many different situations, even though it is important to remember that privacy is a real concern, and with this kind of applications it should never be underestimated.

Some of the applications of this work can include airport security, work interviews, and possibly many kinds of social interactions. It could be eventually used by the public to review a speech of a political candidate or by the police force as an aid in the interrogation process, where people lives are at stake and discerning a true or false testimony might be vital. We really hope this work proves to be socially useful.

1.2 People Lie Detection

It's very rare for people to be able to consistently discern between lies and truths, even though we hear lies regularly in the course of our lives. In fact most untrained people perform like chance when tasked with detecting lies [**Porter2012SecretsAL**], and considering that the ordinary person lies at least twice a day on average [**LyingEverydayLife**], and that in this digital age the amount of lies told daily are increasing [**DigitalDeception**] due to on-line communication making us feel more protected and confident in our deceptions, the problem of detecting lies is an important one and it deserves a good deal of research effort.

People's difficulty in detecting lies stem from the lack of objectivity. We are biased by so many factors when deciding weather a statement is a lie or not, and skilled deceivers can take advantage of that.
An important reason is that people generally think it is easy to spot liars, underestimating the effort it takes by having too much confidence in their own judgment and capabilities, and by believing in gut feelings instead of empirical clues [**VrijDLD**].

In a study by Baker et al. [**EmotionallyIntelligent**] 116 participants were asked to judge 20 videos of people pleading for the safe return of missing family members, half of which were lies where the pleader was the one responsible for the disappearance of the victim. The participants provided confidence ratings, the cues they utilized to make their judgment, and their emotional response to each video.
Weirdly, emotionally intelligent people perform worse at deception detection. This is due to their greater sympathetic feelings towards others (enhanced gullibility) that can often cloud their judgment.

In [**BondDePauloAccuracy**] De Paulo et al. analyze the accuracy of deception judgments from a collection of 206 documents and 24.483 judges. They found that people can differentiate lies and truths with an accuracy of 54%, with a lie detection accuracy of 47% and a truth detection accuracy of 61%. Interestingly, their findings reveal that is easier for people to discriminate between lies and truths from the audio cues rather than the visual ones.

Lie Catchers	**# of Lie Catchers**	**Overall Accuracy**	**Lying Detection Accuracy**	**Truth Detection Accuracy**
Diverse (206 documents)	N/A	54%	47%	61%
Undergraduate Students	192	55.2%	61.5%	49%
Police, FBI, CIA, Lawyers, Judges	N/A	50%	N/A	N/A
Secret Service Agents	N/A	70%	N/A	N/A
Police Officers	37	72%	73%	70%
College Students	20	50%	N/A	N/A

Table 1.1. Performance on deceit detection by human observers [**SU201652**].

In another study [**HartwigGranhag**] 192 students obtained an accuracy of 55.2%

on lie detection, with 61% accuracy for guilty suspects and 49% for innocent ones. Police officers and other trained officials seems to perform better than the average person at detecting lies [**VrijPoliceDetect**], obtaining around 70% accuracy for distinguishing both lies and truths correctly, supporting the hypothesis that training and practice have an effect in bettering people at spotting liars. In the table 1.1 above there is a comparison of lie detection accuracy between different classes of people, e.g. college students or police officers.

1.3 State of the Art

How are lies detected? At the moment there are a lot of different instruments and technologies to attempt to detect deceit, ranging from analyzing psychological features, to using the polygraph or thermal cameras and machine learning approaches.

In computer vision specifically lie detection is a somewhat new field of research, and as such it's done using many different approaches, employing not only RGB cameras but also physical sensors and thermal cameras, and using voice, gaze, action and body analysis tools, all of this often aided by machine learning techniques most of the time consisting of support vector machines and neural networks, and by combining many of these modalities together to achieve better results.

We now proceed to describe the state of the art, based on the latest researches done in the field, with specific focus on computer vision:

1.3.1 Speech

Speech is one of the many methods that can be used to recognize if a person is lying. In fact, the speech signal contains linguistic, expressive, organic and biological data [**norena**].

One of the most used indicator of deception in various studies has been the response latency [**EaseLying**], since inventing a lie requires additional cognitive load as opposed to remembering the truth. The authors also notice that habitual lying makes it easier, and conversely often being spontaneous and telling the truth makes lying harder.
Another indicator of lying is the speech rate, especially when it's different from the average rate of a specific person [**TemporalCues**]. Other verbal cues like grammar usage and word frequency have been used and have achieved high accuracy in psychological researches [**PorterTruthLying**].

Speech analysis can reveal changes that affect behavior, such as stress, emotion, deception etc. by analyzing the pitch and the stress level. When a stressful situation arises, the hormonal levels of the body change, and this causes an increase in blood pressure and heart rate. This in turn affects the muscle in the respiratory system, and so our speech pattern is affected and varies from normality [**norena**].

In sound processing, the mel-frequency cepstrum (MFC, fig. 1.1) is a representation of the short-term power spectrum of a sound. Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up the MFC [**wiki:mfcc**].

In [**relidss**] the authors created a new database by making 40 candidates try to deceive them while telling truthful or deceptive statements for about one to two minutes each. From this experiment they extracted MFCC and pitch, so that they could process them through Matlab's Voice Box.
After acquiring the data, an SVM classifier was trained to classify new data, obtaining an accuracy of lie and truth detection from speech audio respectively of 88.23% and 84.52%.

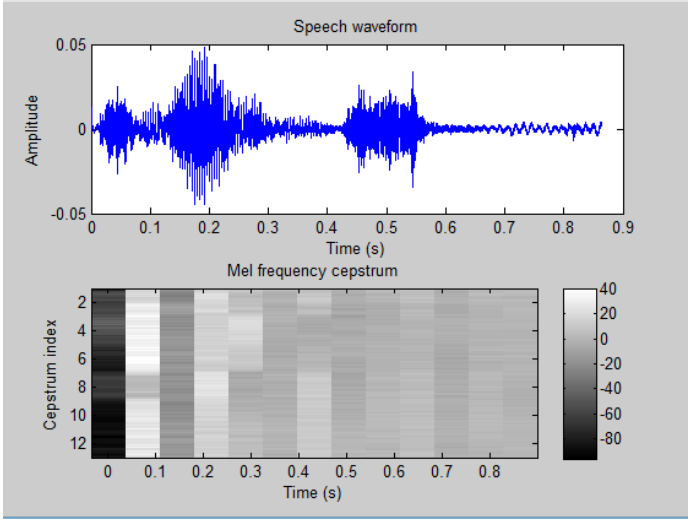


Figure 1.1. Topside is the spoken word, bottom is the MFCC derived from the word.

In [**Mihalcea:2013:ADD:2522848.2522888**] Perez et al. utilize a dataset of 140 videos created by users who uploaded either truthful or deceptive statements. They collected the transcriptions from the videos both manually and via automatic speech recognition software. By using a bag-of-words representation of the transcripts they built a vocabulary, and to classify the data they employed a Support Vector Machine and Naive Bayes classifier, achieving 73% accuracy.

1.3.2 Eyes

Using the eyes to detect lies is one of the most studied approaches, especially in psychological research, as the eyes hold a significant amount of information regarding our thinking and emotional state [**FUKUDA2001239**] (fig. 1.2). We will focus our description on the Computer Vision approaches rather then psychological ones.

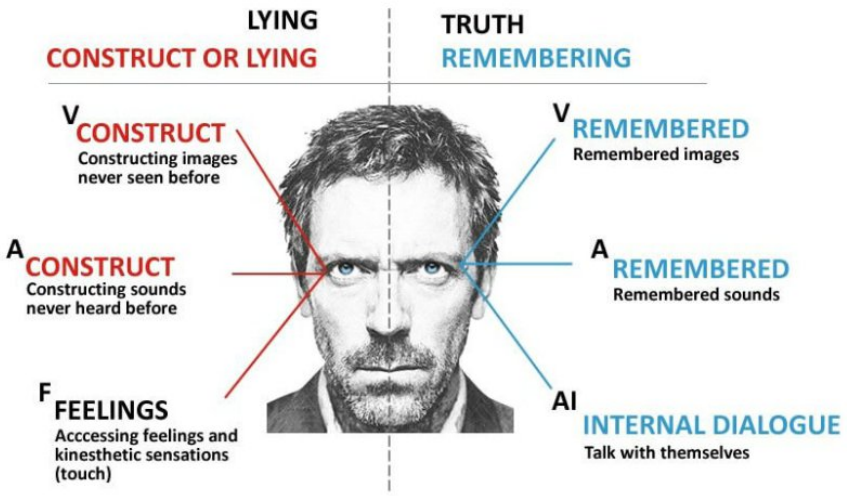


Figure 1.2. The eyes could hold a lot of informations regarding what we are thinking [**eyeLies**].

An important advantage of using the eyes as a vehicle for deception detection is that it is possible to generate a non invasive approach while analyzing them, meaning that subjects do not need to willingly participate or even know if they are being examined or not (the moral matter should be considered in another setting), and there could be no need to have huge and expensive machinery, like for example the polygraph or a fMRI machine (par. 1.3.3).

Cognitive load, which is set to increase while lying, is one of the most significant factor for deception detection and can be analyzed through the eyes. Important factors are also blink rate, gaze aversion and pupil dilation.

In [**8125844**] the authors utilize high speed cameras to record and analyze blink count and blink duration of 50 subjects while asking 10 control questions, to see if there is a variation in them while the subject is being questioned. The authors analyzed the resulting images frame by frame and based on the facial landmarks around the eye they recognize AU45, the action unit (muscle movement par. 1.3.7) associated blinking. The results shows that both blink duration and count are increased while lying.

In another study, Leal and Vrij [**Leal2008**] gathered 26 people, 13 liars and 13 truth tellers, to ask them to lie or tell the truth in a target period, while already having a baseline from two preceding periods. The eye blinks during the target and baseline

periods and directly after the target period (target offset period) were recorded. Compared to the baseline periods, lying subjects show a decrease in eye blinks during the target period and an increase in eye blinks during the target offset period. This means that there is a decrease in eye blinks while lying and an increase just after lying. This pattern resulted very different for truth tellers, showing that there is a significant difference in eye blink behavior between truthfulness and deception.

Singh et al. in [**7324092**] show that while lying there is an increase in cognitive load and a significant decrease in eye blinks, directly followed by an increase in blink rate as soon as the cognitive demand ceases, after telling the lie. A threshold is set by the authors for this study, either at 26 blinks per minute or it is calculated ad personam using the average blink rate from a blink detection algorithm. Blink detection is done through MATLAB using the HAAR Cascade algorithm.

Lim et al. study eye gaze [**Lim:2013:LTE:2535948.2535954**] to investigate the relation with lie detection. The result supports the theory that an increase in cognitive load leads to a decrement in the number of eye movements while lying.

Bhaskaran et al. measure deception by the deviation from normal behavior at critical points during an investigative interrogation [**5771407**] . For starters, a dynamic Bayesian model of the eye movement is trained during a normal conversation with each of the 40 subjects of the experiment.
The remainder of the conversation is then broken into pieces and each piece is tested against the normal behavior. The deviation from normality are observed during critical points in the interrogation and used to deduce the presence of deceit, obtaining an accuracy of 82.5%.

In [**7165946**] Proudfoot et al., using latent growth curve modeling, research how the pupil diameter changes over the course of an interaction with a deception detection system. The assumption is that anxiety changes the pupil diameter. The subjects are presented with crime-relevant target items (possibly incriminating) and non relevant items.
The results indicate that the trends in the changes are indicative of deception during the interaction: pupil diameter is initially bigger for guilty subjects relative to innocent ones. Also the pupil diameter of the participants decreased between subsequent sections of the test, specifically the magnitude of the decrease was greater for guilty subjects who did not see incriminating items.

1.3.3 Neuroscience

Electroencephalogram (EEG) and functional Magnetic Resonance Imaging (fMRI) have been employed for lie detection with decent results for a long time, but at the cost of invasiveness, since both methods require big machinery, a suitable environment, and subjects willing to participate by having electrodes and other monitoring tools attached to their head and body.

EEG is a monitoring method that records brain activities based on its potential.

In [**7440177**] Simbolon et al, the authors use Event Related Potentials (ERP) to measure brain response directly from thought or perception. Among the many types

of signals that constitute the ERP signal, P300 (fig. 1.3) is the most critical for lie detection, as it appears as a response to meaningful rare stimuli (called odd ball stimuli).
Eleven males of age between 20 and 27 took part in the study. The gathered data were then divided into training and test sets to produce different models. The highest accuracy of 70.83% was reached by a SVM classifier in detecting lying subjects.

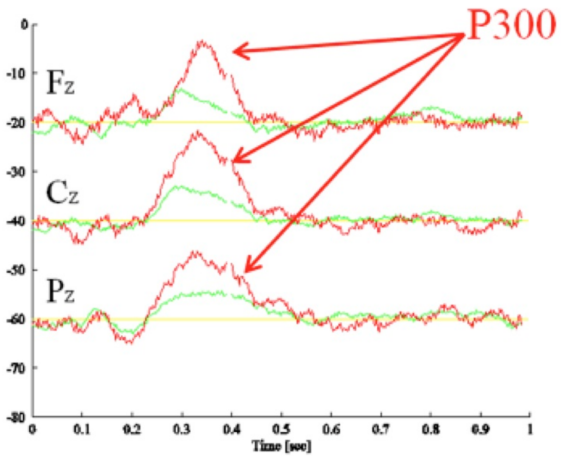


Figure 1.3. EEG of P300 waves image on channels Fz, Cz, and Pz.

In the study performed in [**Lai2017**] twenty people of ages between 22 and 24 years old were subject to a card test using an EEG machine. EEG signals were collected using electrodes attached to the subject's head. The authors used the EEG signals to identify useful frequency bands and to measure the lying state based on spectral analysis, with the use of fuzzy reasoning, obtaining 89.5% detection accuracy.

Arasteh et al. [**7511728**] utilize an alternative approach to the polygraph, the P300 Guilty Knowledge Test (GTK). GTK is based on the amplitudes of the P300 ERP wave as an index for the subject's recognition of concealed information. The Guilty Knowledge Test works on the assumption that among many similar unfamiliar topics, the recognition of familiar ones will be followed by a different response.
62 subjects were part of this experiment and participated in a mock crime followed by the P300 Guilty Knowledge Test. The authors used empirical mode decomposition (EMD) to extract features from the EEG signal and modeled them trough matlab. A genetic algorithm was then utilized for the feature selection and to handle the dimensionality increase of this approach. The classification accuracy of guilty and innocent subjects was 92.73%.

Another neuro-scientific approach to lie detection is functional Magnetic Resonance Imaging (fMRI). fMRI measures brain activity by detecting changes associated with blood flow (fig. 1.4). This technique relies on the fact that cerebral blood flow and neuronal activation are coupled. When an area of the brain is in use, blood flow to that region also increases [**WikifMRI**].

In most of the experiments with fMRI and lie detection, candidates are instructed to lie and tell the truth in specific situations, and the brain activity from these instances

is compared to a baseline condition. The regions showing greater activation for lies than truth are supposed to be the most significant for deception detection. In a recent study it is shown that there is considerable agreement on the significant areas of the brain that regard lying [**fMRILD**].

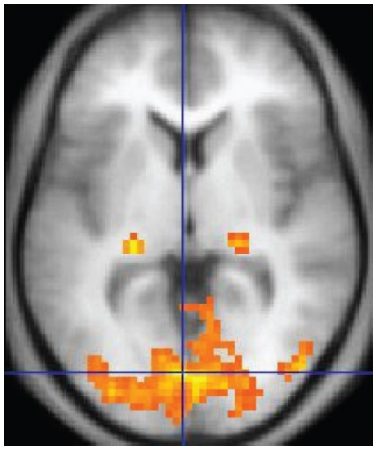


Figure 1.4. fMRI image with yellow areas showing increased activity compared with a control condition [**WikifMRI**].

As much as fMRI is useful for lie detection, it presents some shortcomings [**fMRIDD**] [**fMRIDA**]: many fMRI studies are small, not replicated and done with just a few subjects; there are some contradicting results between some studies; most of the studies are not done in a contest of high stake deception, but in a controlled environment where subjects are asked to lie about some topic or event, but often without a real interest in being deceitful. Another important point is that the fMRI approach requires collaboration and expensive equipment to be carried out, so this turns out to be a very limiting factor.

1.3.4 Thermal Imaging

In thermal imaging, thermal features are extracted from the face using a high definition thermal camera. The objective is to analyze what kind of differences occur when a subject responds truthfully or deceptively to particular stimuli.

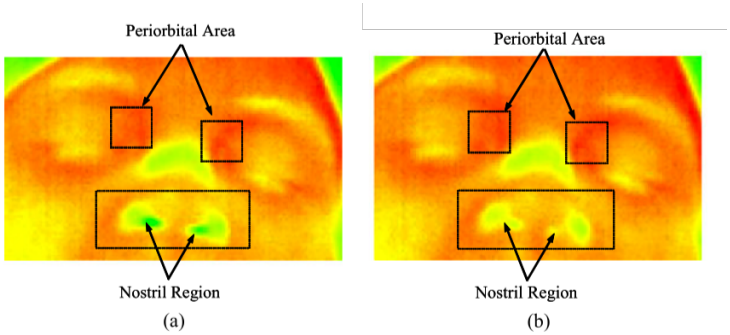


Figure 1.5. Examples of thermal images during (a) questioning and (b) answering [**6967765**].

According to a recent study [**Abouelenien:2015:TAD:2823465.2823470**] examining 30 subjects, the most relevant zones for deception detection, seen by utilizing thermal equipment, and located in the facial area are the forehead and periorbital regions.
In this study the subjects were registered with a thermal camera for one minute to extract the baseline features, and after that the interviewers asked a series of questions. A thermal map was created from the registrations using the Hue Saturation Value to differentiate between lies and truths.

In [**Rajoub**] the authors set up an experiment to collect 492 responses from 25 participants, using a deception scenario requiring the subjects to learn a story, provided by the authors, and practice their stories before an interview by giving them sample questions, so that the cognitive load would increase when receiving new and unseen questions during the interview process.
At the beginning of the interview four baseline questions were asked to register the initial thermal state of the subjects, and then a series of questions were proposed, with answers both present and missing from the previously provided story. After extracting the periorbital region's thermal variation, a k-nearest neighbor classifier was used, with an 87% accuracy in predicting lies or truths.

In [**6967765**] data are gathered non-intrusively from the nostril and periorbital regions using two dimensional far infrared cameras. The study lasted for two years and covered 18 tests subjects involved in real crime cases. The temperature is extracted and converted in change in blood flow velocity, and a signature of the respiration pattern is determined in terms of the ratio of the measured maximum and minimum temperatures in the nostril area. The classification rate for this study is 88.5%.

1.3.5 Multimodal

After seeing all these different techniques to detect lies, it's only natural for researchers to try and fuse or aggregate some of them to see if it's possible to achieve better results [**Abouelenien:2014:DDU:2663204.2663229**].

In [**Abouelenien:2016:ATV:2910674.2910682**] Abouelenien et al. collect data from a dataset of 30 subjects to examine thermal and visual clues of deception. Their aim is to identify the regions that offer higher capability of detecting deceit. The method employed uses the CERT (Computer Expression Recognition Toolbox) to detect facial expression and encodes them through Action Units (explained in greater detail in par. 1.3.7), a way to codify almost any kind of muscle movements on the face.
To extract thermal features they create a thermal map using gray-scale and Hue Saturation Value. They also calculated normalized blinking rates and the mean head orientation angle. In addition, over 60 physiological features are extracted and stored with the use of other sensors and RGB cameras.
The experimental results show that the non-contact feature fusion model outperforms traditional physiological measurements, and that the forehead region is one of the most promising areas to gather information for deception detection using thermal

imaging.

In a following paper [**7782429**] Abouelenien et al. explore a multimodal deception detection approach comprised of physiological, linguistic, and thermal features on a new dataset of 149 recordings. They set out to determine which is the most discriminative region of the face to differentiate between truth and lies based on thermal imaging, and perform feature analysis using a decision tree model. The results show that the forehead could be a better indicator of deceit than the periorbital area. The physiological features did not contribute very much, while the linguistic feature played a critical role, where self-referencing and exaggeration words where the majors indicators of deceit. The overall accuracy of the system is 70%.

Another example of multimodal detection is found in [**DBLP:journals/corr/abs-1712-04415**] where Wu et al. develop a framework to automatically detect deception in videos of trials. They utilize three modalities: visual, audio and text. For visual, they employ various classifiers trained on low level video features to predict human micro-expressions, and to successively predict deception. Improved Dense Trajectory (IDT) features, often used to recognize actions in videos, are good predictors of deceptive behavior.
The authors decided to fuse the score of the classifiers on IDT and micro-expressions to boost the performance.
For speech, they integrated the vision side with Mel Frequency Cepstral Coefficient (MFCC) features analysis from the audio, boosting the performances significantly, reaching an AUC of 0.877.

Noje et al. [**7367432**] set up a study with ten subjects to observe head movements in lie detection. They build an application to detect head movements and position by performing a frame to frame analysis of a video stream. A correlation was found between the head movements and position and the identification of lies, confirming what the psychological theory suggests. The authors suggest that this data would be more revealing if incorporated with other modalities such as voice, gaze, words, expressions et cetera.

In [**Perez-Rosas:2015:DDU:2818346.2820758**] Perez et al. utilize real life trial videos to identify deception, employing a model that extracts features from both linguistic and gesture modalities. The verbal features consist of unigrams and bigrams derived from the bag-of-words representation of the video transcripts, while gesture features are taken from the annotations performed using the MUMIN coding scheme, where each feature indicates the presence of a gesture only if it's present in the majority of the video. Using Decision Trees and Random Forest classifiers they achieve between 60% and 75% accuracy.

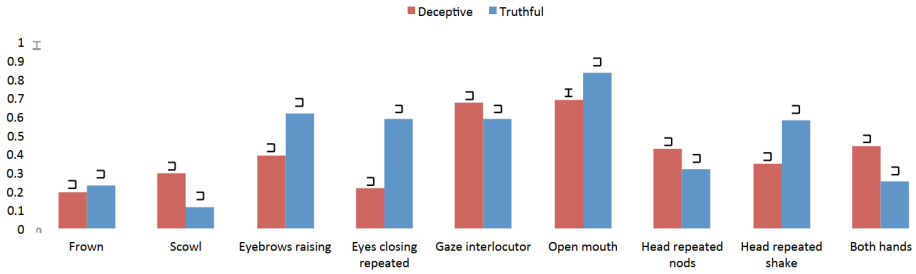


Figure 1.6. Distribution of non-verbal features for deceptive and truthful groups [**Perez-Rosas:2015:DDU:2818346.2820758**].

1.3.6 Facial Expression and Micro-Expressions

Facial Expressions are one of the main methods that we use to express our emotions, and are developed since the first months of our life to help us communicate our feelings to and towards others. But what happens when we want to hide our emotions instead?

Facial micro-expressions are very fast (from 1/2 to 1/25 of a second) and involuntary expressions that come up on the human face when we are trying to suppress or hide an emotion, and are very difficult to control using just one's willpower and without explicitly training for it [**EkmanER**].

Figure 1.7. Six basic Facial Expressions in adults and children [**baby_fe**].

Studying and classifying micro-expression is very valuable and has many applications, especially in psychology and forensic sciences, but it is a hard and complex feat as the duration is very short and the intensity is generally very low. Recognizing micro-expressions using the human eye, without further help from a computer has proved to be a challenging task.
Micro-expressions have been studied since 1966 to recognize and distinguish real or fake emotions, initially by Haggard and Isaacs [**Haggard**], and three years later by Ekman and Frisen [**EkmanLeakage**].

Substantial work on Micro-Expressions has been done in recent years by Pfister, Li et al. In [**pfister2011micro**] they collaborated with psychologists to design an induced emotion suppression experiment. The data was collected with a high speed camera to be able to register the micro expressions of the subjects. A Temporal interpolation model was used to counter the shortness of the video length, while multiple kernel learning, random forest and SVM were used to classify micro expressions reaching an accuracy of 71.4%.

The lack of a big and well formed database was one of the biggest hindrance to their research. To solve this problem in [**xli2012spontaneous**] they reveal a new dataset, the Spontaneous Micro-expression Database (SMIC), which includes 164 micro-expression video clips taken from a group of 20 participants.

They used two high speed cameras to record the face of the subjects while they were watching a selection of videos that induced strong emotional responses, and they had to try and suppress those emotions. After the video the subjects had to answer about the emotions they felt while watching it. The data were then segmented and labeled by two annotators.

A study of spontaneous micro expression spotting and recognition methods was done in [**xli2015reading**]. A new training-free method, based on feature difference contrast for recognizing micro-expressions was presented. The features are extracted from the video using Local Binary Pattern (LBP) and Histogram of Optical Flow. To recognize the Micro expressions, the authors performed face alignment and temporal interpolation, then trained an SVM classifier.
This micro-expression framework was tested on the SMIC and CASMEII database with very good results. After combining micro-expressions spotting and recognition they released a new micro-expression analysis system (MESR) that is able to recognize micro-expressions from spontaneous video data.

Owayjan et al. [**6462897**] designed a lie detection system using micro-expressions. At first an embedded video system is used to record the subject interview. The video stream is converted into frames, and each frame is processed in four stages: converting the images, filtering out useless features, applying geometric templates and finally extracting the measurements to detect the micro-expressions.
Results show that up to eight facial expressions can be recognized, and that lies can be discerned with high precision.

In [**10.1007/978-3-319-47955-2_27**] Kawulok et al. explore how to exploit fast smile intensity detectors to extract temporal features using a SVM classifier. Using exclusively a face detector, without localizing or tracking facial landmarks, they analyze the smile intensity time series. They then employ an SVM classifier to improve training from weakly labeled datasets. Then, to train the smile detectors, they use uniform local binary pattern features. This allows to detect, in real time, between spontaneous or posed expressions.

Su et al. [**SU201652**] aim to test the validity of facial clues to deception detection in high-stakes situations using computer vision approaches. By using invariant 2D features from nine separate regions of the face they perform facial analysis on eye blink, eyebrow motion, wrinkle occurrence, mouth motion, and integrate them with a facial behavior pattern vector. Training a Random Forest to classify the patterns into deceptive or truthful, they achieved a 76.92% accuracy.

1.3.7 Action Units

Action Units (AUs) are defined as a contraction or relaxation of one or more muscles. They have been used in the Facial Action Coding System (FACS), a system developed initially by Hjorztsjö [**facsCH**] and then improved by Freisen and Ekman [**facs1978**], that categorizes all the facial movements by their appearance on the face.

Using FACS it's possible to code nearly any facial expression by deconstructing them into Action Units. For example the Duchenne smile (felt smile) is a combination of

AU6 (orbicularis oculi, pars lateralis) and AU12 (zygomatic mayor) as we can see from fig. 1.8.

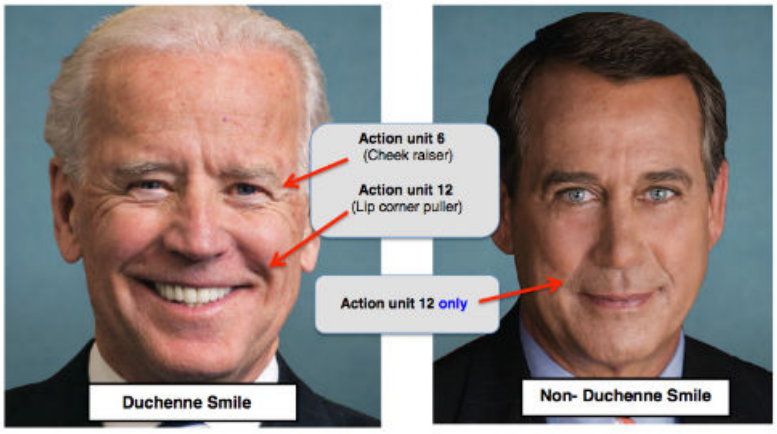


Figure 1.8. Duchenne smile (real) vs non Duchenne (fake).

By using AUs is possible to recognize emotions based on the combination of AUs displayed (fig. 1.9).

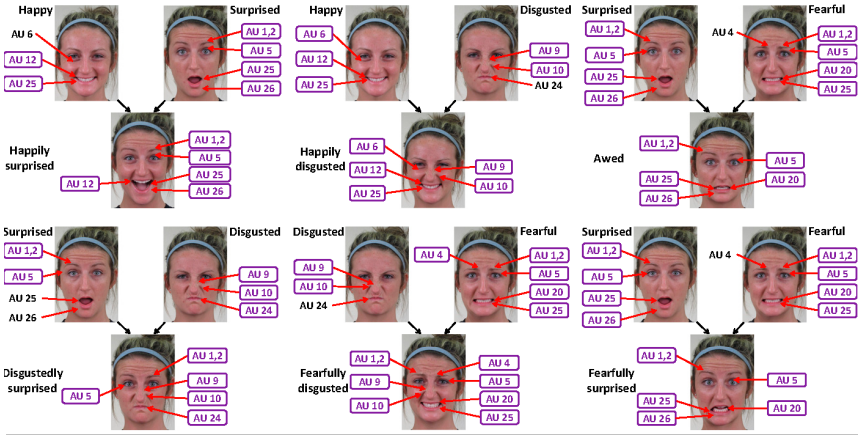


Figure 1.9. AUs of six compound facial expressions of emotion. The AUs of the basic emotions are combined to produce the compound category [**Du2014CompoundFE**].

In the FACS there are 44 AUs categorized with five level of intensity (A to E). Most of the AUs have an onset, peak and offset phase, as shown in fig. 1.10.

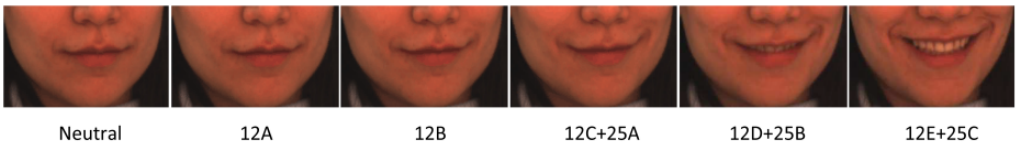


Figure 1.10. AU intensity variations (AU12: lip corner puller, AU25: lips part) [**DISFA**].

Substantial work on AU classification and intensity estimation has been done in [**Baltru2015**] by Baltrusaitis et al. while developing the OpenFace [**Baltru2016**] system.
Baltrusaitis et al. developed a real-time Facial Action Unit occurrence and intensity detection system based on appearance and geometric features, using Histogram of Oriented Gradients, Shape Parameters and Landmark Locations. They achieved good results by using a median based feature normalization technique so that they could account for person specific neutral expressions.

In [**HaoWangAU**] Hao et al. explore the dependencies between AUs, and propose a new AU recognition method consisting of a three layer Bayesian network where the top two layers are Latent Regression Bayesian Networks (LRBN).
LRBN are graphical models consisting of a visible layer representing the ground truth for AUs and a latent one. LRBN is able to capture the dependencies between AUs. They tested this system on the CK+ [**CK+**], SEMAINE [**SEMAINE**] and BP4D [**BP4D-Spontaneous**] database, demonstrating that their approach can accurately capture AU relationships.

In [**AU_LSTM**] the authors try to model three fundamental aspects of automated AU detection: spatial representation, temporal modeling, and AU correlation. They proposed an approach using a hybrid network architecture.
Spatial representation is extrapolated by using a Convolutional Neural Network (CNN). For temporal dependencies Long Short-Term Memory Neural Networks (LSTM) are stacked on top of the CNN.
The output of LSTMs and CNNs are then fused together to predict 12 AUs on a frame to frame basis. This network was then tested on the GFT and BP4D [**BP4D-Spontaneous**] dataset, gaining improvements over standard CNN.

De la Torre et al. [**AU_STM**] [**AU_STM2**] tackle this problem by personalizing a generic classifier without requiring additional labels for the test subject (unsupervised). They use a transductive learning method, referred as Selective Transfer Machine (STM), that personalizes a generic classifier by attenuating people specific biases. This is done by learning a classifier and re-weighting the training samples most relevant to the subject.
The performance of STM were compared to generic classifiers and cross-domain learning methods and evaluated on CK+ [**CK+**], GEMEP-FERA, RUFACS and GFT dataset, and the Selective Transfer Machine approach is shown to outperform the generic classifiers on all datasets.

Although there has been a lot of progress during the years, there is still a lot of work to be done regarding AUs, since many approaches to automatic facial AU detection fail to account for individual difference in morphology and behavior for the target person. This is a hard problem because it's difficult for classifiers to generalize to very different subjects, and training a person-specific classifier is neither feasible (very low training data) nor there is particular theoretical interest in the research community.

1.4 Contributions and Outline

1.4.1 Contribution

The human face holds many clues to recognize emotions and to discern truth and lies. Human emotions can be recognized mainly by the expressions we make when a situation arises, such as being scared, happy, confused, angry and so on. Those expressions are represented by movements of the muscles of the face. Building on this idea, the objective of this thesis is to classify weather a person is lying or telling the truth based on the action units, muscular movements appearing on the face, performed by the subjects and extracted from a video stream.

It is important to note that all the videos are taken from high stakes situations, specifically court trials, so that all the liars and truth tellers have a big incentive in lying or telling the truth, since the validity of many studies is questionable based on the fact that the emotions analyzed are posed (non spontaneous).

As far as we know this is the first time a study uses action units extracted from a video and processed through an SVM to classify between truth and lies in high stakes situations.

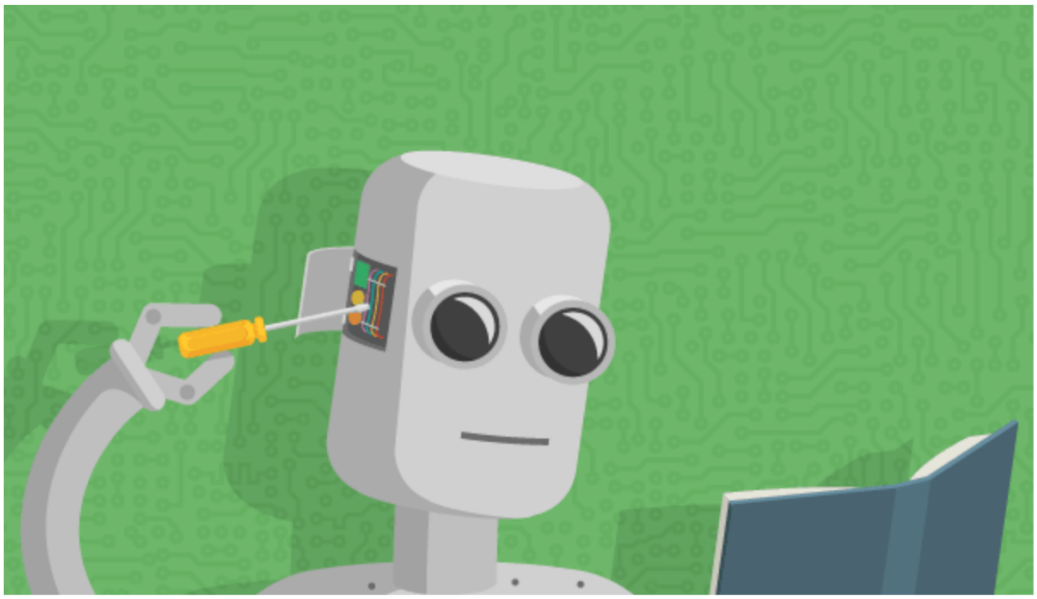
1.4.2 Outline

This thesis is composed by the following chapters:

- **Chapter 2** is about Machine Learning. We will talk about classification, regression, supervised learning and some of the most utilized techniques such as Random Forest and SVM;
- **Chapter 3** will describe the architecture of our work, explaining landmark detection, action unit intensity and presence estimation, and the process to discriminate between deception and honesty;
- **Chapter 4** discusses the dataset and explain the experimental procedure of training and testing a binary classifier for discerning truth and lies. We then proceed discuss the results of the work;
- **Chapter 5** is where we draw the conclusions, and then we present the future improvements we plan to implement and discuss new features.

Chapter 2

Machine Learning



In this chapter we will give a brief introduction of Machine Learning (par. 2.1), explaining Classification (par. 2.1.3), Regression (par. 2.1.4) and Supervised learning (par. 2.1.1), proceeding then to explain different techniques such as Random Forests (par. 2.3) and SVM (par. 2.4), which is the most important technique used for this thesis.

2.1 Introduction

Machine Learning is a subfield of Artificial Intelligence that uses statistical techniques to provide computers with the ability to progressively improve performance on different tasks using data, while not being explicitly programmed [**wiki:ml**].

The applications for machine learning are huge and diverse, and range from character recognition to email filtering, with lots of applications in computer vision, such as image recognition and classification.

Machine learning also focuses on making prediction on data, performed by utilizing techniques taken from statistics and mathematical optimization. This has many applications, ranging from health care by predicting risk factors for diseases or gaining insights for prevention, to sports or politics where actual results of games or elections can be predicted.

The field of machine learning is subdivided in two broad categories (fig. 2.1), supervised learning (par. 2.1.1) and unsupervised learning (par. 2.1.2). This division is made based on whether the data is labeled or not.

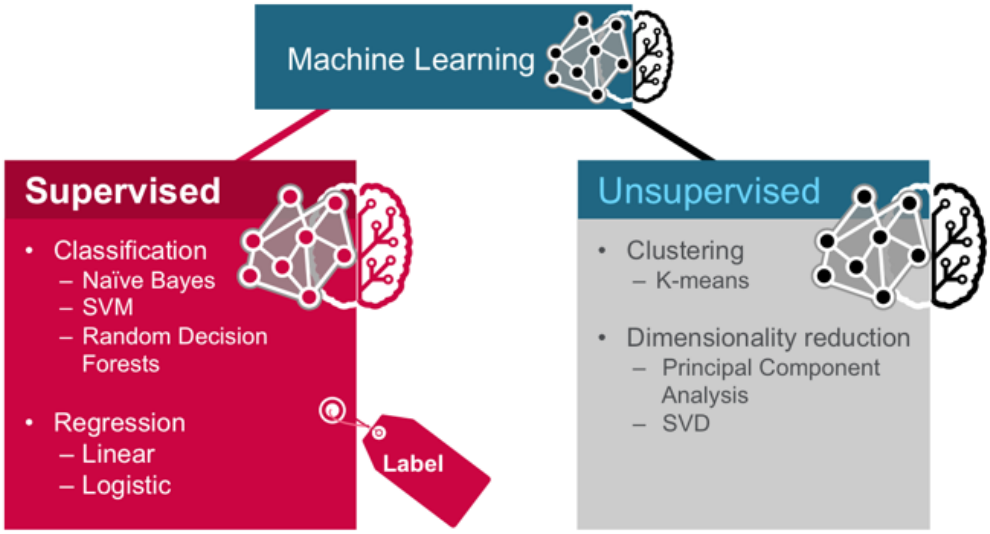


Figure 2.1. Machine Learning Subfields [**ml_mldiv**]

Another subdivision of the field of machine learning is based on the kind of output that the users wish to obtain. The categories of said output are:

- **Classification** (par. 2.1.3) for predicting a discrete class label for a new example data;
- **Regression** (par. 2.1.4) to predict a continuous quantity;
- **Clustering** to group similar but unlabeled data.

2.1.1 Supervised Learning

When using a supervised learning method we want to learn a function that maps an input to an output, based on example input-output pairs [**ai_sup**].
This means that we need labeled training data, consisting of a vector of input objects and an output value. The objective is to correctly classify the new data based on the previously analyzed training pairs.

The general steps to solve a supervised learning problem are (fig. 2.2):

1. Understand what kind of training example to use;
2. Gathering a training set;
3. Model the training set to be fed as input to the algorithm by choosing which features to use and how to represent the data;
4. Choose what kind of algorithm can best train the model;
5. Run the algorithm and evaluate the resulting accuracy on the test set.

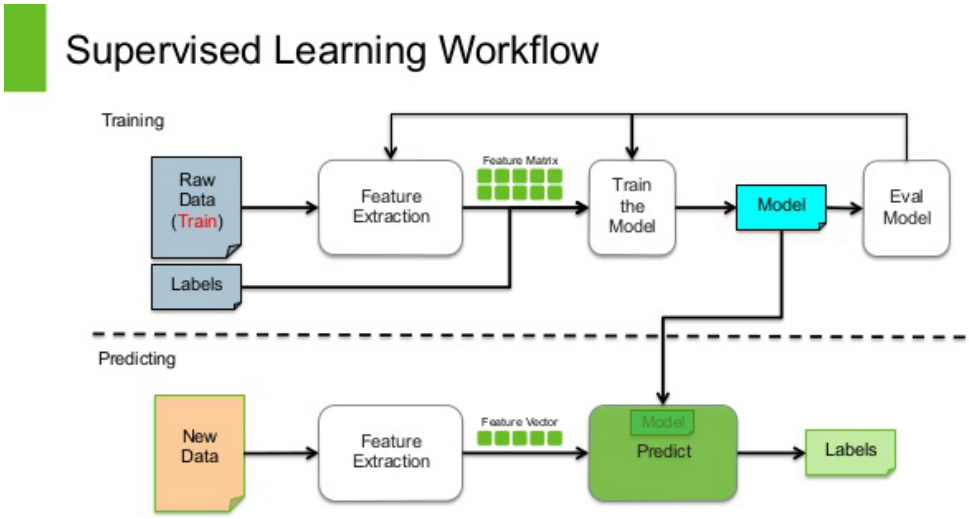


Figure 2.2. Supervised Learning Workflow [**sup_wf**].

There are important considerations to make when using a supervised learning approach:

- **Bias-Variance Tradeoff** [**biasvar**]:
  Suppose we have many different (but equally good) training sets:
  an algorithm is biased for input x if, when trained on each of these data sets, it is consistently inaccurate at predicting the correct output for x.
  A high bias indicates that the data points tend to be very close to the mean, and to each other.

  A learning algorithm has high variance for a particular input x if it predicts output values that are correct on average but inconsistent when trained on different training sets.
  Basically, a high variance indicates that the data points are very spread out from the mean, and from one another (fig. 2.3).

  The prediction error of a classifier is related to the sum of bias and variance, so generally there is a tradeoff between them. Low bias means that a classifier fits the new data well, but if the bias is too low it will fit each training set differently and so result in high variance.

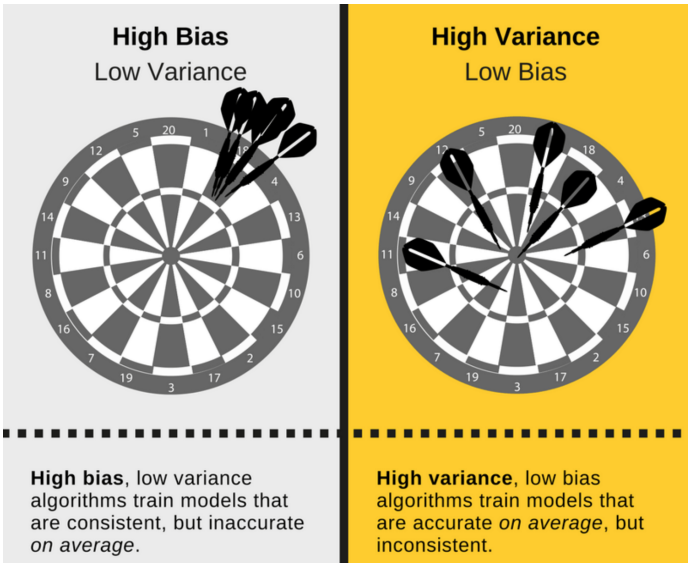


Figure 2.3. Visual explanation of Bias-Variance Tradeoff [**biasvarTradeoff**].

- **Dimensionality of Input** [**wiki:dim_red**]:
  When the input feature vectors are dimensionally very big there could be problems in learning the function, even if not all features contribute significantly to it. This happens because the data depends on too many variables and this could cause high variance.
  To avoid this, it is important to reduce the number of features through manual removal or using feature selection algorithms such as Principal Component Analysis (PCA). This usually improves the accuracy of the classifier.
- **Overfitting and Underfitting** [**overfit**]:

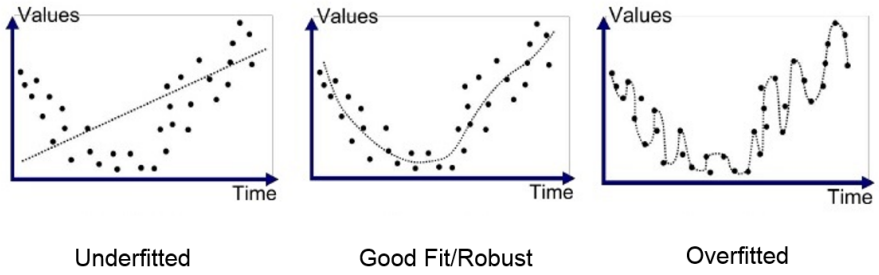


Figure 2.4. Example of overfitting and underfitting [**underfitting**].

Overfitting happens when the algorithm adapts too much on the training data and is no longer able to make accurate predictions on the test data (fig. 2.4). This usually happens when there is an excessive number of parameters than what can be justified by the data [**camb_over**].

Underfitting is the opposite: a model is not able to correctly capture the structure of the data, for example when fitting non linear data with a linear

model.
A common way to avoid overfitting is to resample the data using different techniques, commonly k-fold cross validation or leave-one-out. Other methods include feature removal, early stopping and regularization.

There are many algorithms used to perform supervised learning tasks, the most commonly used are:

- Linear Regression (par. 2.1.4);
- Logistic Regression (par. 2.2);
- Naive Bayes;
- Linear Discriminant Analysis;
- Decision Trees;
- k-Nearest Neighbor;
- Neural Networks;
- Support Vector Machines (par. 2.4).

2.1.2 Unsupervised Learning

Unsupervised learning is the subfield of Machine Learning tasked with inferring a function from the analysis of unlabeled (not classified) data. Being unclassified there is also a difficulty in evaluating the accuracy of the model.
Usually items are grouped by some measure of similarity, like for example in k-means clustering (fig. 2.5).

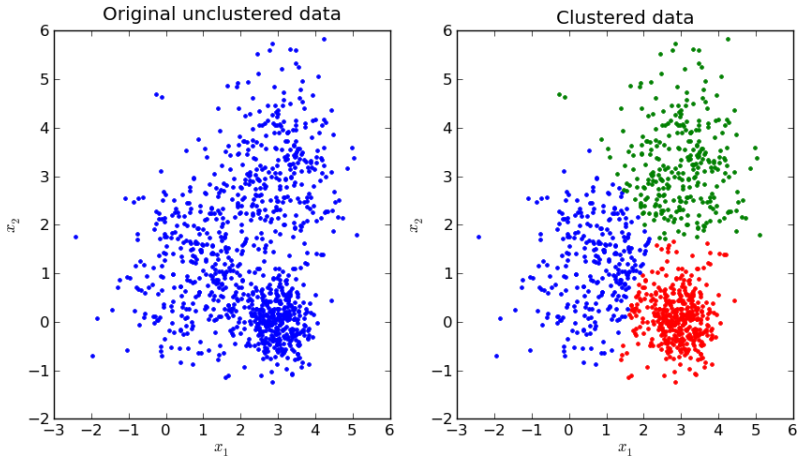


Figure 2.5. Example of Unclustered and Clustered data [**kmeans**].

These are the most widely used unsupervised learning algorithms:

- Clustering
  - k-means;
  - mixture models;
  - hierarchical clustering;
- Neural Networks:
  - Autoencoders;

  - Deep Belief Nets;
  - Hebbian Learning;
  - Generative Adversarial Networks;
  - Self-organizing map;
- Expectation–maximization algorithm (EM);
  - Principal component analysis;
  - Independent component analysis;
  - Non-negative matrix factorization;
  - Singular value decomposition.

2.1.3 Classification

In machine learning, classification is the problem of identifying in which of a set of categories a new observation belongs, based on a training set of data containing observations whose category is known in advance. A common example could be classifying an email as spam or not [**wiki:classification**].

Classification is considered an instance of supervised learning, based on instances where a training set is available. As for unsupervised learning, classification would be clustering since it groups data into categories based on similarity, but without knowing the label of the data.

The observations, called features or explanatory variables, take different types based on the value. They can be categorical, numerical or ordinal, or be compared by similarity between previous observations using some kind of distance function.
The observations representing the categories to be predicted are called explanatory variables (or regressors, or independent variables).

The classifier is the algorithm that implements the classification, or a function that maps input data to a category.

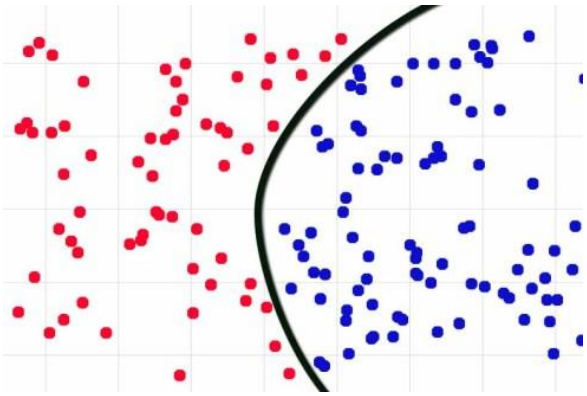


Figure 2.6. Example of data division by a classification algorithm.

Many classification algorithm, such as Support Vector Machine (par. 2.4), logistic regression (par. 2.2), Linear Discriminant Analysis (LDA) or perceptron, can be described using a linear function assigning a score to each category c, and doing the dot product of the feature vector of an instance with a vector of weights, thus

combining them. The predicted category will be the one with the highest score. This function is called linear predictor function and has this general formula:

$$score(X_i, c) = \beta_c \cdot X_i \tag{2.1}$$

where X_i is the feature vector of instance i and β_c is the vector of weights of category c. This kind of algorithms are known as linear classifiers.

2.1.4 Regression Analysis

Regression analysis is used in statistics and machine learning to estimate the relationships among variables.
More specifically, it focuses on the relationship between one dependent variable and more independent variables (also called predictors), and generally is used to estimate the average value of the dependent variable when the independent variables are fixed, by calculating a regression function.

By understanding the relationship between dependent and independent variables it's also possible to perform predictions on future data based on new inputs, or on changes to the old ones [**wiki:reg_an**].

In regression analysis, it is also of interest to characterize the variation of the dependent variable around the prediction of the regression function using a probability distribution.

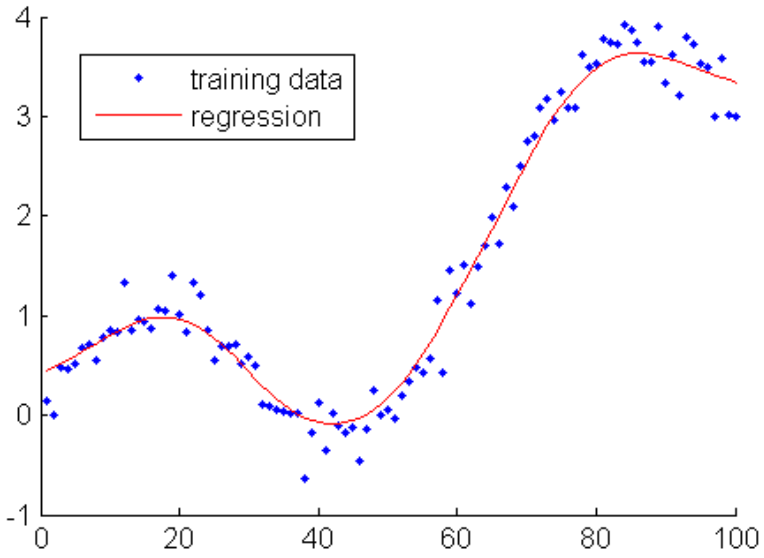


Figure 2.7. Example of Regression Analysis

Regression Model

A general regression model uses the following variables and parameters:

- The unknown parameters, β, that represents either a scalar or a vector.
- The independent variables, X.
- The dependent variable, Y.

A regression model relates Y to a function of X and β [**wiki:reg_an**].

$$Y \approx f(X, \beta) \tag{2.2}$$

This is usually formalized as:

$$E(Y|X) = f(X, \beta) \tag{2.3}$$

If β is of length k and the number of observed data points is enough ($N > k$), then it's possible to estimate a unique value for β that best fits the data.

If this is the case, regression analysis provides the means to find a solution for unknown parameters of β to, for example, apply the method of least squares.

Generally, to apply a regression analysis, the data must abide by some assumptions [**wiki:lin_reg**]:

- The sample is representative of the population for the inference prediction;
- The error is a random variable with a mean of zero;
- The independent variables are measured with no error;
- The independent variables (predictors) are linearly independent;
- The errors are uncorrelated;
- The variance of the error is constant across observations.

2.1.5 Linear Regression

Linear regression (LR) is the most basic case of Regression Analysis, and it is a useful tool for predicting a quantitative response. Also, most of the newer approaches are often a generalization or an extension of the linear regression model.

LR is a linear approach to modeling the relationship between a dependent variable and one or more independent variables. The simplest case of linear regression is when there is only one independent variable, this is called simple linear regression [**wiki:lin_reg**], and has the following form:

$$y_i \approx \beta_0 + \beta_1 x_i + \varepsilon_i \tag{2.4}$$

where ε_i is the error for the ith observation. The error is a catch-all for what we miss with this simple model, since it's very probable that the true relationship is not linear, and that there may be other variables that influence y [**ISLR**].

The aim of LR is to estimate the β coefficients to make predictions. To do so, we utilize of the training dataset to produce estimates $\widehat{\beta_0}$ and $\widehat{\beta_1}$ for the model coefficients. We can then make predictions by computing:

$$\widehat{y_i} \approx \widehat{\beta_0} + \widehat{\beta_1} x_i \tag{2.5}$$

$e_i = y_i - \widehat{y_i}$ is the difference between the true value and the prediction for an observation, and it is called residual (fig. 2.8).

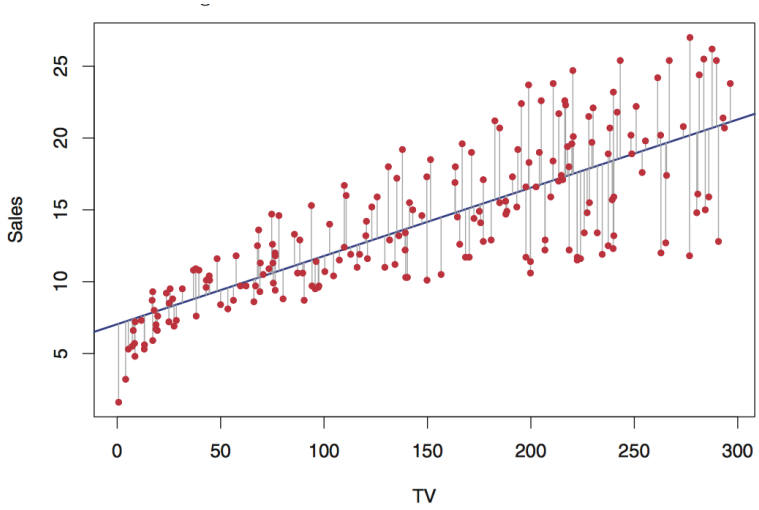


Figure 2.8. The fit is found by minimizing the sum of squared errors. Each gray line segment represents an error, and the prediction makes a compromise by averaging their square [**ISLR**]

The most common method for estimation is called least squares. This method obtains parameter estimates that minimize the sum of squared residuals (SSR):

$$SSR = \sum_{i=1}^{n} e_i^2 \tag{2.6}$$

The minimizers are

$$\widehat{\beta_1} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \tag{2.7}$$

$$\widehat{\beta_0} = \bar{y} - \widehat{\beta_1}\bar{x} \tag{2.8}$$

where $\bar{x}$ and $\bar{y}$ are the mean of x and y.

If we assume that the error term has constant variance, the estimate of the variance of the error is given by:

$$\widehat{\sigma}_{\varepsilon}^2 = \frac{SSR}{n-2} \tag{2.9}$$

And is called mean square error (MSE) of the regression. The denominator is the sample size reduced by the number of model parameters estimated from the same data, $(n-p)$ for p regressors or $(n-p-1)$ if an intercept is used [**MSE**]. In the case of simple linear regression $p = 1$, so the denominator is $n-2$.

We can then estimate the standard errors for the parameters, that tell us the average amount that the estimate differs from the actual value.

$$\widehat{\sigma}_{\beta_1} = \widehat{\sigma}_{\varepsilon}\sqrt{\frac{1}{\sum(x_i - \bar{x})^2}} \tag{2.10}$$

$$\widehat{\sigma}_{\beta_0} = \widehat{\sigma}_{\varepsilon}\sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum(x_i - \bar{x})^2}} \tag{2.11}$$

Standard errors can be used to compute confidence intervals. A 95% confidence interval is defined as a range of values such that with 95% probability, the range will contain the true unknown value of the parameter.

The general multiple regression model, where there are p independent variables, follows this formula:

$$y_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i \tag{2.12}$$

where x_{ij} is the ith observation on the jth independent variable.

2.2 Logistic Regression

In regression analysis, logistic regression is a method for estimating the parameters of a logistic model. A logistic model is a model where the log-odds of the probability of an event is a linear combination of independent or predictor variables.
The two possible dependent variable's values are often labeled as "0" and "1", "true" or "false", "pass" or "fail" etc, and they represent a binary outcome (fig. 2.9).
It's possible to generalize the binary logistic regression model to more than two levels (outcomes) of the dependent variable [**wiki:logisticreg**], called multinomial logistic regression.

The binary logistic model works by estimating the probability of a binary response based on one or more predictor variables.
By using logistic regression is possible to say how much the presence of a risk factor increases the odds of a given outcome.
The model itself simply calculates the probability of output in terms of input, and it is not specifically a classifier, but it can be used to classify data by choosing a threshold value, for example by saying that if the resulting probability is $\geq 50\%$ the class is "1" otherwise is "0".

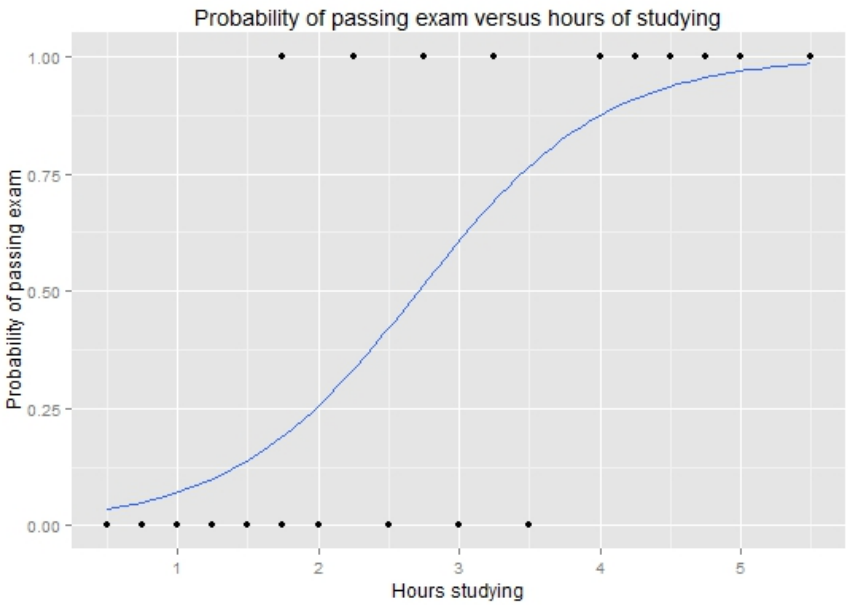


Figure 2.9. Graph of a logistic regression curve showing probability the of passing an exam versus hours studying [**wiki:logisticreg**].

Logistic Function

The logistic function to estimate the probabilities is a sigmoid function (fig. 2.10).
A sigmoid function takes an input from the real numbers and outputs a value between 0 and 1.

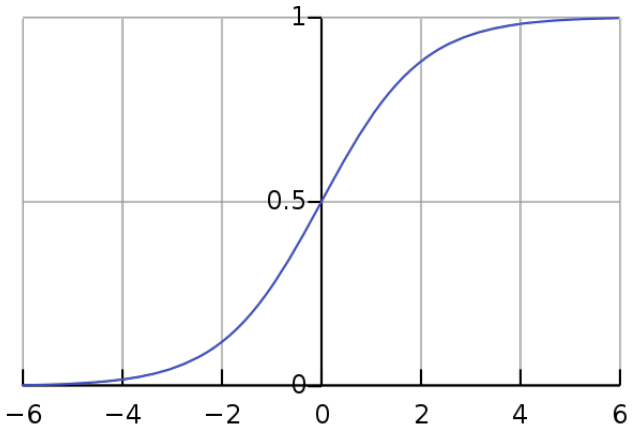


Figure 2.10. Example of a Sigmoid shaped function

We want to model the relationship between $p(X) = Pr(Y = 1|X)$ and X. To model this relationship we use the a logistic function:

$$p(X) = \frac{e^{\beta_0} + e^{\beta_1 X}}{1 + e^{\beta_0} + e^{\beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} \tag{2.13}$$

Equation 2.13 can be manipulated into:

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0} + e^{\beta_1 X} \tag{2.14}$$

$\frac{p(X)}{1-p(X)}$ is called odds, and can take any value between 0 and ∞.

By doing the logarithm of 2.14 we get:

$$\log \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X \tag{2.15}$$

The coefficients β_0 and β_1 in equation 2.13 are unknown, and must be estimated based on the available training data.
To fit the model of equation 2.15, the maximum likelihood method is commonly used.
Intuitively we want to estimates β_0 and β_1 such that the predicted probability $\hat{p}(x_i)$ for a specific case corresponds as closely as possible to the observed class for that case.
This can be achieved by using the likelihood function:

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1 - p(x_{i'})) \tag{2.16}$$

The estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to maximize this likelihood function.
Maximum likelihood is a very general approach that is used to fit many of the non-linear models. For example in linear regression (par. 2.1.5), the least squares approach is a special case of maximum likelihood.

Once the coefficient have been estimated is pretty simple to compute the probability:

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0} + e^{\hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0} + e^{\hat{\beta}_1 X}} \tag{2.17}$$

2.3 Random Forest

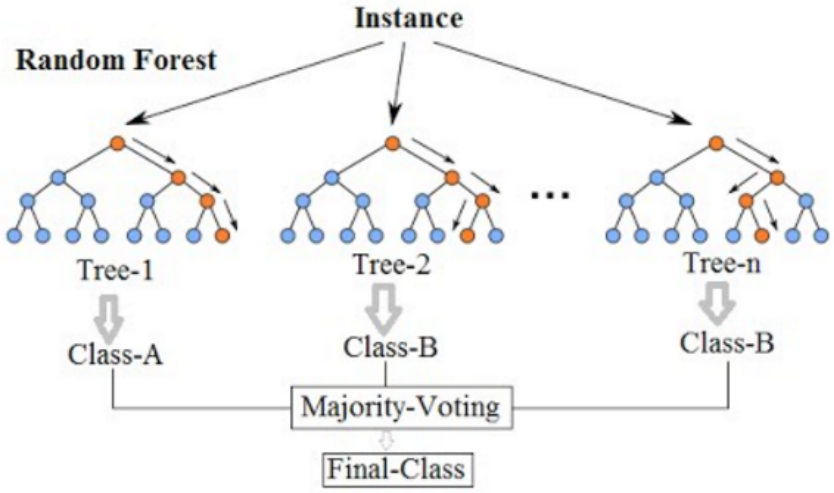


Figure 2.11. Example of Random Forest Classification [**medium:RF**].

Random forests are an ensemble learning method used mainly for classification and regression. They work by building many decision trees while in the training phase, and then outputting the class that is the mode of the classes for classification, or the mean prediction of the individual trees for regression [**wiki:randomforest**][**RDF**].

Random Forest (RF) is preferred to decision trees because decision trees suffer from high variance. By using tree bagging, which is a commonly used technique to reduce variance, it is possible to correct decision trees' habit of overfitting to their training set [**ESL**].

In the following paragraphs we will give an overview of how the Random Forest method is derived from decision trees and bagging.

Decision Trees

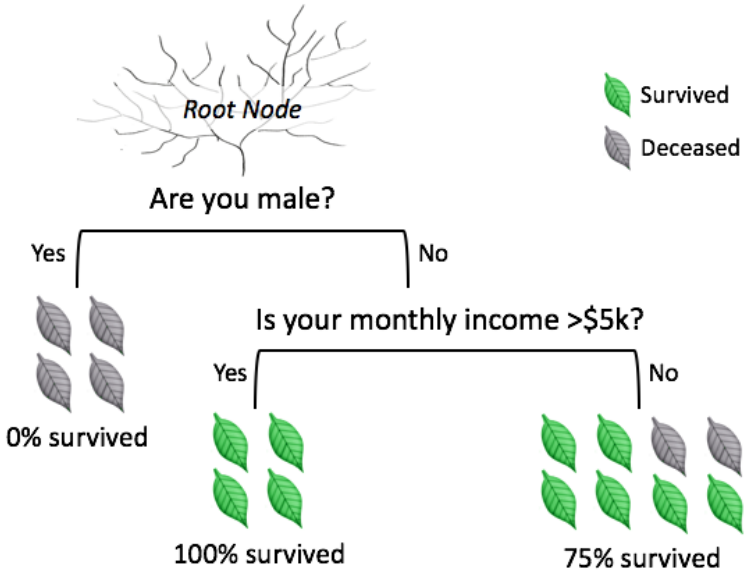


Figure 2.12. Example of a Decision Tree [**KDNrf**].

Decision trees are a widespread method for many different machine learning tasks and can be used for either classification or regression.
The aim of decision trees is to build a model that predicts the value of a target variable based on different input variables.

Each node of the tree corresponds to one of the input variables. Each leaf represents a value of the target variable, given the values of the input variables represented by the path from the root to the leaf (fig 2.12) [**wiki:DT**].

The data usually has the following structure:

$$(x, Y) = (x_1, x_2, \ldots, x_n, Y) \tag{2.18}$$

Where Y is the dependent variable that we are trying to classify and x is the feature vector.

A decision tree is able to represent a classification problem: the arcs of a tree's node labeled with input features are labeled with the possible values for that feature, and they can lead to two options:

- A subordinate decision node, with different input features.
- A leaf labeled with the resulting class (or a probability).

A tree can be learned by dividing the training set into subsets based on the values of the attributes. This is done recursively until a node has all the same value of the target variable, or when splitting is no longer aiding the prediction's result.

Decision trees, however, are often inaccurate. Specifically, by growing trees very deep, they tend to learn highly irregular patterns by overfitting to the train set, and

result having low bias but very high variance) [**ESL**].

Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.
This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model [**ESL**].

Tree Bagging

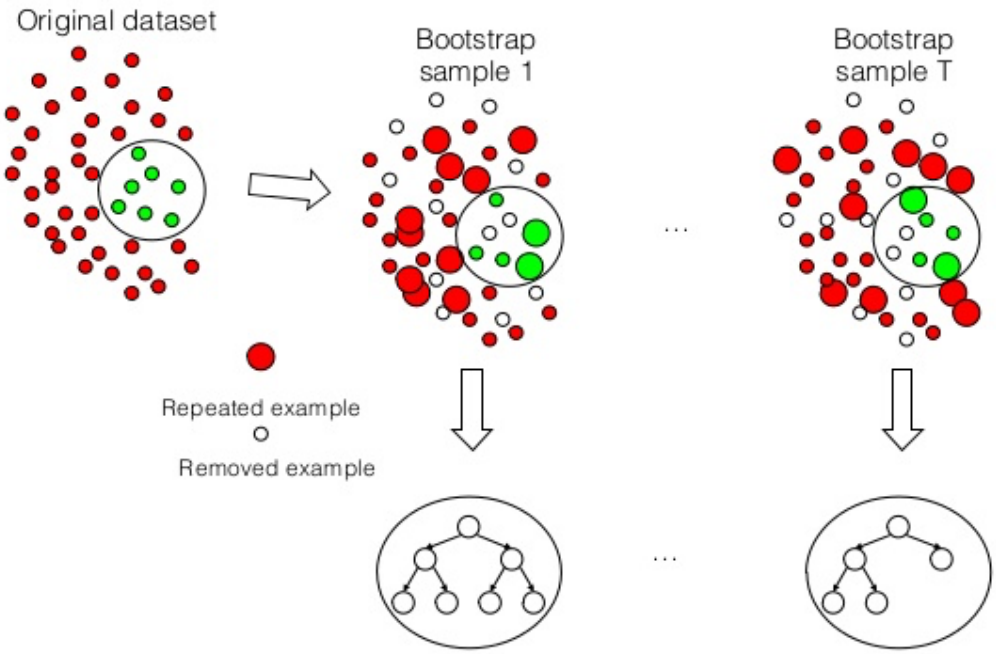


Figure 2.13. Tree Bagging algorithm example [**bagging**].

Given a set of n independent observations $X_1, \ldots, X_n$, each with variance σ^2, the variance of the mean $\overline{X}$ of the observations is given by σ^2/n.
This means that averaging a set of observations reduces the variance. So to reduce the variance we can take many training sets from the population, build a prediction model using each training set, and average the resulting predictions.

Basically we could calculate $\hat{f}_1(x), \hat{f}_2(x), \ldots, \hat{f}_B(x)$ using B separate training sets, and then average them in order to obtain a single model with lower variance [**ISLR**].

This solution is not practical since we normally lack more than one training set.
Instead we utilize *bagging*: we can take repeated samples from the training set in order to generate B different bootstrapped versions of it 2.14. We then train our method on the b*th* bootstrapped training set, and then average the resulting predictions (fig. 2.13).

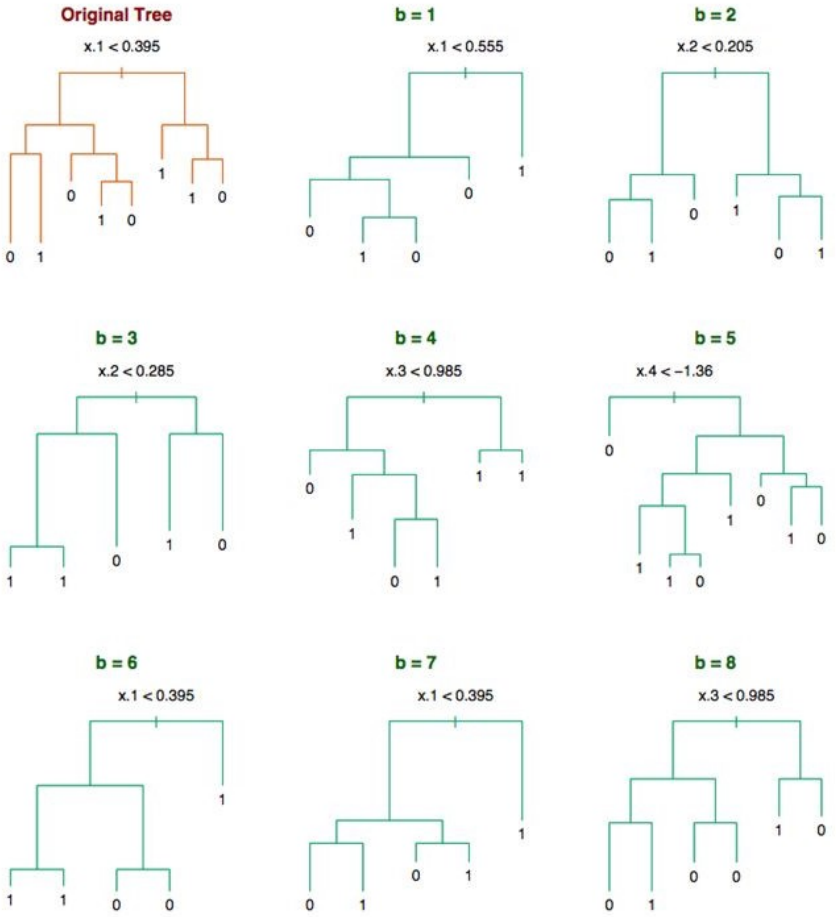


Figure 2.14. Bagging leads to different decision trees' structure [**baggingDT**].

To apply bagging to regression trees specifically, we simply construct B regression trees using B bootstrapped training sets, and average the resulting predictions. These trees are grown deep, and are not pruned. Hence each individual tree has high variance, but low bias. Averaging these B trees reduces the variance [**ISLR**].

If we are using Random Forests for classification (like in this thesis), the simplest approach is the following:
Given a test observation, we record the class predicted by each of the B trees, and take a majority vote: the final prediction is the most frequent class among the B predictions.

More formally, given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1, \ldots, y_n$, bagging B times selects a random sample with replacement of the training set and fits trees to these samples [**wiki:randomforest**]:

For $b = 1, \ldots, B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x') \tag{2.19}$$

or by using the majority vote in case of classification.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}} \tag{2.20}$$

The number of samples/trees B is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error (par 2.3): the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample.
The training and test error tend to settle after enough trees have been fit [**ISLR**].

Random Forests

Random Forests provide an improvement over bagged decision trees by decorrelating the trees. As in bagging, the decision trees are built on a bootstrapped training set. The difference is that every time a split is considered, a random sample of m predictors is chosen as split candidates from the total p predictors. The split can only use one of those m predictors. $\sqrt{m}$ predictors are taken each split, typically with $m \equiv \sqrt{p}$ [**ISLR**].

Practically, at each split in the tree, the algorithm can only consider a subset of the predictors (fig. 2.15). This works because if there is a very strong predictor, that predictor would be used in the majority of the trees, and then all the bagged trees would look alike and would be highly correlated, thus leading to a small reduction in variance.

By forcing each split to consider only m predictors, on average $(p-m)/p$ of splits will not take into account a strong predictor. This, in turn, will decorrelate the trees.

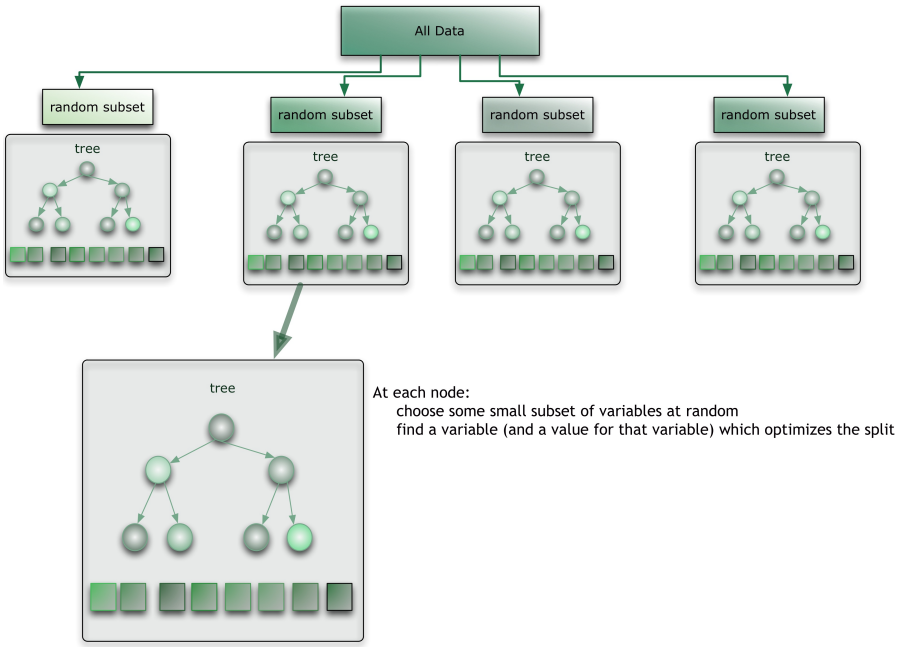


Figure 2.15. Random Forest overview.

Out of bag Error

There is a very direct way to estimate the test error of a bagged model:
on average, we use 2/3 of the observations to fit a bagged tree. The remaining 1/3 that are not utilized are called the out-of-bag (OOB) observations.

We can predict the response for the i*th* observation using each of the trees in which that observation was OOB. This will generate approximately $B/3$ predictions for the i*th* observation. We can then average these predictions, or use majority voting for classification.

We can repeat the process for all the n observations and get an average classification error.
The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation [**ISLR**].

Variable Importance

Bagging has the advantage of giving increased accuracy over using a single tree, at the cost of interpretability of the data. Using bagging we can no longer have easily interpreted diagrams [**ISLR**].

With Random Forest we can instead rank the importance of each predictor:
given a dataset $D_n = \{X_i, Y_i\}_{i=1}^n$ we fit a RF to the data. While fitting the data, the out-of-bag error is averaged over the forest.

To get the importance of the j*th* feature, we permute it's values over the training

data and recompute the OOB error.

The importance is given by doing the average of the difference between the OOB error before and after this permutation. Then the score is normalized by the standard deviation of the differences.

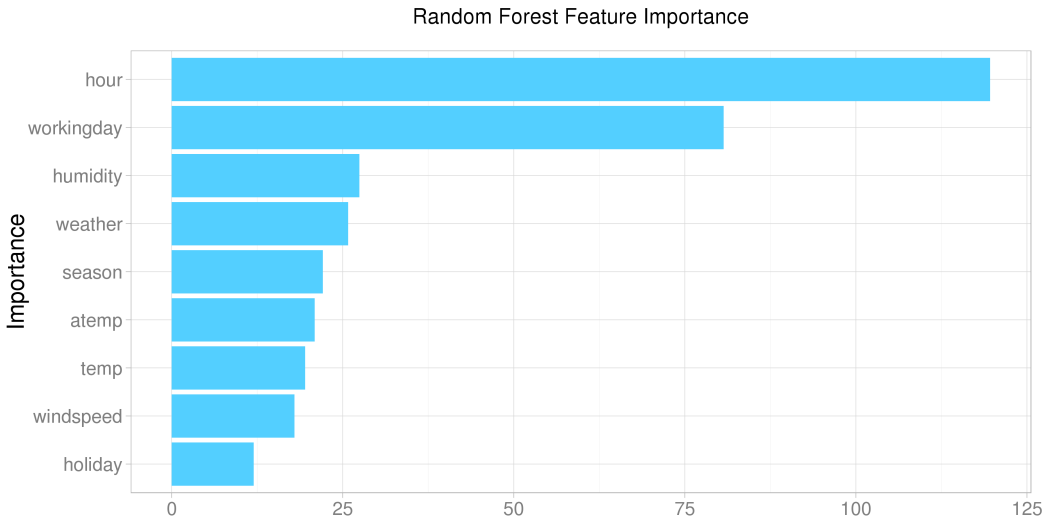


Figure 2.16. Example of variable importance found using Random Forest [**rf_imp**].

2.4 SVM

Support Vector Machines (SVM) are a supervised machine learning algorithm used for both classification and regression.

The main idea is to find the optimal hyperplane for linearly separable data, and then extend this idea to data that are not linearly separable by mapping this data in a new space using a kernel function.

2.4.1 Hyperplane

What is a hyperplane? In p-dimension, a hyperplane is a flat affine subspace of $p-1$ dimension. For example, in two dimensions it's a line, in three dimensions it's a plane (fig. 2.17), and so on.

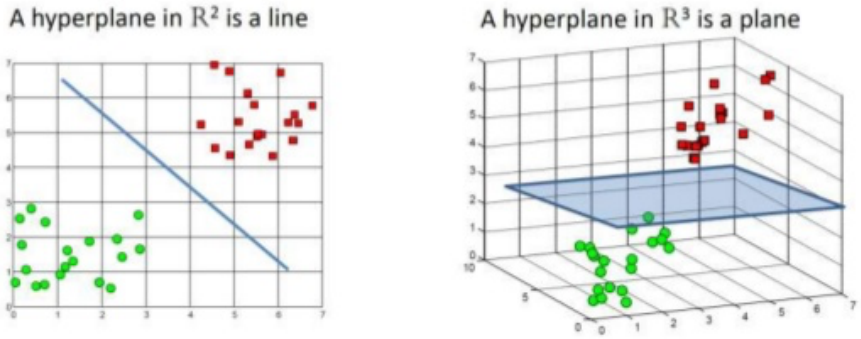


Figure 2.17. Two and three dimensional space hyperplanes [**hyperplaneimg**].

The definition of an hyperplane for two dimensions is:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \tag{2.21}$$

Where any $X = (X_1, X_2)^T$ for which equation 2.21 holds is a point on the hyperplane.

This definition can be extended to an hyperplane of p-dimensions:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{2.22}$$

If X does not satisfy equation 2.22, but

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0 \mid < 0 \tag{2.23}$$

we know that X lies to one side or the other of the hyperplane. So a hyperplane divides a space in two halves [**ISLR**].

It's possible to build a hyperplane that can separate training data based on their class labels, for example by assigning $\{-1, 1\}$ if the result lies on one side or the other of the hyperplane.

2.4.2 Maximal Margin Classifier

If the data can be separated by a hyperplane, it means that there can be an infinite number of separating hyperplanes.
To build a classifier it's necessary to chose one between those infinite hyperplanes, such that the classification results are as accurate as possible.

The distance between the hyperplane and the nearest data point from either side is known as the margin.
Generally the choice is the *maximal margin hyperplane*, meaning the hyperplane that *maximizes* the margins, meaning that it has the farthest minimum distance, for the data we are trying to classify.

In fact the distance of a data point from the hyperplane is a measure of our confidence that the observation was classified correctly [**ISLR**].

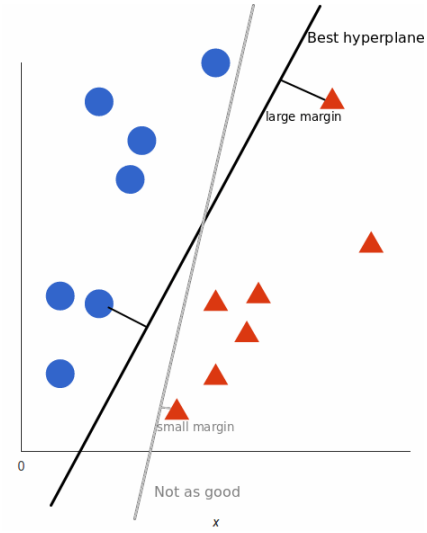


Figure 2.18. Different hyperplanes have different margins [**svm_monkeylearn**].

The idea is that a classifier with a large margin on the training set will have a large margin on the test set, so it will classify it correctly [**ISLR**].

2.4.3 Support Vectors

Support vectors are the data points that lie closest to the hyperplane (fig. 2.19), they are also the most difficult data points to classify and have direct influence on the location of the hyperplane. In fact, moving the support vectors would change the hyperplane's location.

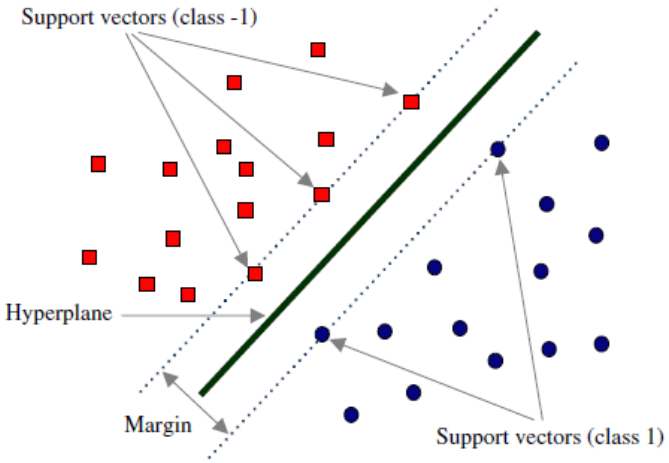


Figure 2.19. Example of Support Vectors.

So the objective is to choose a hyperplane with the largest possible margin between it and the support vectors, since the larger is the margin, the lower the generalization error of the classifier [**ISLR**].

2.4.4 Finding the Maximal Margin Classifier

Finding the maximal margin hyperplane based on a set of n training observations $x_1, \dots x_n \in \mathbb{R}^p$ and with class labels $y_1 \dots y_n \in \{1, -1\}$, translates to an optimization problem:

Maximize the margin M:

$$\beta_0, \beta_1, \beta_2, \dots, \beta_p, M \tag{2.24}$$

subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{2.25}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n \tag{2.26}$$

Equations 2.25 and 2.26 ensure that each observation is on the correct side of the hyperplane, and at least at distance M from the hyperplane.
So M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, \beta_2, \beta_3, \dots, \beta_p$ to maximize M and find the maximal margin hyperplane [**ISLR**].

2.4.5 Support Vector Classifier

Unfortunately this hyperplane does not necessarily exists (fig. 2.20), but we can extend the concept of the Maximal Margin Classifier to find a hyperplane that almost separates the classes by using a *soft* margin. This is actually what an SVM does.

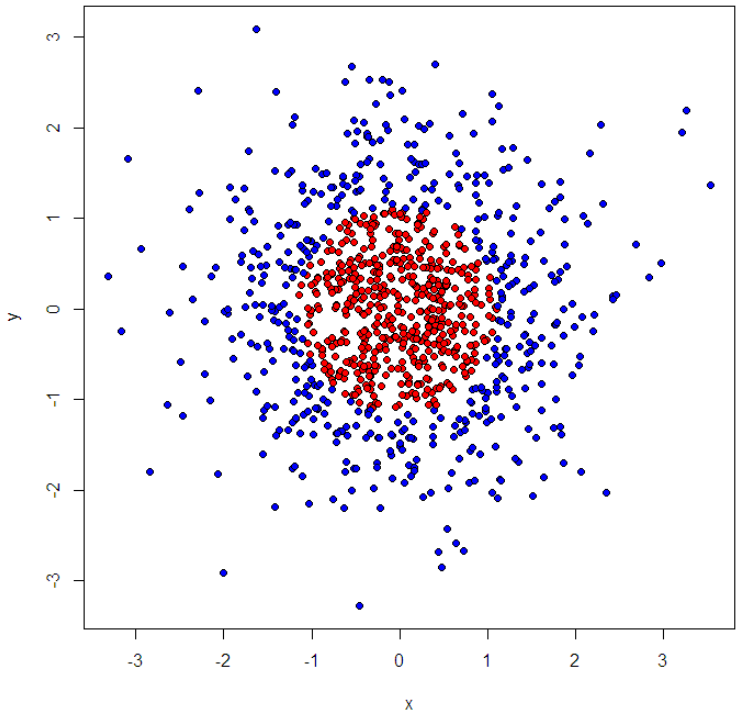


Figure 2.20. Example of non linearly separable data [**svm_not_sep**].

Even if a separating hyperplane did exists, we might not want to use it: such a classifier would necessarily classify all of the training data perfectly, and that would lead to sensitivity to changes in data points (fig. 2.21). Also, being so sensitive could mean that it's overfitting to the data.

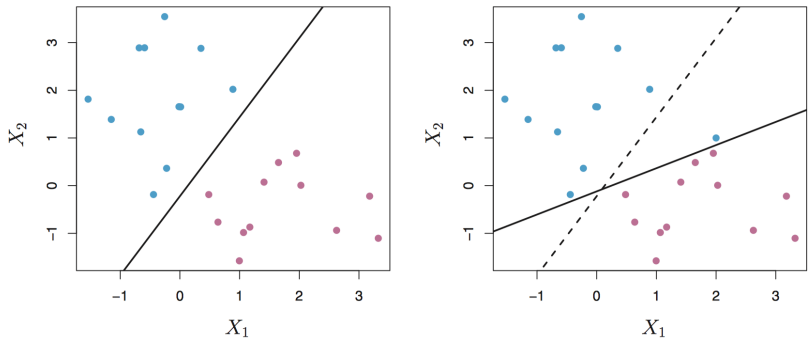


Figure 2.21. Adding a single data point (from left to right) can change the maximal margin hyperplane very significantly [**ISLR**].

Instead we consider a classifier that does not separate the data perfectly, but is less prone to change for individual data points and can better classify *most* of the observations.

This classifier is called *soft margin classifier* or *support vector classifier* (fig. 2.22). In a support vector classifier (SVC) a small subset of the observations are on the wrong side of the margin, or even on the wrong side of the separating hyperplane.

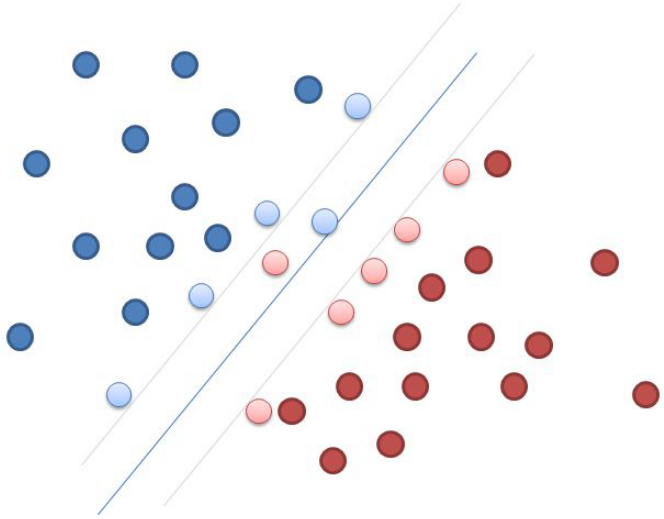


Figure 2.22. Soft margin classifier. Some data points in the picture are allowed to be misclassified, this leads to finding a larger and better margin [**soft_svm**].

The optimization problem we are solving now is the following:

Maximize the margin M:

$$\beta_0, \beta_1, \beta_2, \ldots, \beta_p, \epsilon_1, \ldots, \epsilon_n, M \tag{2.27}$$

subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{2.28}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \quad \epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C \tag{2.29}$$

With $C \geq 0$, and $\epsilon_1, \ldots, \epsilon_n$ being slack variables to allow data points to be on the wrong side of the margin.

When we solve equations 2.27-2.29 we can classify x^* by determining where it lies on the hyperplane:

$$f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + \cdots + \beta_p x_p^* \tag{2.30}$$

Based on the value of ϵ_i we can make some assumptions:

- If $\epsilon_i = 0$ the ith observation is on the correct side of the margin.

- If $\epsilon_i > 0$ the ith observation is on the wrong side of the margin.
- If $\epsilon_i > 1$ the ith observation is on the wrong side of the hyperplane.

C determines the quantity and quality of the violations of the margin and hyperplane that is tolerable by our classifier, and is called *cost*.

- If $C = 0$ then we cannot violate the margin, and $\epsilon_1 = \cdots = \epsilon_n = 0$.
- If $C > 0$ then the observations $\leq C$ can be on the wrong side of the hyperplane.

When C is small the margin is small and we have a classifier that is very fit to the data, with low bias and high variance.
When C is big the margin is larger and the classifier is less fit to the data and has high bias and low variance.

The interesting property of this optimization problem is that only the support vectors (observations that are on the margin or that violate the margin) affect the hyperplane, and thus the classifier obtained.
All the observations that lie on the correct side of the margin do not affect the support vector classifier.

2.4.6 Kernels

Sometimes data is not linearly separable (fig. 2.20). When that happens, Support Vector Machines work by enlarging the feature space using specific function called kernels, utilized to account for a non-linear boundary between the classes.

The solution to equations 2.27 - 2.29 involves only the inner products of the observations, and not the observations themselves [**ISLR**].
We define the inner product between two r-vectors a and b as $\langle a, b\rangle = \sum_{i=1}^{r} a_i b_i$. So the inner product of two observations x_i, x_i' is given by:

$$\langle x_i, x_i'\rangle = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{2.31}$$

the linear SVC can then be described as:

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i\rangle \tag{2.32}$$

To estimate β_0 and $\alpha_1 \ldots, \alpha_n$ all that is required are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'}\rangle$ between all pairs of training observations [**ISLR**].

Every time we compute the inner product (eq. 2.31) in equation 2.32 we could replace it with a generalization of this form:

$$K(x_i, x_i') \tag{2.33}$$

Where K is a kernel function. A kernel function is used to quantify the similarity between two observations.

For example a linear kernel of p features is:

$$K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{i'j} \tag{2.34}$$

or for a polynomial kernel of degree d:

$$K(x_i, x_i') = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d \tag{2.35}$$

the function for a radial kernel is:

$$K(x_i, x_i') = \exp(-\gamma \sum_{j=1}^{p} (x_{ij}, x_{i'j})^2), \quad \gamma \geq 0 \tag{2.36}$$

yet another example is the sigmoid kernel:

$$K(x_i, x_i') = \tanh(-k \sum_{j=1}^{p} (x_{ij}, x_{i'j})^2 + \delta) \tag{2.37}$$

We can visualize some of these kernels in fig. 2.23:

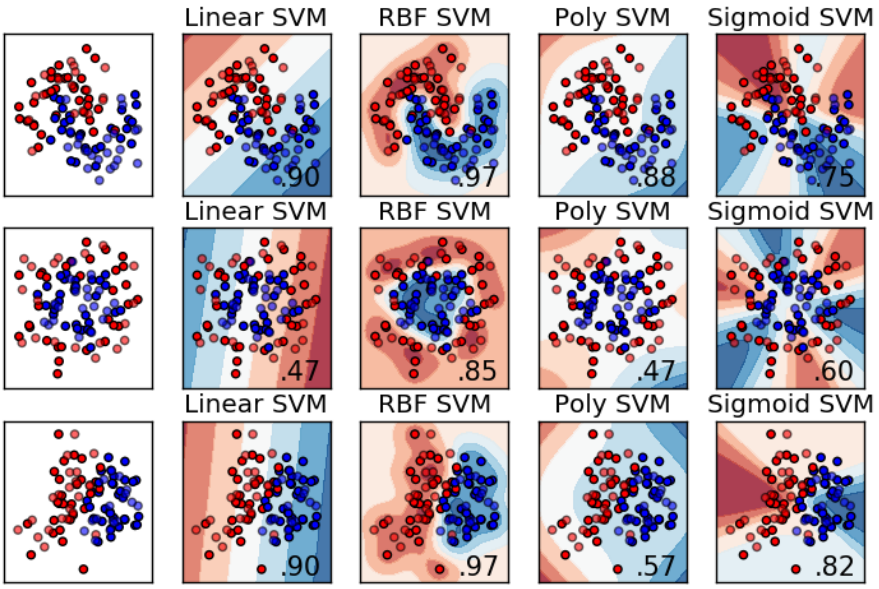


Figure 2.23. Visualization of different kinds of kernels [**kernels**].

There are many more kernel functions that can be used based on the shape of the data.

The advantage of using a kernel is first of all computational: by using a kernel we just need to compute $K(x_i, x_i')$ for all $\binom{n}{2}$ distinct pairs (i, i').

Graphically, starting with a two dimensional non linearly separable data, this is what happens:

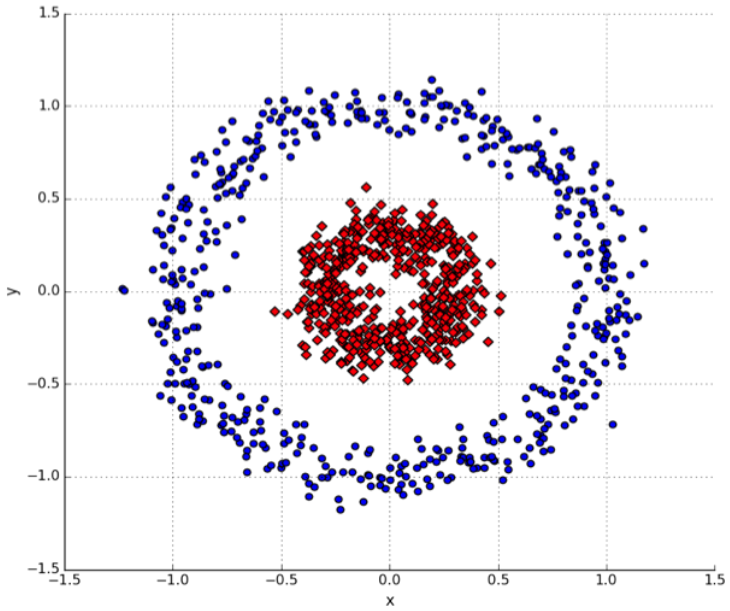


Figure 2.24. Linearly non separable data [**kernel_trick**].

We map the data from input space X into a transformed feature space H, by using a non-linear function $\phi : X \to H$

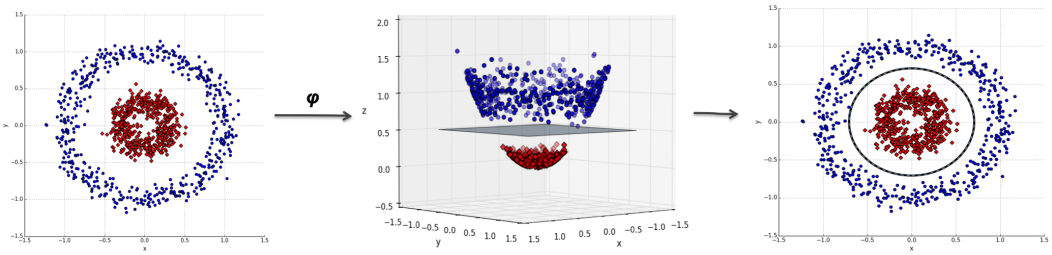


Figure 2.25. Transformation of the data in a feature space where the instances from the two classes may be linearly separable [**kernel_trick**].

Chapter 3

Architecture

In this chapter we will discuss the architecture of our framework and the tools used for this thesis in Deception Detection. We will first explain the general architecture to perform deception detection (par. 3.1), then we will introduce the OpenFace toolkit (par. 3.2), and describe landmark detection (par. 3.3), action unit recognition (par. 3.4) and the deception detection process (par. 3.5).

3.1 General Architecture

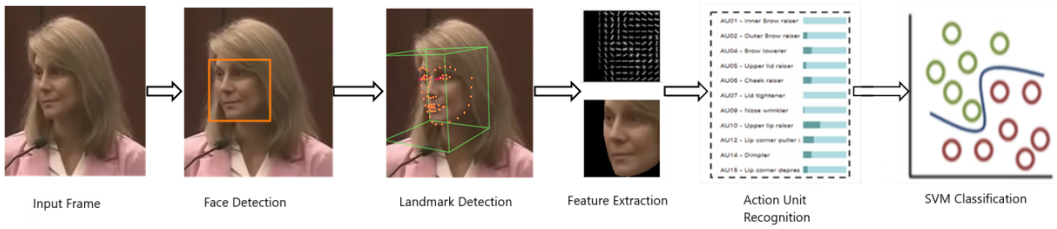


Figure 3.1. System Architecture.

The general idea of this thesis is to classify whether a person is telling the truth or lying by analyzing the subject's facial movement, extracted using an RGB camera, during a video. To achieve this it is fundamental that the subject's face results visible during the videos.

We ultimately implemented this idea using a Support Vector Machine classifier.

The architecture utilized is shown in fig 3.1 and is composed this way:

- We gather a video of a person in a high stake situation and split it into frames;
- Detect the face of the person in the frame;
- Detect the facial landmarks using Convolutional Experts Constrained Local Model (CE-CLM);

- Extract the appearance features in the form of Histogram of Oriented Gradients and perform dimensionality reduction and feature normalization through Principal Component Analysis;
- Extract action unit presence and intensity from the frame using a Support Vector Classifier;
- Feed the extracted data to train a Support Vector Classifier for classification on a test set.

3.2 OpenFace

The OpenFace [**Baltru2018**] toolkit is a collection of libraries and models for machine learning and computer vision researchers, created by Baltrusaitis et al., to support performing facial landmark detection, head pose estimation, action unit recognition, feature extraction and eye gaze estimation.

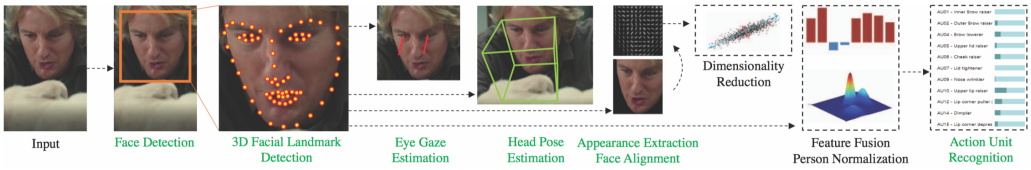


Figure 3.2. OpenFace pipeline [**Baltru2018**].

This tool is fully customizable in such a way that all the support vector machine and neural networks models can be trained with different datasets. Moreover the code is open source so it is possible to change any part of it to fit the needs of the experimenters.

In the next paragraphs we will describe how we used this tool to perform landmark detection (par. 3.3) and action unit detection (par. 3.4).

3.3 Landmark Detection

The first step in Action Unit identification is to detect the facial landmarks. To accomplish this a local detector called Convolutional Experts Network (CEN) (Fig. 3.3) is utilized.

CEN has the advantage of aggregating a neural architecture and patch experts, which are local detectors, which are used to evaluate the probability of a landmark being aligned at a particular pixel location.

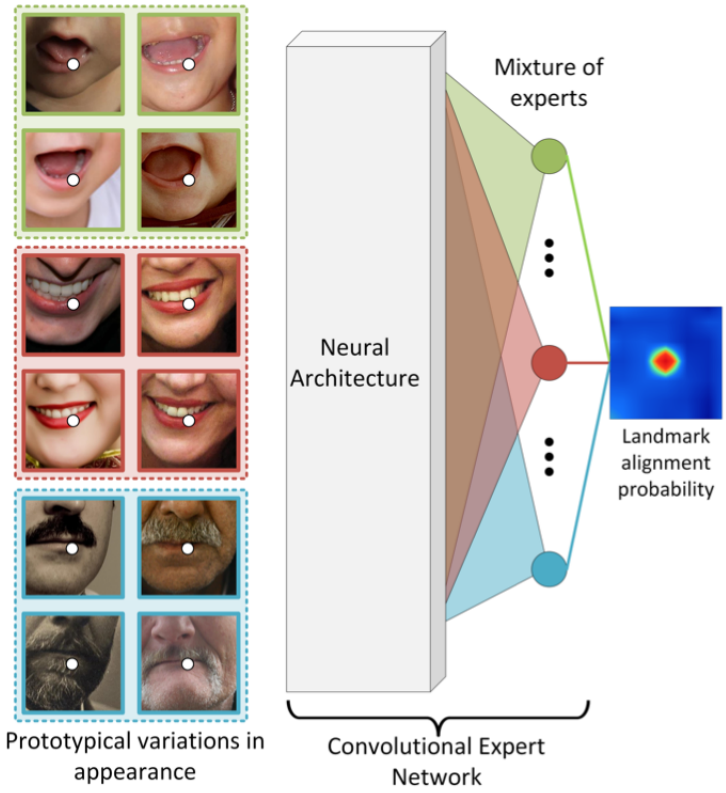


Figure 3.3. Facial landmarks generally cluster around facial hair, expressions etc. To model these variations a Convolutional Experts Network is used in order to get a landmark alignment probability [**Baltru2017**].

OpenFace uses a Convolutional Experts Constrained Local Model (CE-CLM) [**Baltru2017**], which is a Constrained Local Model (CLM) that uses Convolutional Experts Network as a local detector.

A Constrained Local Model is class of methods for locating sets of points, constrained by a statistical shape model, on a target image [**clm_cootes**].

Generally the procedure is as follows (fig. 3.4):

- Sample a region from the image around the current estimate, projecting it into a reference frame;

- For each point, generate a "response image" giving a cost for having the point at each pixel;
- Search for a combination of points which optimizes the total cost, by manipulating the shape model parameters.

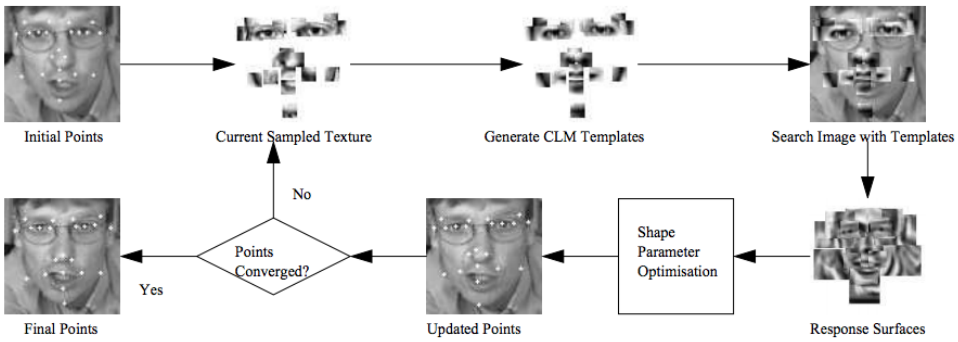


Figure 3.4. Overview of Constrained Local Model [**clm_cootes**].

CLM are used to model the appearance of each facial landmark individually by using local detectors and a shape model for constrained optimization.
The CE-CLM is divided in two fundamental parts: response map computation using CEN, and shape parameter update.
In the first step, the landmarks alignment is computed separately from the other landmarks.
In the second phase, all landmarks are considered together and for misaligned landmarks and irregular shapes their position is penalized, using a Point Distribution Model (PDM).

3.3.1 Convolutional Experts Network

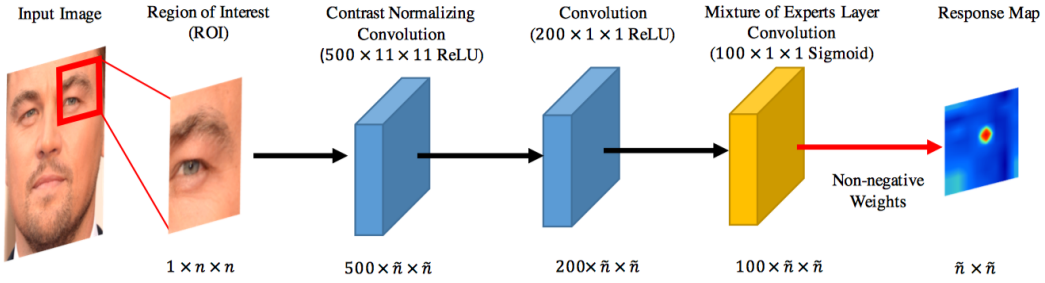


Figure 3.5. Overview of the Convolutional Experts Network model. [**Baltru2017**].

In the initial step the objective is to generate a response map to localize the individual landmarks. This is performed by assessing the landmark alignment probability at specific pixel locations.
CEN takes in input a Region of Interest (ROI) around the currently estimated

position of a landmark, and outputs a response map that calculates the landmark alignment probability at each pixel location (fig. 3.5).

In order to do all this, the ROI is initially passed in a Contrast Normalizing Convolution layer to perform z-score normalization and to calculate the correlation between input and kernel. The output is then convolved in another layer of ReLU neurons (convolution is an operation on two functions to produce a third function that expresses how the shape of one is modified by the other).

The last neural layer before the response map is the Mixture of Expert Layer (ME-layer), which models the alignment probability using a combination of patch experts (local detectors) that represent different landmarks appearance prototypes by outputting individual votes on alignment through a sigmoid function. The response maps from all the local detectors are then combined in the last layer, giving the final alignment probability.

3.3.2 Point Distribution Model

The Point Distribution Model is commonly utilized in computer vision as a standard to represent the mean geometry of a shape, and to gather information from geometric variation inferred from a training set of shapes [**wiki:PDM**].

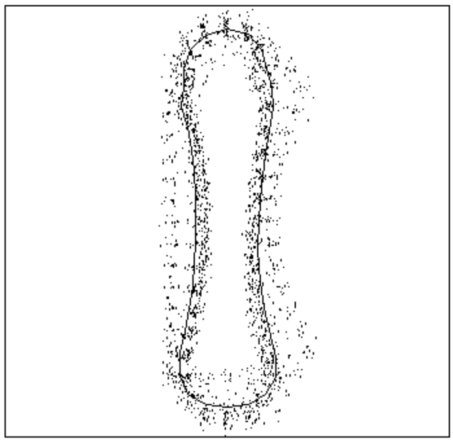


Figure 3.6. Point Distribution Model of a metacarpal. Dots mark the possible position of landmarks, and the line denote the mean shape [**PDM**].

Point distribution models rely on landmark points. The general PDM works this way:

1. A set of training images are manually landmarked to sufficiently approximate the geometry of the original shapes;
2. These k landmarks are aligned in two dimensions resulting in $\mathbf{X} = (x_1, y_1, \ldots, x_k, y_k)$
3. The shape outlines are reduced to sequences of k landmarks, so that any given training shape can be defined by the vector $\mathbf{X} \in \mathbb{R}^{2k}$;

4. The matrix of the top d eigenvectors is given as $\mathbf{P} \in \mathbb{R}^{2k \times d}$, and each eigenvector describes a principal mode of variation along the set;

5. A linear combination of the eigenvectors is used to define a new shape $\mathbf{X}'$, mathematically defined as:

$$\mathbf{X}' = \overline{\mathbf{X}} + \mathbf{Pb} \tag{3.1}$$

   where $\overline{\mathbf{X}}$ is defined as the mean shape across all training images, and $\mathbf{b}$ is a vector of scaling values for each principal component;

6. By modifying the variable $\mathbf{b}$ an infinite number of shapes can be defined. $\mathbf{b}$ shouldn't generally be modified more than $\pm 3\sigma$ [**wiki:PDM**].

With the OpenFace framework Point Distribution Model [**PDM_RLMS**] we model the location of facial feature points in the image, using non-rigid shape and rigid global transformation parameters.

The application of PDM has two objectives:

- They are employed to control the landmark locations.
- They are used to regularize the shapes in the CE-CLM framework by using Non-Uniform Regularized Landmark Mean Shift (NU-RLMS) [**Baltru2013**].

This has an effect in the final detected landmarks, where the irregular shapes are imposed a penalty.

The results can be seen in the figure 3.7 below for different images with different illuminations and angles.

Figure 3.7. Example of different facial landmarks detected in diverse conditions and viewing angles [**Baltru2018**].

3.4 Action Unit Detection

Action Unit (AU) detection plays a fundamental role in our work. We now describe the process used to detect the AUs through an SVM, starting with an overview of the datasets utilized for training the SVM for AUs detection (par. 3.4.1) and then proceding to explain feature extraction (par. 3.4.2).

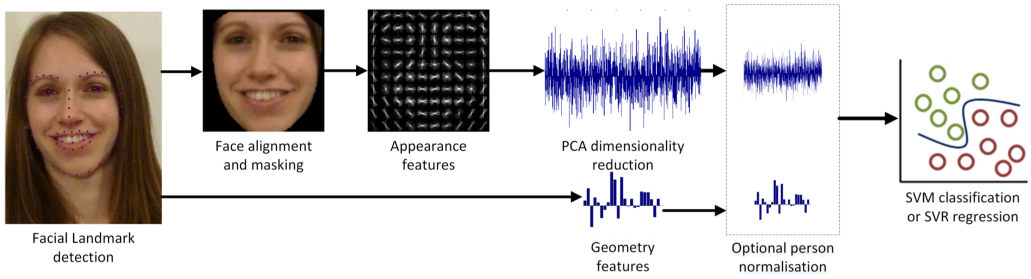


Figure 3.8. AU detection and intensity pipeline [**Baltru2015**].

3.4.1 Training Datasets

There are four main dataset used for training the Action Unit detection system: DISFA [**DISFA**], BP4D-Spontaneous [**BP4D-Spontaneous**], SEMAINE [**SEMAINE**] and CK+ [**CK+**]. These four datasets consist of videos of people subject to emotion inducing tasks.

The **BP4D** database of spontaneous facial expressions includes videos of 41 participants (23 women and 18 men, 21 for training and 20 for validating). The age ranged from 18 to 29; 11 were Asian, 6 African-American, 4 Hispanic, and 20 Euro-American.
Emotion inducing techniques were used to elicit an emotional response. Frame-level ground-truth for facial actions was obtained by using the Facial Action Coding System annotations, performed by trained professionals.
Each participant in the database is associated with 8 tasks. For each task, there are both 3D and 2D videos and the metadata include annotations for 11 AUs for occurrence and 5 AUs for intensities.

DISFA (Denver Intensity of Spontaneuos Facial Action) Database is a non posed facial expression database for automatic action unit detection. It contains videos of 27 participants (12 female, 15 males; 14 used for training and 13 for validation). It includes 4 minute-long videos of spontaneous facial expression, resulting in 130k frames annotated for 12 AUs (fig. 3.9), comprehensive of AUs intensity on a 0 to 5 scale.

AU#	FACS Name
1	Inner Brow Raiser
2	Outer Brow Raiser
4	Brow Lowerer
5	Upper Lid Raiser
6	Cheek Raiser
9	Nose Wrinkler
12	Lip Corner Puller
15	Lip Corner Depressor
17	Chin Raiser & Mentalis
20	Lip Stretcher
25	Lip Part
26	Jaw Drop

Figure 3.9. AUs coded in the DISFA database [**DISFA_AU**].

The FACS annotated **SEMAINE** subset contains recordings of 31 subjects (15 for training and 16 for validation). It consists of one minute long recordings, leading to 93k frames labeled for 5 AU occurrences.

The **Cohn-Kanade** AU-Coded Facial Expression Database is utilized for research in facial image analysis.
CK+, includes both posed and non-posed (spontaneous) expressions and additional types of metadata. Each posed expression is completely FACS coded, and emotion labels are part of the metadata.
CK+ also provides baseline results for facial feature tracking, action unit and emotion recognition.

BP4D, SEMAINE and DISFA have three AUs in common (2, 12, and 17).
SEMAINE and DISFA share AUs 2, 12, 17, 25.
BP4D and DISFA share AUs 1, 2, 4, 6, 12, 15, 17.
This allows for cross-dataset training.

By training on these dataset is possible to detect the following AUs (fig. 3.10):

AU	Full name	Illustration
AU1	INNER BROW RAISER	
AU2	OUTER BROW RAISER	
AU4	BROW LOWERER	
AU5	UPPER LID RAISER	
AU6	CHEEK RAISER	
AU7	LID TIGHTENER	
AU9	NOSE WRINKLER	
AU10	UPPER LIP RAISER	
AU12	LIP CORNER PULLER	
AU14	DIMPLER	
AU15	LIP CORNER DEPRESSOR	
AU17	CHIN RAISER	
AU20	LIP STRETCHED	
AU23	LIP TIGHTENER	
AU25	LIPS PART	
AU26	JAW DROP	
AU28	LIP SUCK	
AU45	BLINK	

Figure 3.10. List of available AUs for prediction [**Baltru2018**].

3.4.2 Feature Extraction

There are two types of features that are used: appearance features through Histogram of Oriented Gradients, and geometry using shape parameters and landmark locations. To extract those features it's necessary to track certain landmarks on the face, and then continue this process by performing face alignment.

Face Tracking

Face tacking is done by utilizing Constrained Local Neural Field (CLNF) facial landmark detector and tracker, backed up by a structural SVM for facial detection [**Baltru2013**].

CLNF is a specific case of Constrained Local Model (CLM), that differs from the original by utilizing more advanced local detectors and a different optimization function.

The Constrained Local Model can be described by the following parameters:
$p = [s, \mathbf{R}, \mathbf{p}, \mathbf{t}]$.

- s is the scale factor.
- $\mathbf{R}$ is the object rotation.
- $\mathbf{t}$ is the 2D translation.
- $\mathbf{p}$ describes the shape of a vector of non rigid variations.

These parameters can be modified to compute different versions of the model. The resulting point distribution model is:

$$x_i = s \cdot \mathbf{R}(\overline{x_i} + \phi_i \mathbf{p}) + \mathbf{t} \tag{3.2}$$

Where x_i is the location of the ith feature point in an image, $\overline{x_i}$ is the mean of the ith element of the PDM, and ϕ_i is the ith eigenvector that describes the variation of the feature point.

Alignment and Masking

For the extracted face to be correctly analyzed, there needs to be a mapping to a common reference frame, and the changes resulting from scaling and in plane rotation needs to be removed.

In order to achieve this, a similarity transform from the currently detected landmarks to a representation of frontal landmarks from a neutral expression is utilized.
The similarity transform is done with Procrustes superimposition that minimizes the mean square error between aligned pixels [**Baltru2013**].
The result is a 112×112 pixel image of the face with 45 pixel interpupillary distance (fig. 3.11).

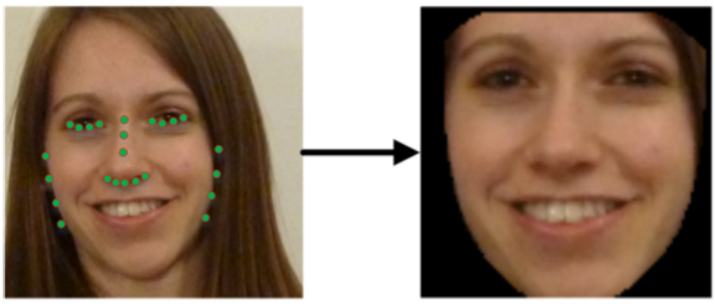


Figure 3.11. Stable points used for alignment to a common reference frame, followed by masking [**Baltru2015**].

To reduce the weight of significant facial expressions (mouth opening, cheeks puffing, eyebrow moving etc.) on the similarity transform, only the most stable facial landmarks must be used.
In order to determine those points, the most stable CLNF detected landmarks on the CK+ dataset [**CK+**] are examined.

Masking, performed using a convex hull surrounding the feature points, is done to remove information from the image that do not regard the face.

Appearance Features

After the face is aligned to a 112×112 image it's time to extract the appearance features.
To get the appearance features, 12×12 block of 31 dimensional Histogram of Oriented Gradients (HOG) are extracted, giving a 4464 dimensional vector characterizing the face. The implementation for HoG comes from dlib [**dlib**].

Once the feature vector is obtained, the next step is to reduce it's dimensionality. In order to do that Principal Component Analysis (PCA) is applied.
To generalize the dimensionality reduction, the training was performed on the FERA 2015 [**FERA15**], CK+ [**CK+**], DISFA [**DISFA**], AVEC 2011 [**AVEC11**] and FERA 2011 [**FERA11**] datasets.
By performing Principal Component Analysis and keeping 95% of explained variability, the feature vector becomes of 1379 dimensions.

Geometry Features

The geometry features are obtained trough the CE-CLM models, and consist of non-rigid shape parameters and landmarks locations (p and $\phi_i p$ in equation 3.2). This results in a 227 dimensional vector representing the geometry features.
Summed to the appearance features, this leads to a 1606 dimensional vector that defines the appearance of the face.

Neutral Expression Extraction

To extract some of the Action Units it is very important to have a neutral starting expression. There are personal differences on how we appear while in a neutral resting state, such as people looking naturally more cheerful or sad then others [**normexpr**]. To address this issue there needs to be a person specific calibration, done by adjusting for neutral expressions [**Baltru2013**].

Neutral expression adjustments are done by computing the median value of the face descriptors in a video, leading to a neutral expression descriptor. This works assuming that in a video most of the frames will contain a neutral expression, and this should hold true especially for real life situations where most of the time the interactions are performed with a neutral expression [**NatAffData**].

Once the neutral descriptor is computed, it is subtracted from the feature descriptor, giving normalized features.

Classification and Regression

The next step after the feature extraction is to recognize the Action Units from each frame of the video. Both intensity and presence are extracted.

Action Units detection is performed through Support Vector Machines (SVM), and Action Unit intensity using Support Vector Regression (SVR), both using the liblinear implementation [**liblinear**]. In both cases a linear kernel is utilized, as more complex kernels had no effect on performances and were quite slower to train.

3.5 Deception Detection

In this step we feed the extracted action units from the videos to a Support Vector Machine for classification. We use a Support Vector Machine because after a lot of testing with different techniques, the SVM was the one that performed the best compared to others, as we can see from par. 4.4.

3.5.1 Data pre-processing

Before feeding the videos to the SVM for Action Units extraction we had to make sure the videos were in good enough condition to be processed. That is not given since all this videos come from real life trials and the recording conditions are not optimal.

Being in an acceptable condition means that the face should be clearly visible, and the video has to have the specific person visible and talking in at least *most* of the video, and no other people on it, so to avoid training on different subjects of which we are not sure if they are lying or telling the truth.

As a consequence many videos had to be cut, trimmed or discarded to reach an acceptable level.

After extracting Action Units intensity and presence from our dataset of videos (par. 4.2) we put the data in a *csv* format to be able to process them and pass them to the SVM.

The data are divided for both AUs intensity, ranging from 0 to 5, and AU presence, having the boolean value of either 0 or 1 if the AU was missing or present in the specific frame of the video.

AU12_r	AU14_r	AU15_r	AU17_r	AU20_r	AU23_r	AU25_r	AU26_r	AU45_r	AU01_c	AU02_c	AU04_c	AU05_c	AU06_c	AU07_c	AU09_c	AU10_c	AU12_c	AU14_c
1.33	2.2	0.22	0.12	0	0	0.53	0	0.83	0	0	1	0	0	0	0	1	1	1
1.31	2.11	0.22	0.04	0.02	0	0.47	0	0.73	0	0	0	0	0	0	0	1	1	1
1.3	2.07	0.28	0	0.02	0	0.35	0	0.65	0	0	0	0	0	0	0	1	1	1
1.28	2.11	0.31	0	0	0	0.26	0	0.58	0	0	0	0	0	0	0	1	1	1
1.27	2.11	0.42	0.03	0	0	0.21	0	0.56	0	0	0	0	0	0	0	1	1	1
1.26	2.12	0.37	0.04	0	0	0.28	0	0.5	0	0	0	0	0	0	0	1	1	1
1.31	2.08	0.45	0.13	0	0.01	0.36	0	0.55	0	0	0	0	0	0	0	1	1	1
1.44	2.05	0.47	0.26	0	0.02	0.26	0	0.61	0	0	0	0	0	0	0	1	1	1
1.55	2.04	0.51	0.35	0	0.02	0.18	0	0.68	0	0	0	0	0	0	0	1	1	1
1.56	2.01	0.43	0.38	0	0.01	0.12	0	0.69	0	0	0	0	0	0	0	1	1	1
1.53	2.06	0.3	0.36	0	0	0.17	0	0.67	0	0	0	0	0	0	0	1	1	1
1.52	2.08	0.27	0.38	0	0	0.18	0	0.71	0	0	0	0	0	0	0	1	1	1
1.55	2.14	0.19	0.38	0	0	0.27	0	0.72	0	0	0	0	0	0	0	1	1	1
1.59	2.1	0.23	0.34	0	0	0.33	0	0.71	0	0	0	0	0	0	0	1	1	1
1.6	2.11	0.32	0.36	0	0.03	0.37	0	0.67	0	0	0	0	0	0	0	1	1	1

Figure 3.12. Example of input data in csv format.

The next step is to remove all the rows that have both no intensity and no presence for AUs, since those rows are most likely errors from the extraction, deriving from the poor quality of the videos, or from the face not being recognized in some frames.

We organized the data in three sets, one concerning only AUs intensity, one with only AUs presence, and another one where we merged the previous ones together, in order to better analyze the data and understand it's structure.

3.5.2 Support Vector Machine Classifier

After pre-processing the data we use the train set to train a Support Vector Machine classifier, and then we utilize the trained model to evaluate on the test set and compare the results with the labels present on the test set.

Generally, each of n frames is converted in a vector of length 36, where 17 variables represent the intensity and 18 variables represent the presence of Action Units. The last variable is the class label.

So a given training input **AU** for our SVM, **AU** is a $36 \times n$ matrix composed of AUI_i and AUP_i, where AUI_i represents the intensity for action units in row i and AUP_i is the action units presence for row i.

These observations belong to two classes, truthful and deceptive, represented by a label $\mathbf{Y} = \{y_1 \dots y_n\}$, where $y_i = \{0, 1\}$ depending on whether the frame belongs to a deceptive or truthful observation.

$$\mathbf{AU} = (AUI_i + AUP_i, y_i) \tag{3.3}$$

An input vector au^* for the test set, this time of length 35, is formed the same way but without having a class label.

$$\mathbf{au}^* = (AUI_1^* + AUP_1^*, \dots, AUI_{35}^* + AUP_{35}^*)^T \tag{3.4}$$

The goal of our SVM classifier is to correctly classify the test observation based on the training done by using the data in the train set.

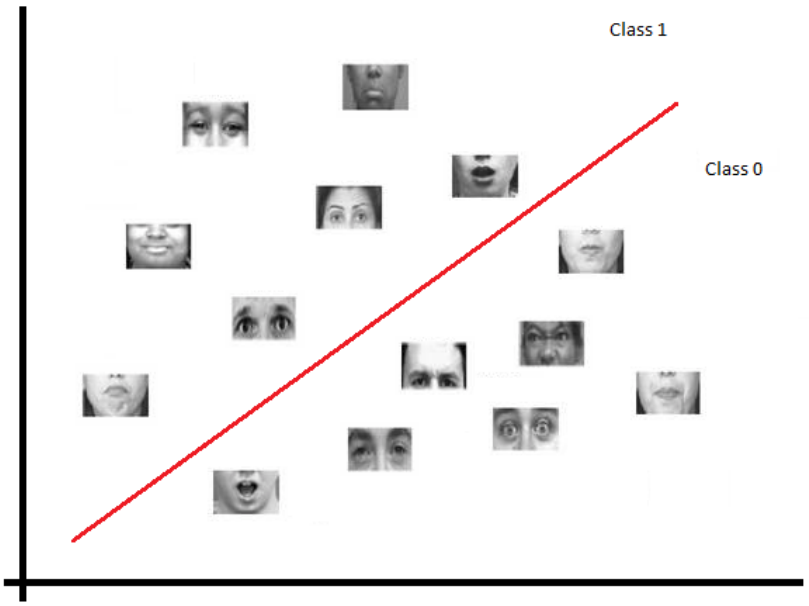


Figure 3.13. Visualization of SVM classification using Action Units.

Suppose we are using a *linear* SVM classifier. We would need to figure out weather a point lies on one side or the other of a dividing hyperplane, and we would look at this kind of equations to classify a test observation au^* based on the sign of $f(au^*)$:

$$f(au^*) = \beta_0 + \beta_1 au_1^* + \beta_2 au_2^* + \cdots + \beta_{35} au_{35}^* \tag{3.5}$$

if $f(au^*)$ is positive, then we would assign the test observation to class 1, and if it is negative, we would assign it to class 0 (fig. 3.13).

Of course there are two things we need to consider:

1. We need to keep the margin M of the hyperplane in consideration;
2. The relationship between features is not linear so we could need to enlarge the feature space;

As explained in par. 4.3.5, to get the best margin we use a grid search and find the best C and γ parameters.

To enlarge the feature space of our Support Vector classifier we make use of kernel functions (par. 2.4.6) and tried linear, sigmoid and radial kernels.

Chapter 4

Experiments

This describes the stack used for this thesis and the experiments done to achieve our results: I will start by reviewing the configurations and libraries utilized (par. 4.1), the database used to acquire train and test set (par. 4.2), all the analysis done on the data (par. 4.3), and finally we will discuss the results of this work (par. 4.4).

4.1 Building Blocks

The high level tools and libraries utilized o realize this work were:

- OpenFace: a collection of libraries and tools to train the models for face detection, landmark detection, feature extraction and action unit recognition.
- R + R-Studio: the environment where I implemented the code, with some important libraries, such as:
  - e1071: library for SVM classification (implementation of libsvm);
  - ParallelSVM: library to perform SVM training in parallel using all the cores available;
  - Random Forest: library to perform Random Forest and calculate variable importance;
  - Corrplot: library for calculating and visualizing correlations.

The experiments were conducted on two systems, Windows and Mac OS, with the following specs:
The windows machine has 8GB of RAM, i5 processor, and 4096MB ATI AMD Radeon R9 GPU.
The Mac has 4gb RAM, i5 processor, and an intel integrated GPU.
SVM training took about one hour on both machines.

4.2 Real Life Trial Database

This section is about the database we used to perform our experiments: it comes from the work done for the paper "Deception Detection using Real-life Trial Data" [**Perez-Rosas:2015:DDU:2818346.2820758**].

Figure 4.1. Examples of images from the dataset videos. [**Perez-Rosas:2015:DDU:2818346.2820758**].

The dataset is gathered from real-life trial videos available on YouTube and other public websites. The dataset also contains statements made by exonerees after exoneration, and some statements from defendants during crime-related TV episodes.

The first step to collecting the dataset was to identify public multimedia sources where the recordings of the trials were available, and deceptive and truthful behavior could be observed and verified.
The videos are of trial recordings where the defendant or witness in the video can be clearly identified, the face is visible enough during most of the clip duration, and the visual quality should be good enough to accurately identify the facial expressions (fig. 4.1).
There are three outcomes for the trials that were considered to label the videos as deceptive or truthful: guilty, non-guilty, and exoneration.
For the guilty verdicts the deceptive clips are taken from the defendant in the trial, while the truthful clips are gathered from the witnesses. There are also instances where the deceptive videos are of suspects denying a committed crime, and truthful ones are from the same person answering questions that where verified by the police as truthful.

In regards to the witnesses, if the testimony is verified by a police officer then it is labeled as true.
Testimonies that help the guilty party are labeled as false.
Exoneration (reversal of the sentence) testimonies are regarded as truthful.

The original dataset consists of 121 videos, 61 of which are deceptive and 60 truthful.

The average length of the videos is 28.0 seconds. The average video length for deceptive videos is 27.7 seconds, while the one for truthful videos is 28.3 seconds. The data consists of 58 total subject, 22 females and 36 males, with ages between 16 and 60 years.

This dataset was annotated following the MUMIN coding scheme for hand movement and facial displays. While annotating, the annotators were able to chose only one label per gesture, for every clip.

Fig. 4.2 shows the frequency counts associated with the gestures considered during the annotation.

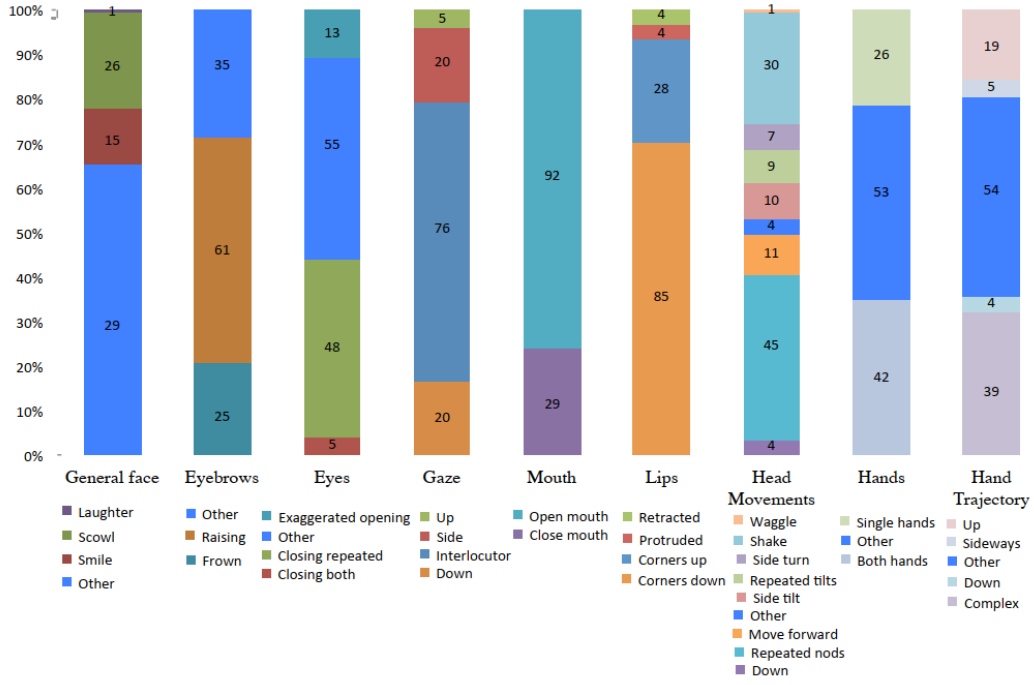


Figure 4.2. Frequency of gestures annotated in the videos from the database [**Perez-Rosas:2015:DDU:2818346.2820758**].

We modified this database in the following way:

- We cut or removed videos where the where the face was covered, hidden or in any way very difficult to see or recognize;
- We also cut parts of videos that had multiple subjects in the scene because Action Unit extraction works on one person only at the moment;
- Since it is necessary to see the person's face to perform AU extraction, we also avoided some videos in which the subject was not visible while talking, but the interlocutor was shown instead.

With deceptive videos labeled from #1 to #61 and truthful videos labeled from #1 to #60 we cut and removed the following videos from the original dataset, since they showed signs of the problems just described:

- **Deceptive**:
  - CUT: Video numbers 46, 47, 48, 49, 52, 54, 56;
  - REMOVED: Video numbers 50, 53, 55.
- **Truthful**:
  - CUT: Video number 7, 28, 43;
  - REMOVED: Video numbers 12, 31.

We also performed a division of subjects in the training and test set "by subject", splitting data to around 75% for the training set and 25% for the test set, to train the classifiers in a way that it wouldn't adapt to specific people, a problem that could occur since there are many videos with the same subject. In fact, the same person never appears both in the training and in the test set as an effort to completely avoid this problem.

The training set consists of 72451 observations of 36 variables, and the test set is composed of 14198 observations of 36 variables.

4.3 Data Analysis

In the following sections we describe the methods used to analyze the data and the machine learning techniques to classify them.
We start with comparing the occurrences for deceptive and truthful videos on the training set (par. 4.3.1), then look at the correlation between variables (par. 4.3.2), then we proceed to explain how we used GLM (par. 4.3.3), Random Forest (par. 4.3.4) and Support Vector Machines (par. 4.3.5) to classify the data belonging to the test set.

4.3.1 Data Comparison

The first thing I did, after cleaning the data, was to compare the extracted deceptive (fig. 4.3) and truthful (fig. 4.4) AUs occurrences (only for presence, not intensity) from the train set, which is shown in fig. 4.5.

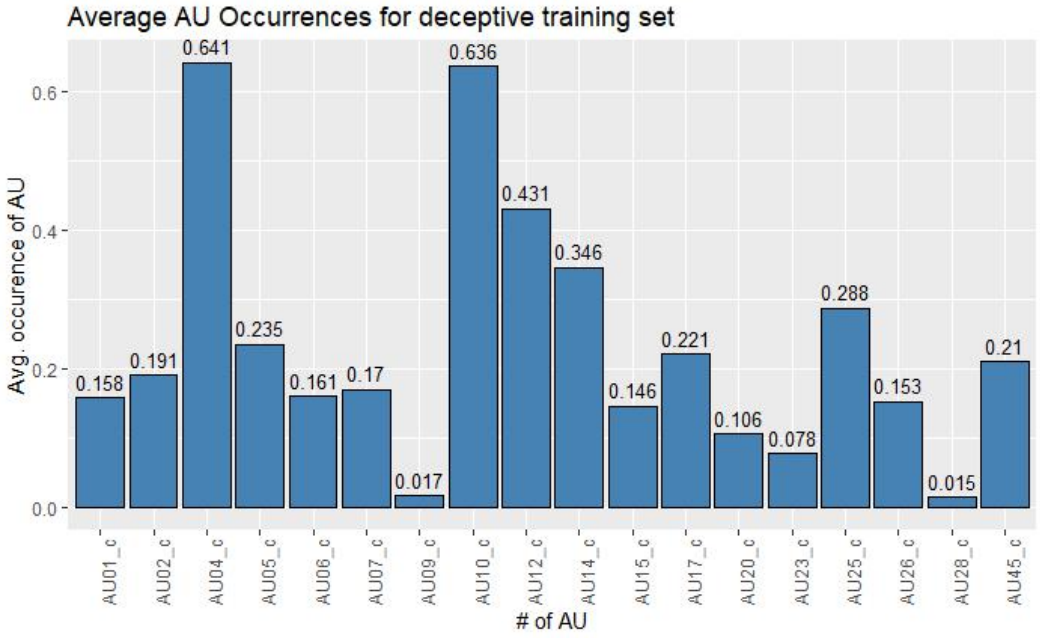


Figure 4.3. Average AU Occurrences for the deceptive training set.

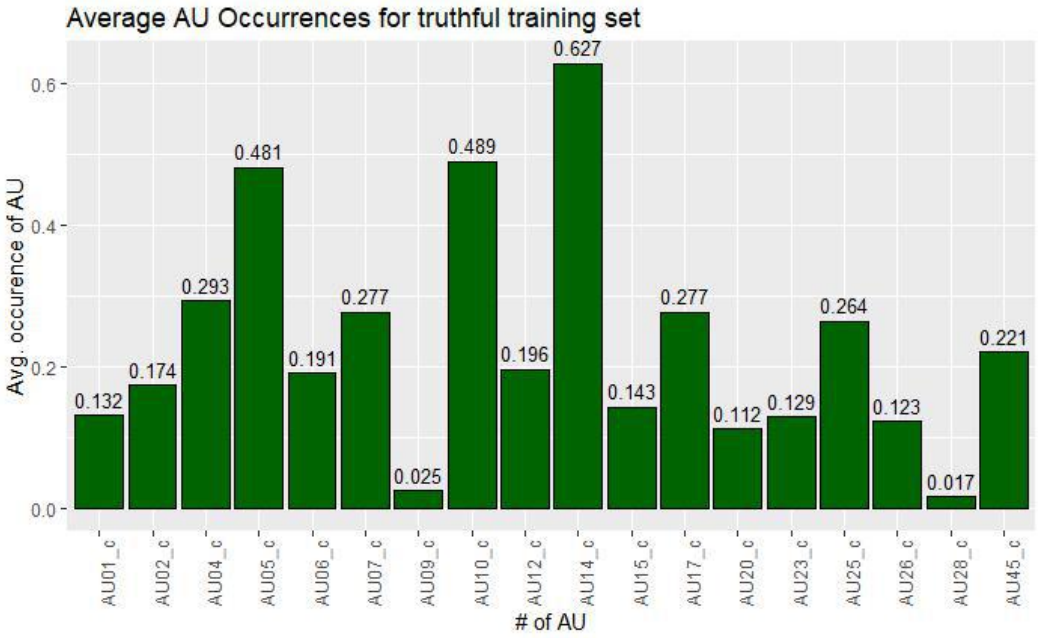


Figure 4.4. Average AU Occurrences for the truthful training set.

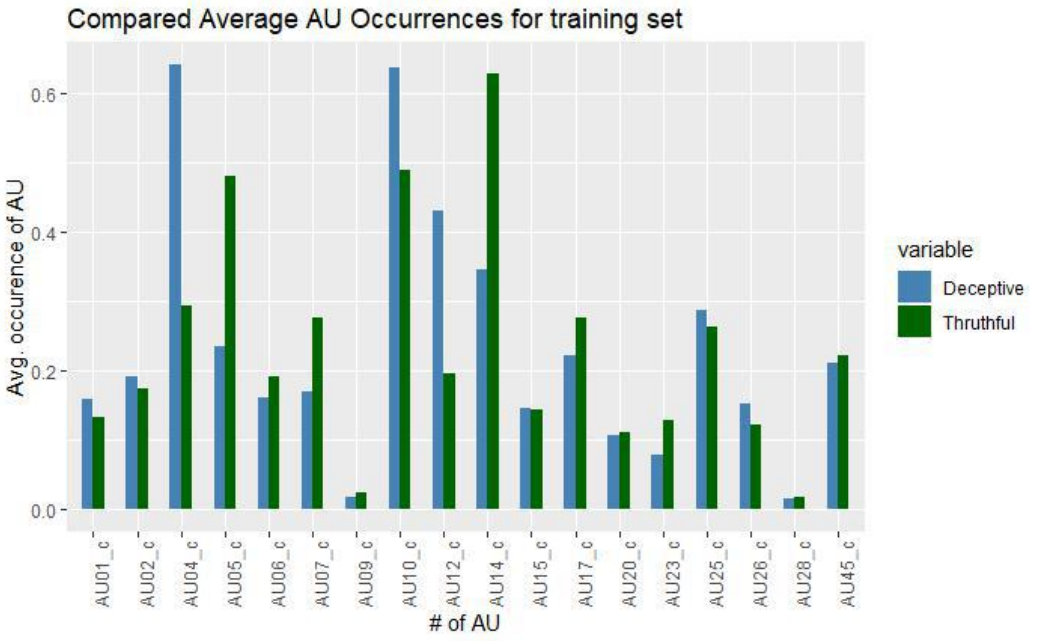


Figure 4.5. Comparison of AU Occurrences for the training set.

This comparison shows interesting differences between truthful and deceptive occurrences for AU04 (Brow Lowerer), AU05 (Upper Lid Raiser), AU10 (Upper Lip Raiser), AU12 (Lip Corner Puller) and AU14 (Dimpler).

4.3.2 Correlation

Variables correlation is shown in figure 4.6 for both presence and intensity. When the AUs label ends with "_r" it indicate intensity and "_c" means presence.

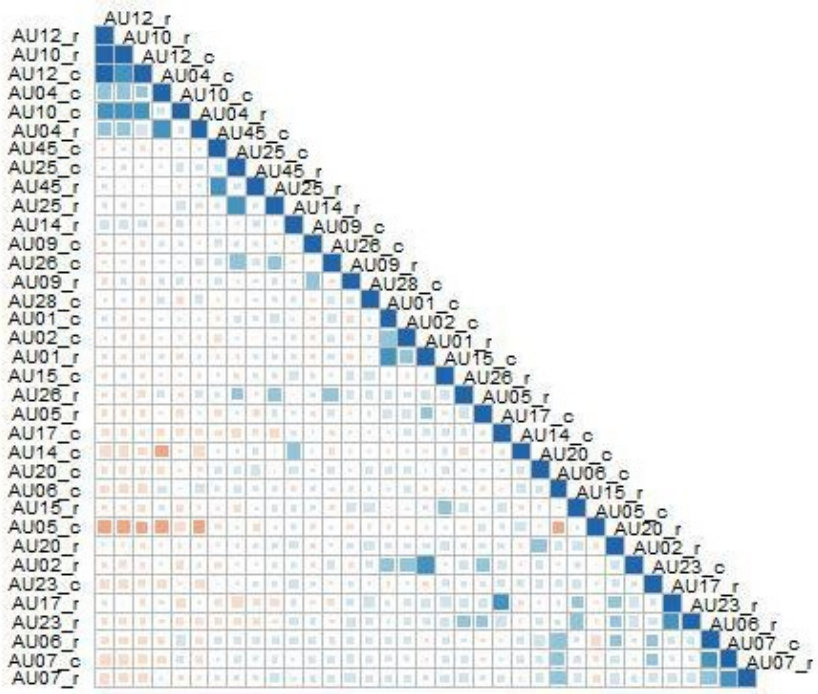


Figure 4.6. Correlation between variables for both presence and intensity in the train set.

The correlation matrix shows some obvious correlations between presence and intensity of the same AU, for example AU12_c and AU12_r, and some less obvious correlation between, for example:

- AU14_c (Dimpler) and AU04_c (Brow Lowerer);
- AU05_c (Upper Lid Raiser) and AU12_r (Lip Corner Puller) or AU10_r (Upper Lip Raiser);
- AU02_r (Outer Brow Raiser) and AU01_r (Inner Brow Raiser);
- AU17_c (Chin Raiser) and AU23_c (Lip Tightener).

Some of these correlation make anatomical sense, while others do not and could indicate a pattern, for example the one between AU14 and AU04.

4.3.3 GLM

Another quick experiment I made was with the Generalized Linear Model in R, which translates into Logistic Regression (par. 2.2), at least to have an idea of the resulting p-values and variable significance, as seen in fig. 4.7:

	Pr(>|z|)	
(Intercept)	2.58e-15	***
AU01_r	0.60606	
AU02_r	1.14e-06	***
AU04_r	1.21e-12	***
AU05_r	7.80e-12	***
AU06_r	3.14e-09	***
AU07_r	< 2e-16	***
AU09_r	< 2e-16	***
AU10_r	< 2e-16	***
AU12_r	< 2e-16	***
AU14_r	< 2e-16	***
AU15_r	< 2e-16	***
AU17_r	2.46e-13	***
AU20_r	< 2e-16	***
AU23_r	0.83786	
AU25_r	< 2e-16	***
AU26_r	3.88e-10	***
AU45_r	< 2e-16	***
AU01_c	1.59e-07	***
AU02_c	< 2e-16	***
AU04_c	< 2e-16	***
AU05_c	< 2e-16	***
AU06_c	6.30e-08	***
AU07_c	0.00136	**
AU09_c	2.77e-15	***
AU10_c	< 2e-16	***
AU12_c	< 2e-16	***
AU14_c	< 2e-16	***
AU15_c	< 2e-16	***
AU17_c	0.76875	
AU20_c	< 2e-16	***
AU23_c	< 2e-16	***
AU25_c	< 2e-16	***
AU26_c	< 2e-16	***
AU28_c	8.89e-08	***
AU45_c	< 2e-16	***

Figure 4.7. p-values from GLM for each AU (intensity and presence).

As we can see AU01_r (Inner Brow Raiser), AU23_r (Lip Tightener), and AU17_c (Chin Raiser) are not significant and that's an indication that we could remove them from the features used to make the classification.

The results for classifying data with GLM is 54.86% accuracy on the test set, which is a little better than chance but definitely not great, and shows that probably a linear relation does not occur in this instance.

4.3.4 Random Forest

I've decided to use Random Forest (par. 2.3) to estimate variable importance, shown in fig. 4.8.

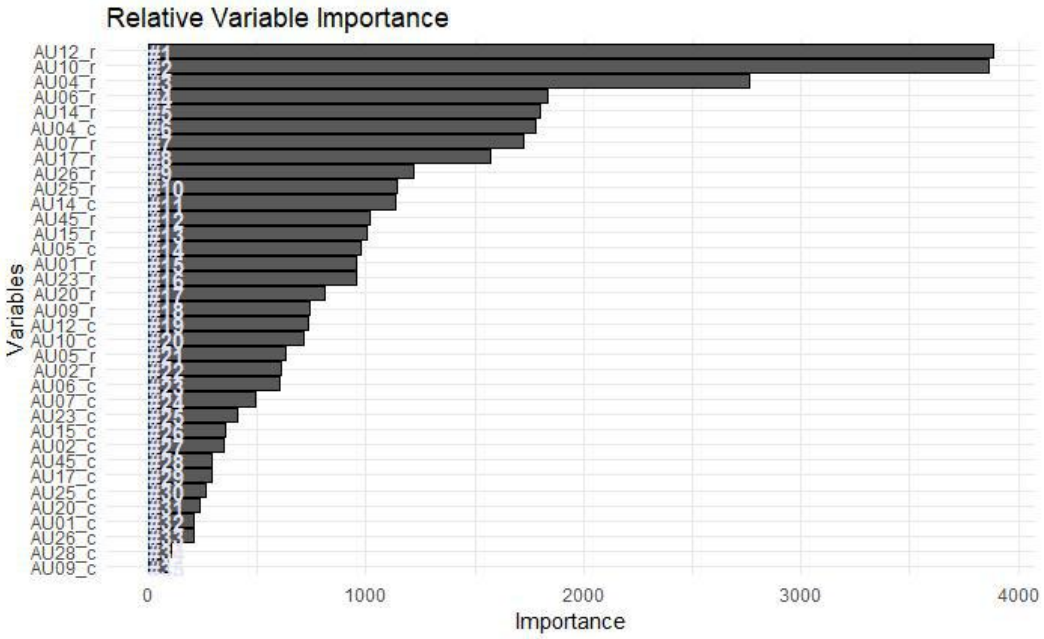


Figure 4.8. Variable Importance extracted from classifying data with Random Forest.

The number of grown trees are 200 as going further did not increase the prediction accuracy at all.

Figure. 4.8 shows that the intensity of AUs 04, 06, 07, 10, 12 and 14 are the most significant, while the sole presence of an AU is not particularly significant, beside for AUs 04, 05 and 14.

As we can see AUs that have low p-values in GLM have also low importance in Random Forest.

Classifying with Random Forest gave me 58,42% accuracy on the test set, which is better than GLM but it is still not good enough to be significant.

4.3.5 Support Vector Machine Classification

To approach classification with a Support Vector Machine we tried both linear, radial and sigmoid kernel.

The cost c and γ parameter were found using a tuning function that performs a grid search over the combinations of cost and gamma for all the kinds of kernels we used, and then by empirically changing the values to see if better results could be achieved.

In the tuning function, γ varied from 10^{-5} to 10^{-1} and c from 10^{-3} to 10^{2}, growing by 10^{1} every iteration.

After the grid search I did some experimental changes and found empirically that the best parameters for the linear kernel were $C = 1$ and $\gamma = 0.04$

The results for classifying with a sigmoid kernel are 67% accuracy on the test set (to be precise 66.99394), which is a good starting result, while the linear kernel performed worse at around 61% accuracy.

The best scoring kernel was actually the radial kernel, giving us 72,64206% accuracy when predicting on the test set, by setting $\gamma = 0.001$ and $C = 3$, obtaining the values of the parameters via both the tuning function and trial and error.

4.4 Results

Using Support Vector Machine we were able to classify deceptive and truthful videos with an accuracy of 72,64%.

The table below shows a comparison of the results obtained using different machine learning algorithms, and different kernels for SVM.

Algorithm	**Accuracy**
Logistic Regression	54.86%
Random Forest	58,42%
SVM - Linear	61,56%
SVM - Sigmoid	66,99%
SVM - Radial	72,64%

Table 4.1. Comparison of the results obtained with different approaches for this dataset.

Compared to other similar work (table 4.2) using action units and micro expressions our result is in line with the others, and it shows promise as we don't consider it over and we are confident it will improve by going forward and implementing the ideas exposed in the "future work" section (par. 5.2).

Research Paper	Accurac
Deception Detection in Videos [**DBLP:journals/corr/abs-1712-04415**]	75,02%
Deception Detection using Real-life Trial Data [**Perez-Rosas:2015:DDU:2818346.2820758**]	70,24%
Does "lie to me" lie to you? An evaluation of facial clues to high-stakes deception [**SU201652**]	76,92%
Our work	72,64%

Table 4.2. Comparison of our result with similar researches.

Chapter 5

Conclusions

5.1 Final Considerations

In this thesis we presented a way to discriminate between lies and truths, based on the extraction of Action Units presence and intensity from videos, through the use of an SVM classifier.

The applications for a system that can accurately detect detection are numerous and space in a lot of fields:

- In airport security it could be used to detect ill willed people;
- An automated deception system for security checks, for example analyzing the response to simple questions;
- It could be used by the police force as an aid for interrogation;
- In politics it could be adopted by voters to analyze speeches from political candidates.

Of course there are privacy concerns that need to be answered when starting to utilize a potentially control oriented technology, first of all privacy invasion.

5.2 Future Work

Some possible developments to be applied to this thesis are:

- One of the difficulties of this kind of studies is the lack of a big dataset for high stake deception. It would be beneficial to gather such dataset to improve the results, bettering also the quality of the videos;
- Extend Action Units extraction to more than one person in the scene. It would be interesting to consider direct interaction with questions and responses;
- Using temporal information for each video to capture the beginning and the end of the interaction. The lie does not necessarily appear in *all* the video, but probably just at some point, even though the majority of the statements are

either deceptive or truthful. We were thinking about implementing a LSTM to observe the temporality;

- A multimodal approach to consider more than just the face. For example we could consider the body and analyze the voice and speech patterns to extract information from the subject and use thermal camera to account for the blood flow near the periorbital areas.

List of Figures