University of Hradec Králové
Faculty of Informatics and Management
Department of Informatics and Quantitative Methods

Text classification with artificial neural networks

Bachelor thesis

Author: Anouk, Wilstra
Field of study: Applied Informatics

Supervisor: Ing. Milan Košťák

Hradec Králové 08 2022

Declaration:

I declare that I prepared the bachelor thesis independently and with the use of the mentioned literature.

In Hradci Králové dated 14.8.2022 Anouk Wilstra

Acknowledgement:

I would like to show my appreciation to my supervisor, Ing. Milan Košťák, for the provided guidance throughout the project. I hereby also thank my close friends and family for their support in the past year.

Annotation

Artificial neural networks allow for the modelling of more complex problems, including natural language processing. The ability for computers to read and understand human written text is critical in the age of big data and artificial intelligence, and it is heavily researched. This thesis uses neural networks for author profiling, classifying texts into three author age categories. Three different neural network types are used to create a whole range of models, namely feedforward- (FNN), convolutional- (CNN) and recursive neural networks (RNN). Each of the types has its own strenghts and reasons to be used. FNNs are the most generic and work well for most problems, CNNs use feature extraction and RNNs specialize in sequential data.

The subsequent results are then compared with the results achieved using a different machine learning technique, naive Bayes. Some models resorted to guessing, thus reaching accuracies as high as ∼70%, whereas others learned, achieving worse results, around 50% only. After many experiments, the work concludes that naive Bayes is better suited for this data set, with CNNs a close second, and provides reasons why this may be so, along with further research options that may help the neural networks beat the baseline accuracy.

Anotace

Název práce: Klasifikace textu pomocí umělých neuronových sítí

Umělé neuronové sítě umožňují modelování komplexnějších problémů včetně zpracování přirozeného jazyka. Schopnost počítačů číst a rozumět textu napsaného člověkem je v době big data a umělé inteligence kritickou a velmi zkoumanou oblastí. Tato práce se věnuje využití neuronových sítí pro profilování autorů, konkrétně klasifikaci textu do tří kategorií věku autorů. Pro vytvoření celé řady modelů je využito třech různých neuronových sítí, jmenovitě feedforward (FNN), konvolučních (CNN) a rekurzivních (RNN). Každý z typů má své vlastní přednosti a důvody pro použití. FNN jsou nejuniverzálnější a fungují dobře pro většinu problémů, CNN využívají výběru rysů a RNN se specializují na sekvenční data.

Následující výsledky jsou porovnány s výsledky dosaženými využitím jiné metody strojového učení, naivního Bayesova klasifikátoru. Některé modely se uchýlily k hádání, čímž dosáhly až ~70% přesnosti, zatímco jiné se učily, dosahujíc horších výsledků, pouze okolo 50 %. Práce po vykonání mnoha experimentů vyvozuje závěr, že naivní Bayesův klasifikátor je vhodnější pro použitou sadu dat, těsně následován CNN, a vysvětluje důvody, proč tomu tak může být a zároveň popisuje další možnosti výzkumu, jak pomoci neuronové síti překonat minimální přesnost.

Contents

List of Figures

List of code samples

List of tables

1 Introduction

Since the start of the digital age, the amount of data has been growing and the need for algorithms to process big data arose. The data sets nowadays are generally too large or simply too complex to process using traditional software and specialised methods are needed. Many man-hours would be needed for some of these processes to be run to completion, which is why these tasks are preferably transferred to computers. Machine learning algorithms are especially popular, as they can provide useful outputs without explicitly being told how.

Personal data collected online can provide immense insights into the users that visit our website, and can be used to personalise the user experience, generally for the purpose of making a profit. This can be done directly, through targeted marketing, or more indirectly, by showing more articles the user interacts with or likes, so they spend more time on the website, during which time more advertisements are shown. It can also be used to manipulate people or influence their decisions. Therefore, there are strict privacy laws in place regarding personal data online. Although making the collection of this data harder, it still doesn't stop the interest in trying to analyse who the user is.

One possible method to find out more about our users without breaking these privacy laws is through their textual communication. Author profiling tries to link texts to features of the writer. We may, for example, want to know the sex or the age of the user. Author profiling falls into the category of natural language processing, a field where a computer tries to understand written or spoken text in a way a human does. The computers generally work faster and can, in many instances, achieve as high accuracies as human workers, making this a very cost-effective option.

Several machine learning algorithms can be used, but commonly deep neural networks, naive Bayes, support vector machines and logistic regression are used. They are all based on relatively simple mathematical models, but are run repeatedly or recursively, and again, doing them by hand or in an excel sheet wouldn't be effective. Another particular feature about these mathematical models is that they

learn over time. So not only can computers now process and analyse data about users faster than humans can, but they also improve their results over time.

Many libraries are prepared to make the programming of these algorithms as simple and effective as possible, but still, a substantial amount of preprocessing of data and tweaking are required. This process can be slow and requires knowledge of the algorithm and used data set. Analysing and comparing different algorithms and models is therefore essential.

2 Aim

This bachelor thesis aims to discuss and create deep learning models to classify texts. More specifically, create a neural network model that beats the baseline accuracy, calculated using naive Bayes, to extract information about the texts' authors. The feature of focus is age, classifying the users into three categories: 10s (age 13-17), 20s (age 23-27) and 30s (age 33-39).

To achieve this goal, three different variants of neural networks will be built, tested and compared to the baseline results and to each other. The types of networks in question are a feedforward neural networks (FNN), a convolutional neural network (CNN), and a recurrent neural network (RNN). Finally, the best-performing model will be selected and experimented on to attempt to improve its results even further.

2.1 Hypothesis

I expect the baseline calculated using the naive Bayes algorithm to lay on the slightly lower end of the summarising accuracies provided by Cesare et al. (Cesare et al., 2017) Although the TF-IDF method can harvest good results, I think that the blog topics are so widely spread that it will struggle to reach astonishing results such as in Goswami's study (Goswami et al., 2009) using only that, yet still reach an acceptable 60-70% accuracy.

I hypothesise that the LSTM model will outperform the other two selected neural network models, but only by a few per cent. I believe that using these more basic models, I can reach accuracies around 70-85%, depending on the complexity of the model.

Fully connected dense neural networks contain a innumerable links and information but no context. Convolutional neural networks' focus lies on feature extraction, and I am of the opinion that it could outperform the basic feed-forward network by a lot, provided it is set up correctly. It will require some testing with different numbers of filters and pooling layers to get the network to focus on the decisive textual features and remove additional noise. I believe that the difference between running the CNN with stopwords and running it without stopwords can

result in wildly different results because of the filtering performed. If more filtering has happened in advance, it could reach interesting results faster, possibly scoring better.

Long Short-Term Memory networks were designed especially for (textual) sequences and retain some information from previous words later on in the analysis. Although I believe it can outperform the other two types, I think this difference would be more apparent in generative or summarising experiments, where the context is even more important than here. Considering that scientists already can reach results around 80% based solely on features in text, I could be wrong, and the CNNs may end up beating the other models.

3 Related research

The following chapter outlines previous research in related fields: author profiling using different machine learning techniques, natural language processing and neural networks.

3.1 Author profiling

It is well known that different age groups use different vocabulary and stylization, and this is a heavily researched topic. (Argamon et al., 2007b, 2009; Hinds, 2018; Huang & Paul, 2019; Peersman et al., 2011) The fascinating part is that most studies use alternative machine learning methods, with many handpicked language features they keep their eyes out for. The results are generally good, often reaching 80-90% accuracy, but a lot of research analyzing the language and choosing the relevant features precede the machine learning stages.

Researching the use of automated detection of demographic traits on the internet is becoming more popular. However, other machine learning topics such as image recognition and text prediction still remain in the lead. Although I could not find any numerical data, tutorials and articles on the latter two are more numerous and easier to find. The articles mentioned in the summarising article *How well can machine learning predict demographics of social media users* (Cesare et al., 2017) are all from 2006 and later, whereas papers on the use of neural networks for image recognition were already innumerable by the mid-90s. Although some pieces on author profiling existed too, they were rarer.

Even among the articles on author profiling, neural networks were not the only type of machine learning algorithms used, with other frequent methods including support vector machines, naive Bayes, k-nearest neighbours and random forests. The other traits that are typically explored are gender and ethnicity. A study comparing several approaches found that gender was the easiest trait to identify accurately, whereas age predictions, although using the same methodological approach, ended up with a wider range of accuracies. (Cesare et al., 2017) The results for all the machine learning models combined were between 43% and 95%

accuracy, with the age-specific studies using neural networks reaching above 70% accuracy and a deep neural network reaching results up to 94% accuracy. (Guimaraes et al., 2017; Kim et al., 2017; Peralta et al., 2021)

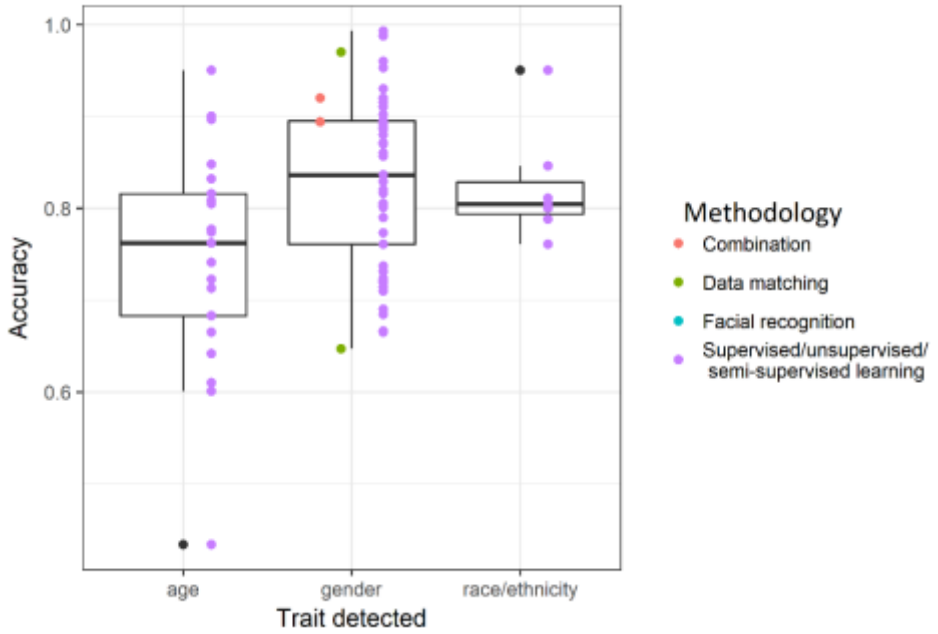


Figure 3-1: Distribution of predicted accuracy by trait, looking at 160 machine learning studies. (Cesare et al., 2017)

Despite this sounding high, other studies based solely on language features and their use among different demographics already reached accuracies as high as 80%. (Argamon et al., 2007b; Goswami et al., 2009) Goswami and his team, for example, used word frequencies to categorise the texts correctly.

Non-dictionary Word	Gender Variation		Age Variation	
	Male	Female	10s Age	30s Age
Yay	10.7294	44.3713	35.5822	17.1137
wat	12.433	57.5201	4.5026	20.7462
tv	21.6521	56.5894	37.3394	29.7338
Thats	10.4031	20.4301	207.2521	27.2996
thats	41.2864	176.967	207.2521	27.2996
sunday	14.6321	42.0974	30.1094	9.2871
saturday	14.0642	44.6894	32.2876	8.5007
sa	20.5888	76.5011	37.5407	9.1373
ok	74.8883	295.5773	252.0228	97.3277
nite	11.7564	41.143	37.0831	8.7628
ng	11.5147	58.0621	31.7568	8.0513
na	30.3999	117.2907	73.123	12.4327

Figure 3-2: Partial list of non-dictionary word frequency per 100 000 words in gender and age classes as presented by Goswami et al., 2009

Whilst these results on their own are already interesting, they mean little without additional performance measurements such as recall, precision, F1 values or UAC. This is because the accuracy can be high on overfitted models and high accuracy on its own could in reality indicate a false good result, not performing well on never seen data. The Deep Convolutional Neural Network (DCNN) model obtained values as high as a recall of 95.6% and an F-measure of 93.0%, although this was performed on a binary dataset and the results of multiclass predictions are generally found to be lower. Specifically, a comparison showed that the F-score went down by as much as 0.2, increasing the binary testing to a 4-category multiclass experiment, scoring 0.91 for binary, 0.73 for three categories and 0.71 for 4 categories. (Cesare et al., 2017) Interestingly enough, predicting a numeric age rather than class-based, results in an F-score of 0.73, suggesting that unless performing binary analysis, a regression model could make more sense. Others, already mentioned above, on the other hand, suggest that it is possible to get a lot better results if the age gap is large enough, as the differences between the groups are larger. Another alternative is to use more sophisticated algorithms, such as deep neural networks. Also interesting is that for age specifically, adding metadata actually, unexpectedly slightly decreased the accuracy of the classification, going from 76% accuracy to 74%.

Nowadays, a large number of articles on or related to author profiling are written at PAN, a conference focusing on stylometry. It has been the most popular topic,

available to pick from since 2013, with more than 40 publications on the topic in the last 3 years alone. (Potthast et al., 2009)

Almost all the above-mentioned studies focus on the English language. The assumed reason is the abundance of data in the English language. One study used ten different machine learning algorithms on non-English texts, multilayered perceptrons being one of them, but they performed relatively poorly, reaching approximately 50-75% accuracy, with, in some cases, it performing worse than the baseline. (Duc Pham et al., 2009)

3.2 State-of-the-art results

Neural networks are generally gigantic and can contain thousands to billions of internal variables to adjust. We have between 90 and 100 billion neurons in our brains. (Chris Firth et al., 2014) In comparison, the largest artificial neural network is the Generative Pre-trained Transformer 3 (GPT-3), containing 175 billion parameters accepting an unprecedented 2048 tokens as its input. It was trained on almost all the accessible data on the internet at the time and provides many interesting tasks, of course including the more typical classification tasks and grammar correction, but also more unusual jobs like converting a movie to an emoji, explaining code, a recipe creator and a sarcastic chat bot. It can also learn entirely new tasks by only looking at a small number of examples. It is based on transformers, a relatively new (2017) type of neural network not relying on encoder-decoder combinations, convolutions or recurrence and instead uses attention mechanisms. With the many parts, it is not viable to implement something similar in this work, but as the leading ANN in NLP, it is worth mentioning. (OpenAI, 2020a, 2020b; Romero, 2021; Vaswani et al., 2017)

Although GPT-3 can perform many tasks, it can only provide one task at a time. Gato is a generalist agent that also uses a transformer-based neural network similar to large NLP neural networks to perform 604 different tasks simultaneously. These tasks cover a wide range: captioning images, controlling a robot arm to stack blocks, engaging in an interactive dialogue and playing games. It allows an input of 1024 tokens, a batched input of all the mini-tasks. (Reed et al., 2022)

Several of the articles in section 3.1 mention that they use state-of-the-art pre-trained models or achieved state-of-the-art results, but none of them mentions a specific number that they are trying to beat. The general accuracy to beat seems to be 70%, with some reaching higher and wanting to achieve above 80%, but some mentioning as low as 60%. Those numbers are significantly lower than the state-of-the-art results for other neural network applications, with image recognition using convolutional neural networks often reaching above 90%, the typical MNIST example using Keras and TensorFlow already reaching a 97% validation accuracy. (TensorFlow, 2022)

4 Sociological impact

Author profiling comes with large possible sociological impacts, both positive and negative. Two relevant examples are outlined below.

4.1 Internet usage and safety

Back in 2018, only 66% of adults over 65 used the internet compared to 98% of 18-29-year olds. (Internet Usage Statistics, 2021) This makes it a lot harder to find the needed data and researchers tend to treat age as categories with the last category simply being over 35 or 40+. (Hinds, 2018) The amount of data online has since more than doubled (Holst, 2021) and regression may now be used instead of the more vague classification. In the case of age, your language use between one year and the next is unlikely to change drastically, but between two separate categories, there may be a more significant gap. It may therefore be harder to get exactly correct answers when using regression. One study solved this issue by discarding intermediate age groups to remove the ambiguity. (Argamon et al., 2009)

Although the internet has many positives and provides endless possibilities and opportunities, it also comes with risks. Especially for school-aged children, there are 4 groups of risks: (niDirect, 2022; Raising Children Australia, 2006)

1) Content risk: Seeing upsetting content, such as animal cruelty, violence or sexual content.

2) Contract risk: Agreeing with unfair terms and conditions, agreeing to data collection, coming into contact with pages with weak internet security, and opening them up for other risks.

3) Contact risk: Adults posing as children, sharing information or meeting up with strangers.

4) Conduct risk: Cyberbullying

Points 1-3 give a good motivation for why author profiling focused on the age parameter should be studied. Although there are many counters that parents could

use, such as child-safe search engines, seemingly safe pages could still cause harm. If it would be possible to accurately estimate users' ages only using their written content, we could automatically flag underaged users from adult pages or, on the flipside, flag and block adults from child-oriented fora. Nowadays, metadata is already used in some "I'm not a robot"- checks, (Scott, 2019) but similar know-how is, to my knowledge, not yet used in automatic age classifications on the internet.

4.2 Ethics of author profiling

Demographics and why we should care about them is a far-reaching topic, that I will only briefly touch on. In general, demographic data is any array of socioeconomic information, often looking at gender, age, ethnicity, income and employment status. These pieces of data can then be used to see changes over time, e.g., growth or declines in the age in the population overall, of the adult population, population growth in general, diversity, gender distribution and income. This information is then used to plan and make investments in the correct places to either resolve related problems. An example mentioned is the number of elderly massive increasing resulting in the need for senior housing, certain health services and public facilities. (French, 2014) Author profiling online can furthermore be used to target users to personalize their content. Currently, one of the only ways to receive this data is by surveying or the census. The first is a tedious and expensive process and the second often contains older data and does not show the here and now. They do however show accurate information on many topics, like migration, language use, age, sex, number of children, foreign borns and computer use. (United States Census, 2022) Being able to automatize this process and find users that fit your needs could be revolutionary.

Having access to all this data about users can also be misused, such as selling the data or targeting individual people or groups to hurt them or make a profit, which is why we have to make some ethical considerations. Generally, when using personal data, explicit user consent has to be given, but when using such a sizeable data set as when working with neural networks, we instead rely on group privacy, that is they are not profiled as individuals but as members of a group. If the group is large

enough, the information provided is inadequate to pick out an individual and so, is still protected. This is why for instance, publishing the test results in a table with students' names can be considered a violation of privacy, whereas when published as a histogram of overall results, it is not. However, with sufficient pieces of information, e.g., previous test scores and scores of friends, we may (soon) be able to uncover enough sensitive information to identify individuals and therefore data mining is considered to be a grey area by many. To limit the data that companies can collect about their users and the actions they can perform with this data, policies like the General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) were enforced. (Bonta, 2018; Intersoft Consulting, 2018) For similar reasons, large social media websites with larger amounts of user data only let you scrape through special capped APIs. (Facebook help center, 2022; Twitter, 2022)

5 Theory

This chapter gives an overview of what artificial neural networks are and how they fit into the field, mentions different components of a neural network, possible data preprocessing steps and other theoretical knowledge needed to understand the practical network building in chapter 6: Implementation.

5.1 Natural Language Processing

Chapter 4 described some of the reasons why one may like to understand and analyse text, spoken or written. Natural Language Processing (NLP) is a branch of Artificial Intelligence, where computers try to do just this. It heavily overlaps with linguistics, and some of the methods are quite similar to human learning – for example, Part Of Speech (POS) tagging – that is identifying whether a word is a verb, noun or adjective – before performing sentiment analysis – analyse the emotion – or natural text generation – letting the computer write the following word or sentence. (Bird et al., 2009; Brownlee, 2017; IBM Cloud Education, 2020) Two of the methods routinely used in NLP will be explained in this thesis, namely neural networks and naive Bayes.

5.2 Neural networks

Artificial intelligence is most simply described as machines exhibiting some form of human-like intelligence. At the current date, there are no machines that have passed the Turing test and won the Loebner Prize golden medal. With the criticism the Turing test has received, and the lack of funding, the contest has been suspended and no one may ever win it. (AISBX, 2019; Powers, n.d.; Sundman, 2003) All intelligence achieved has been in a very narrow field, such as playing chess or checkers, driving driverless cars, making mortgage decisions and more. (Brighton & Selena, 2007; Conti, 2017; Redmon, 2017; Urmson, 2015) Machine learning is then a subfield of Artificial intelligence, where the machines learn by themselves, and artificial neural networks are a specific set of algorithms falling into this field.

Artificial neural networks are inspired by the human brain and the neurons within them. The neurons in the human brain are connected and communicate by sending

pulses to each other over the synapses between them. Each synapse triggers a release of neurotransmitters that change the data received and send the data on to the next neuron. (Saaty, 2000) Soon will be shown that this is very similar to how artificial neural networks function. From now on, when not specified, neural networks will refer to artificial neural networks (ANNs).

Describing an artificial neural network in simple terms is almost as complicated as explaining the human brain and can be looked at from many angles. One possible description is that a neural network is a self-learning mathematical model that accepts a set of examples (input and expected output) and, by manipulating the given data, can accurately predict the output of new, unseen input data. This thesis will focus on supervised learning, meaning a human has labelled the data in advance, giving the ability to compare results and decide whether the machine is correct or not at the time of its decision-making. Unsupervised learning contrasts this and lets the machine run more or less freely. Afterwards, a human goes in to see whether it spawned any helpful output.

Neural networks describe fuzzy concepts- meaning there is no definite rule or it may not necessarily have a perfect answer. Instead, a good enough rule and solution are sought after. Examples of their implementations include image recognition, filtering spam, mortgage calculations, finding outliers, and translations. Solving any of the above examples using traditional programming methods, would soon result in the many if-statements, making the programming and understanding harder.

Neural networks take an input, also named a feature and an output describing it, called a label. The combination of a feature and a label is called an example. An example could, e.g., be an image of a dog (feature) and the label being the string “dog”. In a different category, calculating the house price, the feature could be a list of selected house properties such as the number of windows, square footage, whether it has a pool etc. and the label being the actual worth of the house.

The example with the price of the house is a type of regression problem. We take several inputs and output a single number. The dog images being labelled as dogs is

instead an example of a categorization problem. We prepare some possible output classes and the class with the highest probability is the predicted class. The age can be predicted either as a number or as a category.

There are three common types of categorization:

1) In binary categorization, the output is either true or false. For instance, a person is either a child - under 18 – or and adult - 18 and over.
2) Multiclass categorization enables the use of more output classes. This is what will be used for the author profiling, as it allows the use of several categories. The categories that will be used are the 10s, 20s and 30s. Even though there are several classes to pick from, there is only ever one correct option. A person cannot fall in the age category 14-17 and at the same time be aged 33-39. It is important to ensure that the class options are disjoint.
3) Multilabel categorization allows several outputs. It can among other things, be used to label Wikipedia articles, so that related pages can easily be found. For example, an article on convolutional neural networks could be labelled artificial intelligence, machine learning and neural networks. One input, the article, gave several outputs, the labels.

5.2.1 The structure of a neural network

The basic structure of a simple neural network can be seen in Figure 5-1. Each grey circle represents a neuron, the basic building block of the network. A neuron can be thought of as a mathematical function taking some input and creating an output. The lines between them show which neurons are connected. Neurons are separated into layers, such that neurons in the same layer do not interact. Most neural networks have at least one layer where all neurons in that layer communicate with all neurons in the layer before and the layer after. Such layers are called dense and fully connected.

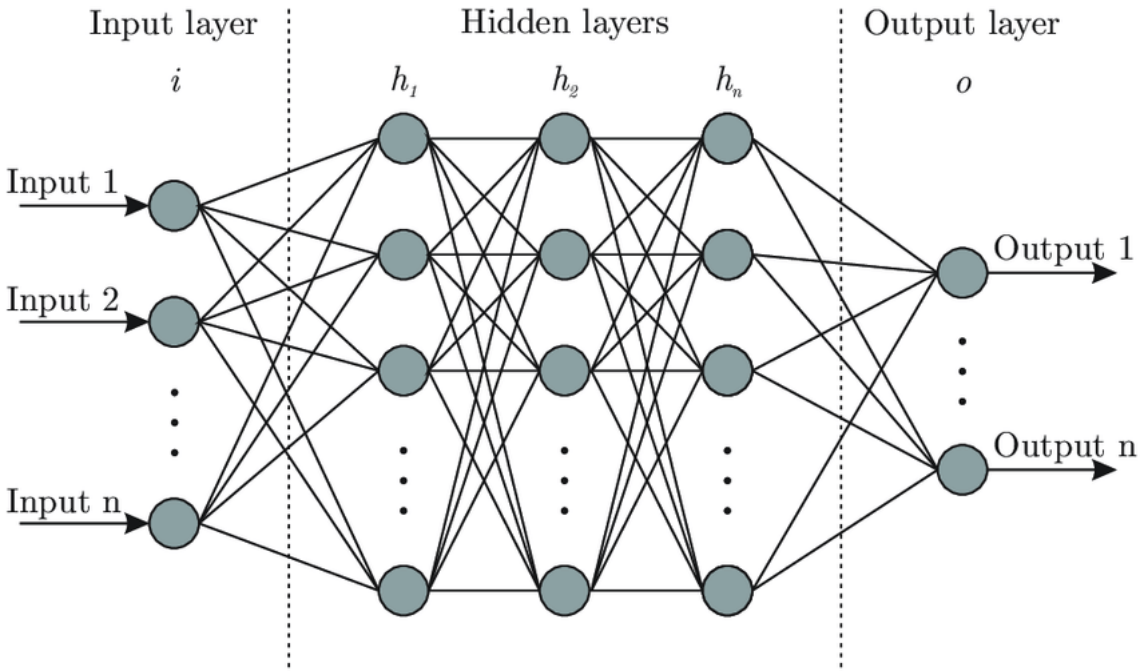


Figure 5-1: Artificial neural network architecture (Bre, 2017)

To go from inputs to outputs, some mathematical operations have to be performed. This is done in the in-between layers, commonly called hidden layers. These hidden layers can be a number of types that will be described in the next section, 5.2.2: Types of layers and models.

5.2.2 Types of layers and models

Summarises the neural network types used in this project and the different layers within them.

5.2.2.1 Dense

"You have between 90 and 100 billion of them [neurons], yet not one of them has any idea who you are. But somehow, by chattering among themselves across billions of interconnections, neurons conjure up your self-awareness" (Chris Firth et al., 2014)

Just like in the human brain, a set of neurons firing should cause a reaction in the next layer of neurons. In the most basic hidden layers, called dense layers, each neuron has several internal variables, namely one weight for each incoming value from neurons in the previous layer and one value titled bias. The output of these neurons is calculated as the sum of the individual weights times their respective

values (weighted sum) and is offset by a bias. The weights describe the importance of that neuron's activation for the following layer's neuron, where the size is proportional to its influence and a negative indicates a negative effect.

The bias is then used to discard any small activations considered to be insignificant. The most simple version of this, taking several binary inputs and creating one binary output was already used in the 50s and is called a perceptron. Linking several of those together created a multilayered perceptron and was able to make some basic decisions. (Nielsen, 2019) Every decision, however, was binary. Either yes or no, never partially true. Although neurons being either entirely on or off more closely represents how the neurons in the human brain work, it is not very efficient when wanting to make complicated decisions in computing. (Saaty, 2000; Sanderson, 2017a) To improve the results, some changes were made over the years. Later versions of the neuron models accepted non-binary input and output values.

Once using non-binary inputs, there are two major issues with this output value. Its range is anywhere from minus infinity to infinity and, secondly, it is still not able to solve non-linear problems very well. To solve these issues, an activation function may be used, which squeezes the output values into a smaller range of values and may add non-linearity depending on the used mathematical function. This will be discussed further in section 5.2.2.5.

A neuron that can learn – i.e., update its internal parameters (weights, biases, values of filters) to achieve better results – is generally a sigmoid neural. Small changes in perceptron models could cause the output to flip to the other binary output possibility, whereas the sigmoid neurons let small inputs create small changes in the output and get closer to a good model – it learns. (Nielsen, 2019) When there are no loops or other special characteristics in a neural network, it can be referred to as a feedforward neural network, pointing to the forward motion of the data flow, from input to output and the process of checking our data against the set weights and biases is called a forward pass.. (Amani & Soleimany, 2018)

A fully connected dense layer is also routinely used towards the end of the other models to reduce the number of neurons to the number of wanted output neurons. In classification problems, the number of neurons in the last (dense) layer corresponds to the number of classes.

5.2.2.2 Convolutional Neural Networks

Convolutional Neural Networks are commonly used for image classification but may also be used on sequential data such as texts. To understand CNNs, a step back must be taken to look at filters, the kind that can be found in any sophisticated image editing programs: Gaussian blur, filtering using median, sharpen, blur, maximum, and minimum. (Adobe Photoshop, 2021; *CNN: Convolutional Neural Networks Explained - Computerphile*, 2016) Each filter has its own kernel, a matrix with carefully decided values that results in the wanted result.

A convolution is a mathematical operation by which the kernel slides across the image and performs the dot product of the values of the kernel and the original image, and the resulting value gets saved into a convoluted image. The kernel then gets slid over by a number of pixels, the stride, where the operation is repeated. This occurs until the entire image has been convoluted. There are several ways to handle the edges, trivial methods being padding (surrounding the image with zeros) to keep the original image size or valid, not adding additional zeros and as such only using valid values. This second method may result in ignoring the edge pixels, in which case some information gets lost.

When classifying images, this is useful in 2 main ways: it removes fuzziness in the image by using operations like blurs, median (salt & pepper effect) and averaging. The next step uses gradients operations (Sobel, Robinson, Laplace etc.) to recognize specific patterns such as lines, corners and, later on, more complicated patterns. Each filter recognizes one such pattern, and therefore several filters are used in one convolutional layer. For colour images, 3D convolution filters are used, one two-dimensional filter for each colour channel. (Bailey et al., 2019; Ježek, 2021; Russo, 2019) It is vital to recognize that, in neural networks, the programmer only sets the number of filters and is not in charge of whether they want to use Sobel or Robinson.

The neural network updates the values of the filters, just like the weights and biases in the dense layer, to achieve as good a result as possible. To achieve an even better result, several convolutional layers are used in a row, called stacking.

Figure 5-2: Testing different gradient operations on Lena (Olisa, 2018)

Texts may also use convolutional layers to recognize different patterns, known as feature extraction, or to filter words. As an analogy, let's imagine that words such as "and", "you" and "the" may be considered to be worth less to us - as they are used in every age group - than less frequent words that may mean more in this context, such as "school", "work" and "pensioner" and therefore should be considered more strongly. Neural networks for images use Conv2D (greyscale) or Conv3D (colour images) layers and work in one dimension only, whereas ANNs for classifying text prefer Conv1D layers, where the one dimension is the order of the sequence.

5.2.2.3 Pooling layers

Another method to reduce the size of the data considerably is pooling. It is a way to down-sample the given data and remove dimensionality. It is often used in combination with convolutional layers or recurrent layers, following them, to reduce the precision. It may sound counter-intuitive, but a too precise convoluted image (e.g., a perfectly horizontal straight line of 1 pixel wide) would not recognize a slightly different feature (moved over, 2 pixels wide, rotated) as effectively.

In the pooling layer, we define two variables: the pooling size, which defines how much we want to reduce the image size and the stride, by how much we move the grid over each time. In Figure 5-3 an image of 6×6 pixels is summarized by a 2×2 grid and a stride of 2. The resulting image will be four times smaller (3×3 pixels). The orange square, representing the grid, contains four values. Depending on the pooling method, we will get a different output. Although others exists, we generally use max pooling or average pooling. MaxPooling takes the greatest value in the grid and copies it over, ignoring the other values. Average pooling equivalently takes the average of the four values inside the grid. See figures Figure 5-3: Max pooling and Figure 5-4: Average pooling.

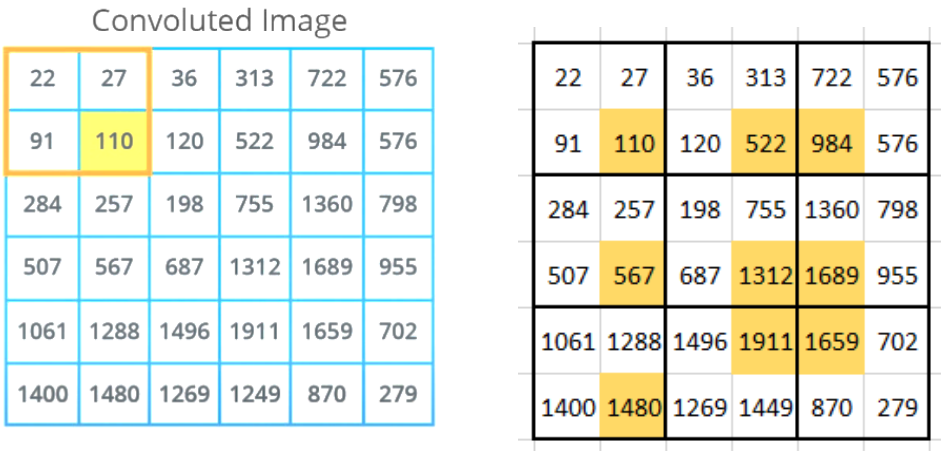


Figure 5-3: Max pooling

Left – Performing the pooling operation

Right - Selecting the max values

There is another type of pooling, known as global pooling, which, as the grid size uses the size of the input data. It is an aggressive way of summarizing data and may be used to, for instance, find the topic or mood of a text. It has an average and maximum version, GlobalAveragePooling and GlobalMaxPooling. (Brownlee, 2019; Russo, 2019) Global pools can, in addition to other things, be used in recurrent neural networks to get one output per sequence (word, n-gram or character) rather than one output value per feature.

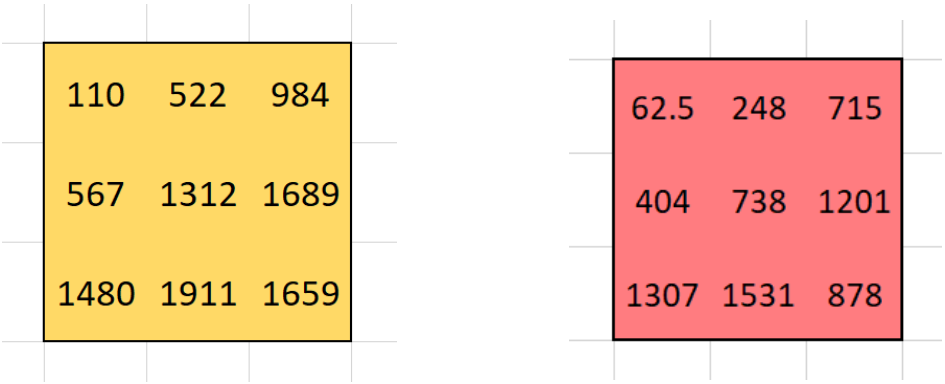


Figure 5-4: Average pooling

Left – Performing the pooling operation

Right - Calculating the average values

5.2.2.4 Recursive Neural Networks

Before getting into what recursive neural networks are, we will again take a step back and focus on the shortcomings of the models mentioned so far and then slowly get into how recursive neural networks resolve some of those problems.

Feedforward networks take each of the examples and analyse them separately from each other. This is a perfectly fine method for many problems, including evaluating the market price of one house does not depend on whether it is evaluated before or after a second house. The data points are standalone. Textual data, however, is a great example of sequential data, a form of data where the order matters. Other examples are the stock market, weather, location of moving objects and sound. The next prediction depends on their history.

Some methods exist, that could be used to give feedforward and convolutional neural networks a small peak at the history, but each of them has its flaws. One of the most seen examples is to use a fixed window –looking at a set number of tokens (words, characters, sounds, market values) before the one we are interested in. Deciding the size of such a window is problematic, as too small a window does not provide you with information that is further away (at the beginning of a sentence, seasonal fluctuations in the weather), whereas longer windows contain more information that may not in any way be relevant to the topic and the window has to be a fixed size for the full data set. Another problem with the fixed window size is

that parameters, such as weights and biases, are not shared. This means that a feature that may be recognised at one location, may not be recognised in a different location. This means that sentences meaning the exact same, only changing the word order, may not give the same output.

Because of this, a search for a more reliable method began, providing long-term dependencies (resolving short windows), shared parameters (feature recognised, no matter the location), variable length inputs and preservation of the sequence order. Most simply put, they needed a way to share the important information found in previous neurons to be forwarded to the next neurons, letting them take this into consideration too. And so the recursive neural networks were formed.

Figure 5-5 provides a typical representation of an RNN. On the left, it looks quite similar to a typical feedforward neural network, with the addition of a recursive arrow, which is where the name comes from. It represents taking the internal state, also known as the hidden state, of the neural network at time t-1 and putting it back into the network as an additional input to the usual x_t to produce output $\hat{y}_t$. Simply meaning that the output always depends on the current input, in our case word, and the previous internal state, which is some number that represents the previous words. The right side of Figure 5-5 shows the unrolled version of the RNN, where we more clearly can see the flow of the process, with the different inputs coming in, creating (temporary) outputs and forwarding some internal state onto the next calculation. Depending on the problem, we may be interested in all the outputs (text or music generation) or just the final output (word prediction, categorization). One more important thing to note is that the matrix of weights is the same for every time step. It does not get updated until we receive the final output, or more typically, the final output of the batch.

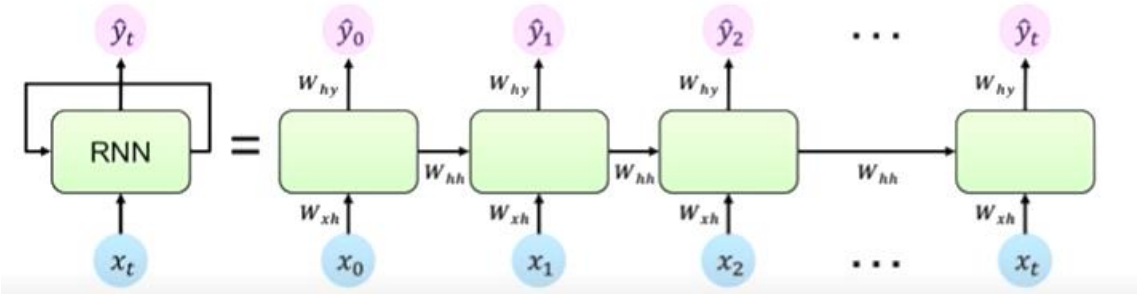


Figure 5-5: Recursive neural network, unrolled state (Amani & Soleimany, 2018)

The mathematics of RNNs is more complex and depends on the type of recursion cell that is selected. One of the more basic models for an RNN is shown in figure 5-6, and its calculations are still relatively similar to previously seen networks. For a timestep t, we have one new input. Rather than instantly using this input and the weights to create an output, we add one step in the middle: we update the internal state. This hidden state is based on the previous hidden state and the current input. We perform a version of a weighted average (multiply the hidden state and the input by their individual weights and add them) before putting the result through an activation function, for instance tanh. This state is saved to be used in the next step before calculating the output for this timestep.

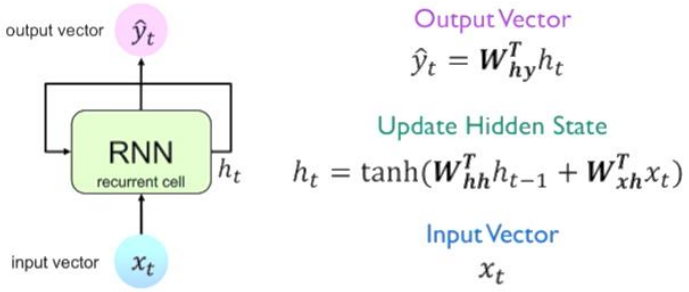


Figure 5-6: RNN calculations (Adapted from Sanderson, 2017b)

The two most commonly used forms of recursion are Long-Short Term Memory (LSTM) and Gated Recursive Units (GRU). They contain more complex computations that are generally chained, but the principle remains the same – the output depends on the input and hidden state. LSTMs are notorious for their ability to retain information for many timesteps while remaining useful for shorter-term predictions too. A typical LSTM cell representation can be seen in Figure 5-7.

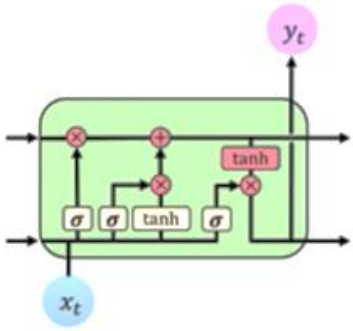


Figure 5-7: LSTM cell (Amani & Soleimany, 2018)

It uses an additional structure known as a gate, which selects what information should be kept and removed. To perform these actions it sigmoids (σ, keeping only a selection of the information, number between 0 and 1) and pointwise multiplications (×). The general process contains 4 steps. First, the LSTM cell forgets any irrelevant history. Then, relevant new information is then stored. Next, the internal state is then updated before finally calculating the output (using an output gate). (Amani & Soleimany, 2018; Lopez, 2019; Olah, 2015) As the LSTM cells are preprogrammed in Keras and only a few parameters can be changed, this level of mathematical understanding is sufficient at this point. (Keras, n.d.)

Gated Recurrent Units are in many ways similar to LSTMs – they also use gates to calculate the internal state, but additionally contain a forget gate and don't have an output gate the way an LSTM does. (Chung et al., 2014; Kostadinov, 2017; Lopez, 2019) A comparison of the internal state calculations of the different RNNs is shown in Figure 5-8.

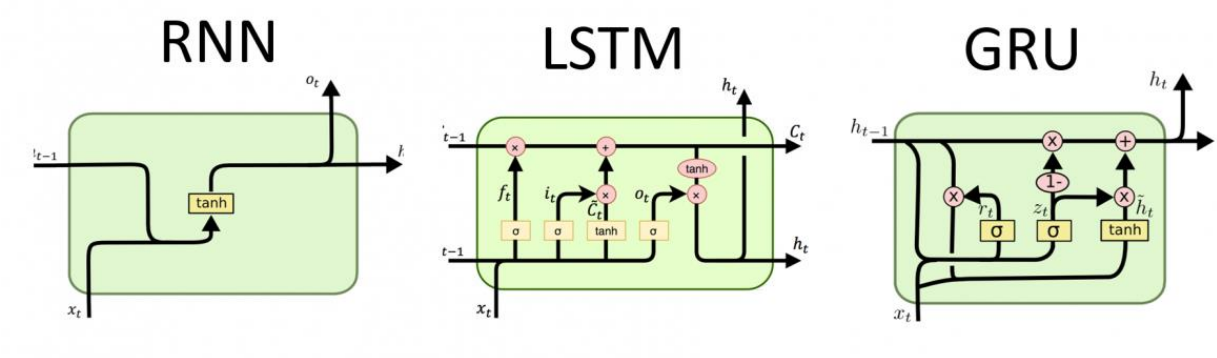


Figure 5-8: Internal calculation comparison between a basic RNN, LSTM and GRU (Lopez, 2019)

5.2.2.5 Activation functions

Activation functions are functions used to convert large values, generally the weights of the neurons, to a more standardized range. The activations are decided in the model creation stage.

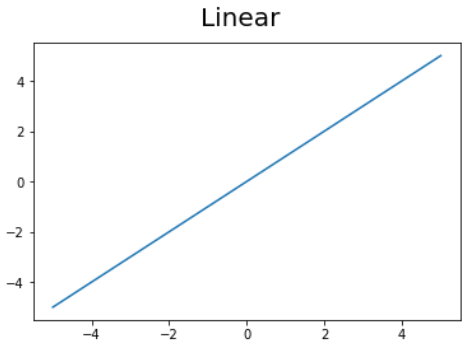


Figure 5-9: Linear plot

Linear is the most simple form and does not change the output. The output range is -∞ to +∞.

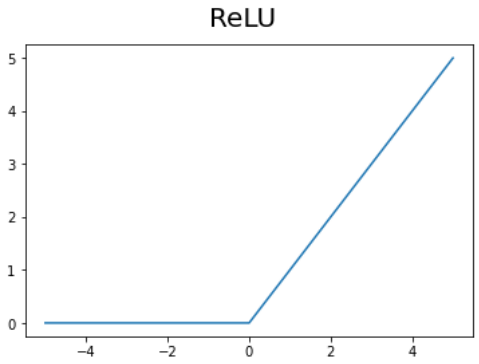


Figure 5-10: ReLU plot

ReLU, which stands for Rectified Linear Units, works as a linear activation for positive inputs but nullifies all negative inputs. It is quite popular and provides many of the same benefits that more complicated algorithms like Sigmoid and tanh provide and additionally solves the vanishing gradient problem. The range is 0 to +∞. There are versions of ReLu, such as ReLu6, where the upper range is also cut off. ReLU6's range is 0 to 6. (Brownlee, 2020)

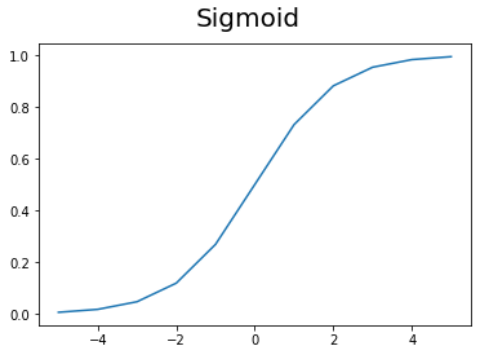


Figure 5-11: Sigmoid plot

Sigmoid, $S(x)=\frac{1}{1+e^{-x}}$, squeezes the output to a range of 0 to 1. It solves some problems like the range to infinity, but the network may learn very slow due to the shallow gradients. It also has a vanishing gradient, meaning it changes minimally towards the ends.

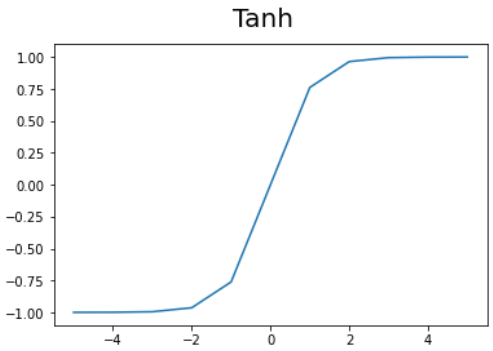


Figure 5-12: Hyperbolic tan plot

Tanh has steeper derivatives than Sigmoid and can resolve the slow learning problem. However, it only takes inputs between -1 and 1 and still has vanishing gradients.

Softmax is a mathematical algorithm used to convert the raw vector outputs into values between 0 and 1, the probabilities that the given example belongs to a given category. In neural network classification problems, it is used after the last dense layer.

5.2.3 Neuron learning and model improvement

Previously was mentioned that the neurons in the networks, no matter whether in a CNN, FNN or RNN, learn over time. They update their internal parameters to achieve better results, that is minimise the total error. However, the method of how they achieve this has not yet been spoken about. To simplify the thought, backpropagation in FNNs will be covered, and although mathematically somewhat more complicated, the same principle applies to the other models.

It is already known that the data flow in feed-forward networks is forwards – from the input to the output. Backpropagation is the opposite process, where instead the flow from the calculated output back to the input is considered.

5.2.3.1 Cost functions and loss

Cost functions, or loss functions, are the equations used to calculate the total error, called the loss in the predictions of the network and the chosen method depends on the type of the problem. Learning is the process of minimising the loss of a model.

The most basic form is a linear cost function, with MeanAbsoluteError likely being the best-known. Simply calculate the difference between the real and estimated values, take the absolute value and sum them all up before taking the average. In the case of multiclass categorical predictions, the correct category should have a true value of 1 and the rest 0. The main problem with this cost function is that 2 models, one with many small errors and a second with many even smaller errors but a few large errors would be considered equal.

To solve this, the MeanSquaredError can be used. Instead of taking the absolute value of the difference, we square the resulting value. This way bigger errors get penalized more and so, resolved faster.

Another way to look at the problem is to penalize less probable solutions. We can estimate the distribution of a random variable and calculate the entropy, a quite abstract concept describing the expected uncertainty. Keras provides different loss functions to minimize the entropy, depending on the kind of problem.

BinaryCrossentropy is used for binary classification problems – when the output is either 0 or 1 – and can be described as the negative average of the log of corrected predicted probabilities, where corrected probabilities simply means substituting the calculated probability with one minus that probability for all cases where the prediction was wrong, then giving us the prediction that the opposite is true. This correction is not that easy to make for larger datasets and this correction can be made directly in the calculation by accounting for both log(P) and log(1-P) in the initial calculation. Log of a probability (a value between 0 and 1) is always negative and is therefore countered with the additional minus.

This can be summarized in the cross entropy equation

$$\frac{1}{N}\sum_{i=1}^{N} -[y_i \times \log(p_i) + (1 - y_i) \times \log(1 - p_i)]$$

Where for binary cross entropy N=2.

For multiple class categorizations, it is widely accepted to use categorical cross-entropy, Softmax (Section 5.2.2.5) followed by cross-entropy loss, or sparse categorical cross-entropy, a combination between Sigmoid (Also in section 5.2.2.5) and cross-entropy loss. (Godoy, 2018; Gómez, 2018)

5.2.3.2 Optimizers and gradient descent

At this stage, the capability of the model is known – we know whether it is predicting the age category for the corpus accurately or not, and we can compare the outputs for different values of internal variables – and the improvement stage begins. Imagine a function relying on one input variable only. Given any point, calculating the slope at that point, a decision can be made on whether we should increase or decrease the given variable to decrease the cost.

The size of this increase or decrease in the parameters is called the step size, also known as the learning rate or simply alpha, pointing to its assigned mathematical symbol. It generally has a value between 0.001 and 0.1. This is another variable that can be experimented with, larger step sizes may overshoot and jump over the

minima, and too small a step size would take too long to get close to the minimum. Therefore, an adaptive learning rate may want to be considered, making the steps smaller the closer you get to the minimum. (Using the callback reduceLROnPlateau) Although we may only find a local minimum of the graph and not the global minimum, the found minimum can be seen as a low enough cost and a well enough model.

Gradient descent applies this same principle in a multivariable world, using multivariable calculus to calculate the gradient (∇), a vector of partial derivatives, which gives us the steepest increase, the calculation of which will be shown in the next section. Performing the opposite action, that is -∇ , gives us the steepest decrease and gives us the fastest way to the minimum.

Add the resulting gradient vector to the learnable parameters (the weights and biases), hopefully making their next overall output for that one example less wrong, and repeat for all the other examples. The magnitude of the vector values shows the importance of a suggested change, and some changes will therefore have larger effects than others, some only helping one example while destroying the progress as a whole. Rather than calculating the loss for each individual forward pass, known as online learning, it is typical to group the examples in so-called batches and instead use their average cost to use during the backpropagation. A bigger batch is more memory intense, but the internal variables are less likely to fluctuate, and the set is more likely to find a better optimization. When the gradients across all examples are averages, we retrieve a very exact version of gradient descent that is both time and memory intensive. Because of this, backpropagation is generally performed in smaller, randomised batches – the standard in Keras is 32 examples. The smaller, less accurate steps are used in an often used optimizer called stochastic gradient descent (SDG) that can be used on both classification and regression models. (Brownlee, 2021a; Sanderson, 2017a)

Another trivial optimizer is Adam, short for Adaptive Movement Estimation. It is an extension of SDG, using the first and second moments of the gradient. (Brownlee, 2021c; Kingma & Lei Ba, 2017)

5.2.3.3 Backpropagation

The algorithm to calculate the gradient in hidden neurons is called backpropagation and is based on the chain rule, a basic calculus principle. The more widespread version of the equations uses the Hadamard product or element-wise product, a matrix operation taking two matrices of dimension m×n and resulting in another matrix of the same size, where each index would be the multiplication of the elements in the same location, that is A ⊙ B = C, where $c_{ij} = a_{ij} \times b_{ij}$.

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(BP2)}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad \text{(BP4)}$$

Figure 5-13: Backpropagation equations (Nielsen, 2019)

The notation in Figure 5-13 is more generalised but less intuitive than the following example, Figure 5-14 toFigure 5-17 (Sanderson, 2017b). For each example, we get a predicted output and a true output. As mentioned before, we want to minimize this difference. Looking at each class within the example, we can note down how much we would like the output to change by. For example, the result should be that the text was written by someone in their 10s, that is y = [1, 0,0] and the prediction, $a^{(L)}$ = [0.5, 0.4, 0.1]. Although the prediction is correct, the resulting cost calculated using the formula in Figure 5-14, is quite large.

$$C_0 = \sum_{j=0}^{n_L - 1} (a_j^{(L)} - y_j)^2$$

Figure 5-14: The cost function of the example (Adapted from Sanderson, 2017b)

The wanted change in output is [+0.5, -0.4, -0.1]. These outputs cannot be changed directly, but the activations can be influenced by changes in the weights and biases in the previous layers.

Imagine if the previous layer only had one neuron, the activation value would be calculated by the activation function of the weights time the previous activation plus the bias. In general, there is more than one neuron in each layer and this would be rewritten as in Figure 5-15 and Figure 5-16.

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

Figure 5-15: The final layers activation depends on the sigmoid of z (Adapted from Sanderson, 2017b)

$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

Figure 5-16: z is the weighted sum of the weights, previous activations and the bias (Adapted from Sanderson, 2017b)

Finally, what we are interested in is how big a change to the previous layer needs to be made to create the wanted change in the cost. This can be calculated as the chain rule of the previous equations, resulting in relatively simple derivatives that need to be multiplied through and summed up, working our way back through all the layers until we reach the first. Hence the name backpropagation.

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}$$

Figure 5-17: Applying chain rule to find the wanted gradient (Adapted from Sanderson, 2017b)

5.2.3.4 Metrics

The resulting model should have a low loss, but loss alone is not enough to objectively decide whether a model is performing well or not. For this purpose, different metrics are used.

These metrics compare the actual (true) labels to the predicted labels. When the prediction is correct, this is called a true positive (TP) or true negative (TN), the second part referring to what was predicted. When the prediction is wrong, it is said to be a false positive (FP) or false negative (FN), the second part is, again, named after the prediction, not the true value. The relevant elements are the elements that we are trying to identify. A summary of this can be found in Table 5-1: Definitions of True and False Positives and Negatives.

		Predicted labels	
		Positive	**Negative**
Actual labels	**Positive**	TP	FN
	Negative	FP	TN

Table 5-1: Definitions of True and False Positives and Negatives

These four measures are then used to calculate the needed metrics, namely accuracy, precision, recall (also known as sensitivity) and f-measure. There are additional measures such as specificity and the negative predictive value but they are less conventional.

As an example, consider a dataset with 100 points, and 20 of them are articles written by people in their 10s. The model identifies 15 articles, out of which 10 are correct and 5 are wrong.

		Predicted labels	
		In 10s	**Not 10s**
Actual labels	**In 10s**	10	10
	Not 10s	5	75

Table 5-2: Example table of TP, FP, TN, FN. Revelant element is in 10s.

The accuracy is a measure of what percentage of examples were predicted correctly, that is $\frac{TP+TN}{TP+TN+FP+FN} \rightarrow \frac{10+10}{10+10+5+75} = 20\%$

Precision is a measure of relevance or completion. It looks at the ratio between all the labels' true positives and the selected items. $\frac{TP}{TP+FP} \rightarrow \frac{10}{10+15} = 40\%$

The recall is similar to precision but instead looks at the fraction of relevant elements retrieved, true positive over the relevant items. $\frac{TP}{TP+FN} \rightarrow \frac{10}{10+5} = 66\%$

The difference between recall and precision is clearly illustrated in figure 5-14. When increasing the recall, the precision decreases and vice versa. Selecting which metrics you want to follow depends on the problem. Another way to find a middle ground where both values are acceptable is by comparing the F1 score, or F-score for short. It is an average of the precision and recall values. $\frac{2\times Precision\times Recall}{Precision+Recall}$ $\rightarrow \frac{2\times 0.4\times 0.66}{0.4+0.66} = 60\%$

Scikit learn provides methods for all the above metrics, where generally the only required inputs are the true and predicted labels. (Scikit learn, 2019b, 2019a, 2019d; Wang, n.d.)

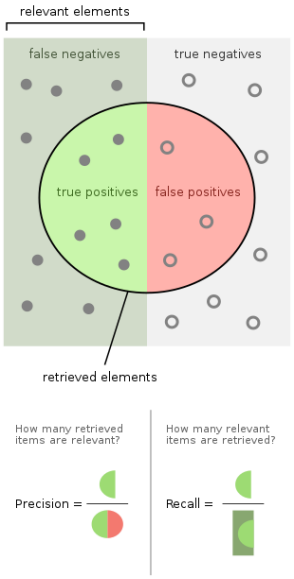


Figure 5-14: Precision and recall (Walber, 2014)

Once we have the predictions, the results are generally plotted in a confusion matrix. It can be seen as an extension of Table 5-1, where now all the options are shown. All the possible categories are plotted along both sides and the cells represent how many examples in that row got predicted as that column value. A perfect model has all the entries lined up in the main diagonal. It is a simple way to see confusions, where the model makes mistakes. For example, the model represented with the confusion matrix in Figure 5-15 often confuses the Versicolor for a Virginica and research should be done on why this is the case and whether a different model or even method isn't more suitable. The values in the cells can either represent the number of examples or percentages. Again, scikit learn has a method to create a simple confusion matrix and prints it as an array. For a visual output, it is often combined with the Seaborn heatmap. (Aruchamy, 2021; Seaborn, 2012)

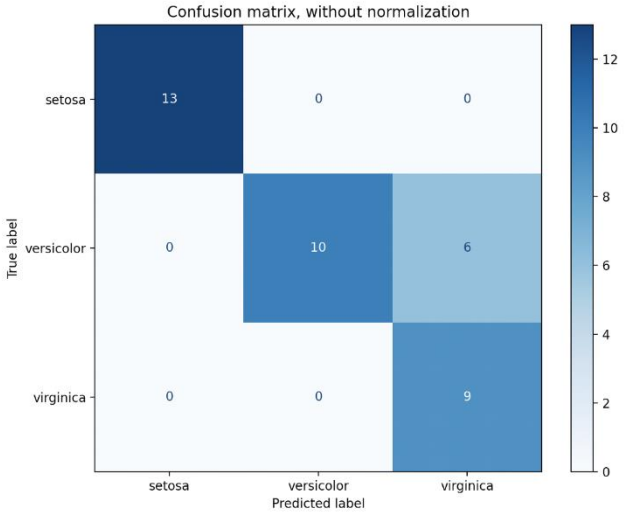


Figure 5-15: Confusion matrix comparing lilies, confuses Versicolor for Virginica. (Scikit learn, 2019c)

5.3 Naive Bayes

Metrics such as mentioned in the previous section can also be compared with other machine learning models. Naive Bayes is one of those. Bayes theorem is a well-known statistical model, which states $P(A|B) = \frac{P(B|A)\times P(A)}{P(B)}$. In classification problems, we are trying to calculate P(category|feature), for each of the categories. The feature is in this case a long text and therefore we split it into pieces, so P(B) =

b_1, b_2, …, b_n for each of the n tokens in the feature and calculate the probability for each of the tokens instead, that is:

$$P(A|b_1, b_2, \dots, b_n) = \frac{P(b_1|A)\times P(b_2|A)\times \dots \times P(b_n|A)\times P(A)}{P(b_1)\times P(b_2)\times \dots \times P(b_n)}$$

Instead of using the tokens directly, the textual input is generally preprocessed into TF-IDF format. It calculates how important a specific term is for one of the documents (d) of the corpus (D = Σd_i). More specifically, the documents can be thought of as a collection of all the blog post texts within one age category. TF stands for term frequency and simply is how often a word occurs relative to the whole document. TF(term, document) returns a percentage. The values for each term and each document is collected and represented as a matrix. The problem some words would have a very high TF, yet don't add much meaning, such as "you", "the" or "and" and would be found ubiquitously in all the documents. Inverse Document Frequency (IDF), resolves this issue by ruling out words that are contained in all documents. IDF(term, document) = log ($\frac{NumDocuments}{NumDocumentsWithTermT}$). Using the three categories – 10s, 20s and 30s – and the term *the* which would for sure be found in all of them, the IDF would be log(3/3) = log(1) = 0. Again, all those values would be saved in a matrix. The TF-IDF simply multiplies the TF and the IDF. So even when a term such as *the* may turn up 500 times in the category 10s, the TF-IDF(the, 10s) = 500 × 0 = 0. (Gandhi, 2018; Ritvikmath, 2020; Scikit learn, 2011c)

Naive Bayes additionally to Bayes theorem assumes strong independence between features, something which is not always true but gives a good approximation. It also treats all word orders the same. As mentioned before, two sentences can have the same vector of word counts, so-called bags of words, but have different meanings. A liked example is I'm good, not bad at all and I'm bad, not good at all. (Amani & Soleimany, 2018)

For document classification, multinomial naive Bayes is generally used - specialised for multinomial distributions, a generalization of binomial distributions - where an additional smoothing value (α) is used. There are cases where a token occurs in one document but not in the others. The count would result in 0 and the final TF-IDF

value would also be 0, even if the other words may occur often. The smoothing value is added to the original count to overcome this issue. Commonly α is set to one and is then called Laplace smoothing. (Scikit learn, 2011a, 2011b; Starmer, 2020)

5.4 Preprocessing data

In the previous sections, several potential issues with the data have been described. The solutions to these and some additional problems are described in this subchapter

There are many methods how we could preprocess and normalize our data. A different study highlights the growing concern of biased data and their biased results. Even something as simple as not balancing the length of the texts (padding) resulted in an unfair bias. (Dixon et al., 2018, p. 20) However, one should always proceed with caution and all of the mentioned methods may not be implemented on all data sets.

The Oxford dictionary states that there are approximately 170'000 words in use. To make our work easier, while working with natural language processing, we limit this to the most used words, the number depending on the variety in the texts (e.g., social media posts are more general than movie reviews), with 10'000 words not being a foreign number. (Russo, 2019)

To summarize the data preparation step, we remove variables for which we have too few data points, remove duplicates, perform a frequency distribution and decide the wanted vocabulary size based on this analysis. Then we consider whether the variables need to be changed in any way, for example, value mapping, where each word (or special meaning) gets mapped to a number.

5.4.1 Tokenization

Section 5.3 on Naive Bayes explained the idea behind why we cannot input the whole text at once. It has to be split. The splitting process is called tokenization, with each part of the text named a token, and there are three universally accepted types.

1) Word tokenization – Each word is a separate token. The text is split on spaces. The problem is that the whole corpus may contain many words, some not as relevant because of their scarcity or because they are misspelt. Because of this we often set a vocabulary size (S), and only the S most used words are included. Any other words are Out Of Vocabulary and are given an OOV character, typically [UNK], meaning unknown word. In this way, the existence of a word is retained, even if we don't know the meaning of this word.

2) Character – each character is a separate token. The OOV problem does not occur here as, although depending on the language, there are only around 20-30 possible tokens. Instead, character tokenized texts can struggle with comprehension, as it needs context to convey any meaning.

3) Subword tokenization – is between the word and character level. It splits the words into subwords, in ML more commonly called n-grams. The n-grams can be of fixed length, they can use stepping or not – skip a few characters – or use an algorithm such as Byte Pair Encoding (BPE), which can also be used for data compression, that makes a list of possible subword combinations and saves only the most used ones. The phrase *Subword tokenization* with n-gram length without skipping would result in [Sub, ubw, bwo, wor, ord, rd , d t, …]. With a skip of size 2, 2 characters would be skipped between each iteration: [Sub, wor, d t, oke, niz, …]. BPE does not make sense on such a small data sample, but expected tokens include [est, er, es …] and other word endings on top of the single characters [a, b, c, …]. (Khanna, 2021; Pai, 2020)

5.4.2 Words to vectors

There are several ways to turn words into numbers. The resulting conversion table is called a dictionary.

1) One-hot encoding - each word has a vector where the represented word is a 1, and the rest are 0s, resulting in a very large vector, for instance, 10'000 words long.

2) Label encoding, where each possible value gets given a numerical value, eg word = 5, book = 7. This may however capture relationships between words that have similar numbers, but are not closely related in meaning.

A sentence can be represented as a bag of words, saving the number of occurrences of each word in a vector. However, the word order is not accounted for, resulting in some problems with RNNs. This may be favourable to categorize longer texts, such as books or news articles, but in general, the individual texts are shorter, in which case padding would be preferred. Then a vector of label encodings is saved instead, with special characters for out of vocabulary, padding and the start of a text. Padding has no meaning and is therefore customarily given the value zero. It just fills the gap, giving us the needed input size. (Sethi, 2020)

5.4.3 Embedding

Even on the word level, it can be hard to relate words. Embedding is a method that compares the meanings of words and puts them closer to each other in the vector space. We can imagine that words such as happy and excited may have similar meanings and so strongly related vectors, whereas happy and sad are quite opposite and wouldn't. A feature learned on one word can transfer to other, close vectors. (Brownlee, 2021b; Johnson, 2021; Koehrsen, 2018; TensorFlow, 2017b)

These embedding spaces can be visualised using an embedding projector. (TensorFlow, 2017a, 2021)

One other way to use embeddings is to take a pre-trained embedding, from a large, state-of-the-art NLP NN and use it as a feature extractor on your text, before entering the model. This is a form of transfer learning, a concept that won't be described here, but could be explored further in future projects.

5.4.4 Stop words

When using word tokenization especially, once you have your vocabulary, you will notice that the most numerous words will be [UNK] and [], followed by quite meaningless words, along the lines of [how, I, do, the, a, for, …]. These words don't

add any real meaning and are therefore stopped before making the vocabulary. The Natural Language Toolkit (NLKT) provides methods, including nltk.corpus's stopwords, for this and you can quite easily import the stopwords for a given language and filter them out. (Bonaros, 2019; GeeksForGeeks, 2022; NLTK, 2001; Singh, 2019)

5.4.5 Punctuation, numbers, emails, URLs

Similar to the previous point, punctuation, numbers, email addresses, links and similar also don't convey as much information to the computer as it may to a human, at least not in the case of classifications or sentiment analysis. It can be removed using a regex or an imported punctuation list. (Bonaros, 2019; GeeksForGeeks, 2019)

5.4.6 Word stemming and lemmatization

Word stemming and lemmatization is also based on the idea of linking similar words together, but in a different way. One base of a word can turn into many different versions, inflexions, simply by changing the tense or taking making a noun our of the given verb. Stemming and lemmatization reduce the number of words used to describe the whole set of similar words.

Stemming, as the name suggests, uses the stem of the word. It removes everyday endings like -ing, -'s and -es. It can be performed with simpler algorithms but may result in tokens that are not actual words. Lemmatization does this more properly and returns a correct dictionary form, but it is harder to implement. It is useful for some cases as it can link irregular words – e.g. good, better, best – and is better at distinguishing between homonyms. For example, *having* turns into *hav* after stemming and *have* after lemmatization. Lemmatization and stemming return many of the same tokens, but generally there are some differences across the entire corpus. (Beri, 2020; Manning et al., 2008)

5.5 Hardware and software

5.5.1 Python and its libraries

Python is widely used in machine learning (Argamon et al., 2007b; Hinds, 2018; Huang & Paul, 2019; Peersman et al., 2011) because of the many useful libraries and frameworks. NumPy is one of the main libraries on which the others build. It is a mathematical library allowing for complicated vector and matrix calculations, building on C, a lower level compiled language, and in so doing speeding up the calculations performed compared to when run in pure Python, an interpreted language, which requires the code to be compiled first.

When working with machine learning, and neural networks in particular, in addition to NumPy, the following libraries are often used:

- MatPlotLib (Hunter, 2003) – a library to plot different graphs. Works well with NumPy data.

- Pandas (McKinney, 2008) – provides data structures such as data frames to allow for easier data manipulation and provides some visual guidance in the data analysis.

- Scikit-learn (Cournapeau, 2007) – sklearn for short, provides various machine learning algorithms including naive Bayes (section 5.3), the mentioned metrics (section 5.2.3.4), a basic confusion matrix and more.

- Seaborn (Waskom, 2012) – Based on MatPlotLib, provides methods for data visualisation and is often used together with Scikit learn's confusion matrix to give a visually more pleasing and understandable output.

- NLTK (Team NLTK, 2001) – Natural Language ToolKit – provides many of the features mentioned in section 5.4 and other handy tools for natural language processing.

- TensorFlow (Google Brain, 2015) – is focused on deep neural networks. It links in closely with Keras. Tensorflow additionally also provides different

premade datasets and TensorBoard – a tool to visualise the different experiments and their metrics. The name TensorFlow comes from tensor, a multidimensional vector, and flow, pointing to the data flowing from one neuron to another. (Fleisch, 2011)

- Keras (Chollet, 2015) – can be seen as an interface for TensorFlow. It provides, among others, the layers, optimizers, callbacks, methods to save and reload your models so you can use the learned model elsewhere, and the functional and sequential APIs.

5.5.2 Hardware

Machine learning code can be run on a Central Processing Unit (CPU), but for many more data-intense projects, the work can be parallelised and instead run on a GPU (Graphical Processing Unit) or TPU (Tensor Processing Unit).

CPUs were built to handle many, generally long, system operations at once, whereas GPUs perform and return fast mathematical calculations to then be rendered. Rendering operations are very similar to the tensor operations needed for neural networks. CPUs are generally good with calculations, but they only have 1, 2 (dual) or 4 (quad) cores. GPUs have many more cores, all containing Arithmetic Logic Units (ALUs), which control the multiple threads on each core. They can massively parallelise the calculations. For small data sets, however, using a CPU can be faster as it generally has larger cache memory and can store more information at once. It may simply be able to load and process more data at once.

Google Colab provides GPU farms, where you can use their GPU computing power through the cloud.

Tensor Processing Units are another step up and are processing units created by Google especially to be used with neural networks and TensorFlow. They are optimised not just for tensor operations like GPUs, but for larger tensor operations. They have separate parts for matrix multiplications and processing vectors, each performing these tasks very effectively. TPUs can be 20× faster than state-of-the-art GPUs, or as Google states "*Models that previously took weeks to train on other*

hardware platforms can converge in hours on TPUs." (Google Cloud, 2016) However, this is not true for small batches, and additionally, it struggles with high precision arithmetic and frequent branching.

A common factor for all these calculations is that they are generally performed in multiples of 8, coming from the idea that a byte contains 8 bits which flowed over into many other regions of computing.

6 Implementation

This chapter will in detail describe the creation and comparison of the aforementioned models. Figure 6-1 gives a brief outline of the performed steps.

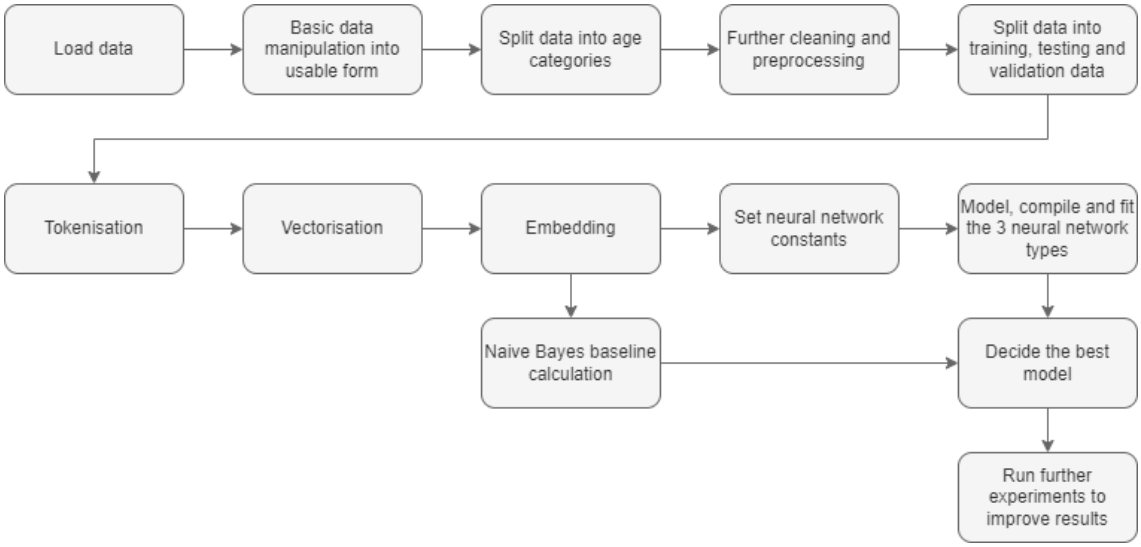


Figure 6-1: Workflow outline

6.1 Calculating the baseline

As described in Chapter 3: Related research, the results in different machine learning experiments vary widely, reaching 43-95% accuracy. Deep neural networks did the best out of all the neural network models achieving a 94%. This can, given the limited data, be considered our state-of-the-art result which I am unlikely to beat. Anything above 43% seems significant enough to publish, likely as it beat the statistical expected value of 33% when guessing wildly when using 3 categories. My initial aim was to beat 60% accuracy with similar values in the recall, precision and F1, but as all those values depend a lot on the data set and we don't have ground-truth data for the user ages - the "true" age in the dataset is what the user provided on their blog and could be false - it seems fairer to compare it to a different prediction algorithm.

I decided to use naive Bayes for the baseline calculation, as it is generally fast to calculate and it is similar to Goswami's team's method based on word frequency. It uses TF-IDF, further described in the theoretical section on naive Bayes (section

5.3), to convert the text to numbers, and then we model it using multinomial naive Bayes. This is most easily done with a simple two-step pipeline and the Sklearn library. (Neagoie & Bourke, 2020)

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline

baseline_model = Pipeline([
                           ("tf-idf", TfidfVectorizer()),
                           ("clf", MultinomialNB())
])

baseline_model.fit(train_text, train_labels)

baseline_predictions = baseline_model.predict(train_text)

calculate_result_metrics(y_true=train_labels,
y_pred=baseline_predictions)
```

Code sample 6-1: Baseline calculation – Naive Bayes

Once the baseline model is created, it is fitted to the training text and predictions are made. These predictions are then compared to the true age categories and the following metrics are returned:

Accuracy: 0.6424748782350782
Precision: 0.7570033463974328
F1: 0.5815958976728455
Recall: 0.6424748782350782

So 64% will be the accuracy to beat.

6.2 Data set selection

It is imperative to consider the data source. *"The quality of the data is the most important factor to influence the results from any analysis".* (Myatt, 2007) Selecting a data set turned out to be a harder process than expected, largely due to ethics and the need for the legal obtainment of the data. Data mining is generally considered a grey area and you have to go through special APIs to mine. This is true for many social media, including Twitter. Not everyone can get access to their endpoints and this turned out to be a too time-consuming option. Secondly, a lot of the data sets

used in other research are not published, as the users only gave consent to the researchers using their data. The results of their experiments can be published under the group privacy clause, but the actual personal data can not. This left me with only a small selection of possible data sources.

Most of the leftover corpora were from PAN, a set of conferences focused on digital text forensics and stylometry. Although many of their data sets are used for author identification, only two of the ones I managed to get access to were related to age. Even then, they did not have the text entries saved directly but instead contained the link to Twitter where the text could be found. Unfortunately, many of the links were broken and again, it involved data mining Twitter.

Finally, I noticed that several author profiling articles cited the same researchers. Looking them up I found several interesting articles and data sources, including the dataset I finally decided to settle on: *Effect of Age and Gender on Blogging - Blog authorship corpus.* (Argamon et al., 2007b)

On their website it is described as follows:

The Blog Authorship Corpus consists of the collected posts of 19,320 bloggers gathered from blogger.com in August 2004. The corpus incorporates a total of 681,288 posts and over 140 million words - or approximately 35 posts and 7250 words per person.

Analysis of a corpus of tens of thousands of blogs – incorporating close to 300 million words – indicates significant differences in writing style and content between male and female bloggers as well as among authors of different ages. Such differences can be exploited to determine an unknown author's age and gender on the basis of a blog's vocabulary. (Argamon et al., 2007b)

The selected data set is relatively unique, not only because of its massivity but also in the number of additional features it follows and it can be checked whether wrong predictions aren't related to a more specific group of users, e.g., Aquarius male students in their 20s get predicted wrong rather than just people in their 20s.

Looking at the data results from other similar studies, it was suggested that multiclass prediction with 3 or more classes could be less accurate than using regression to get a numerical value. However, as this dataset contains preexisting gaps, it would not make sense to use such a method as regression does not like empty data points and we would have to make educated guesses to fill in the gaps, likely resulting in worse results in the end, although further research could be performed to check this.

6.3 Data preparation and manipulation

6.3.1 Loading the data set

Before being able to handle the data, first, the data had to be loaded, which is where the first problem occurred. The given corpus is immense, an extraordinary ~300MB. Initially, I loaded the data using the code in figure X, where *blogtest_0.csv* was a pre-split subset of blogtext.csv (Schler et al., 2005) saved locally as the file itself was too large to load this way. It was also too large to upload to Github or similar platforms.

```
fileToLoad = files.upload()
df = pd.read_csv(io.BytesIO(fileToLoad['blogtext_0.csv']))
```

Code sample 6-2: Loading partial dataset using io

I found the same dataset uploaded to Kaggle (Tatman, 2017), a page widely used for machine learning, partially as it provides easy ways to load the datasets fast, no matter their size. Following the instructions found on the blog Analytics Vidhya (Gupta, 2019), I can now load the whole data set in seconds rather than 20 minutes per run for only a subset.

The new method instead uses a personal Kaggle file which can be used to load any Kaggle dataset. Afterwards, it unzips the dataset and turns it into a pandas data frame – a table format.

```
! kaggle datasets download rtatman/blog-authorship-corpus
! unzip blog-authorship-corpus.zip
df = pd.read_csv("blogtext.csv")
```

Code sample 6-3: Importing the Kaggle dataset and turning it into a data frame

6.3.2 Initial data analysis and data preprocessing.

	id	*gender*	*age*	*topic*	*sign*	*date*	*text*
0	2059027	male	15	Student	Leo	14,May, 2004	Info has been found (+/- 100 pages…)
1	2059027	male	15	Student	Leo	13,May, 2004	These are the team members: Drewe…
2	2059027	male	15	Student	Leo	12,May, 2004	In het kader van kernfusie op aarde…
3	2059027	male	15	Student	Leo	12,May, 2004	testing!!! testing!!!
4	3581210	male	33	Investment Banking	Aquarius	11,June, 2004	Thanks to Yahoo!'s Toolbar I can …

Table 6-1: The first 5 results in the data frame, shown using df.head()

The initial data analyses, such as shown in Figure 6-1 and Figure 6-2, are important to ensure that the data is split fair and confirm there is no major bias in one direction. Even if the other features will not be used directly - as the additional metadata could impair the results - controlling them increases the reliability of our final results. We can also check the data types in each column and look at some data examples.

id	19320
gender	2
age	26
topic	40
sign	12
date	2616
text	611652
dtype: int64	

Table 6-2: Unique elements in each column, extracted using df.nunique()

Furthermore, `df.shape` can be used to find the number of data entries and features, here returning *(681284, 7)* and the method `df.nunique()` gives us the number of unique values in the data frame. The number of unique texts comes out as *611'652*

whereas there are *681'284* rows in the dataset, meaning there are likely duplicates which we will have to remove. This can be done with `df.drop_duplicates()`

It is clear that the date is in a too detailed format and will be converted into the year only before graphing it. Code sample 6-4 takes the date as a string and keeps only the last 4 values, such as shown in Code sample 6-5.

```
df_no_duplicates["date"] = df_no_duplicates["date"].apply(lambda x: x[-4:])
```

Code sample 6-4: Simplifying the date, returning only the year

```
test_string = "11,June,2004"
test_string[-4:] //returns '2004'
```

Code sample 6-5: Example of date simplification.

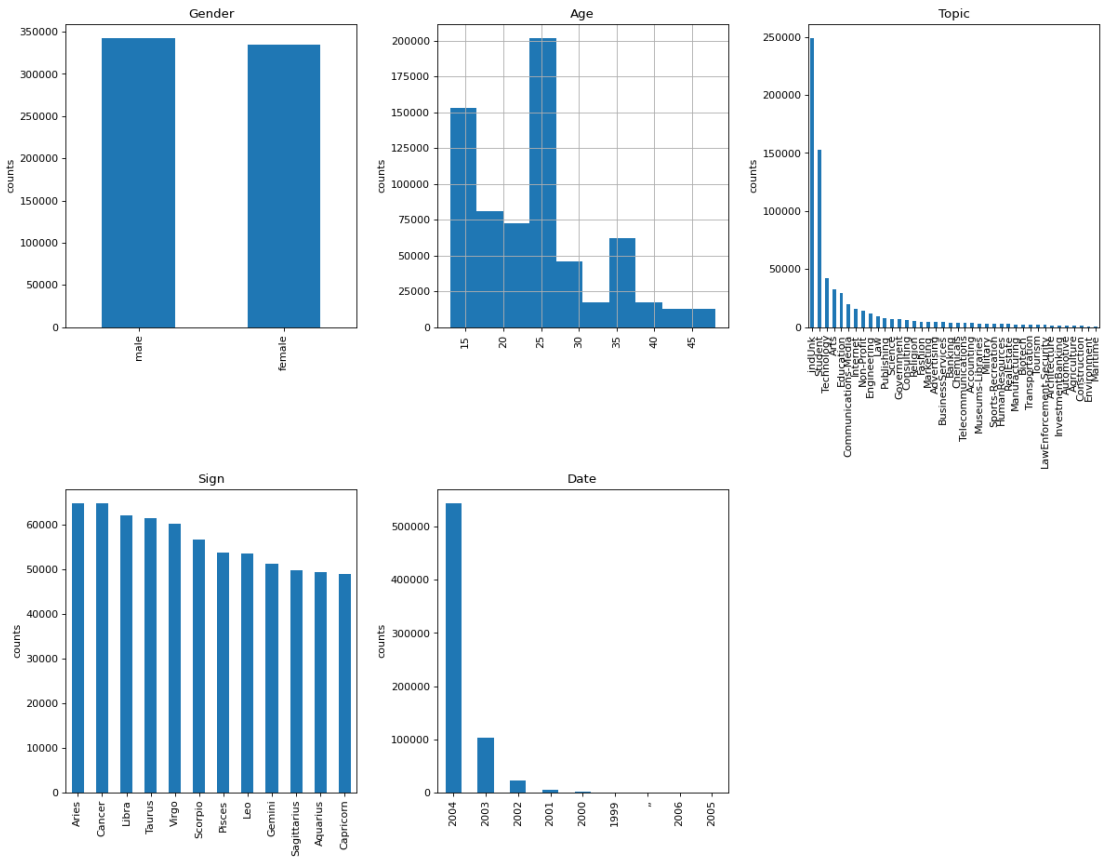


Figure 6-2: The results of plotPerColumnDistribution(df_plot, 5,3), showing the distributions of the users' and their texts' attributes

There are only 5 columns that are worth graphing as the others contain too many unique values. The valuable features are *gender, age, topic, date and sign.* Additionally, it is worth looking at the text length, which will be shown later.

As seen in Figure 6-2:

- The gender distribution is pretty even. No need to interfere
- The age distribution is poor and should be grouped
- Most of the topics are unknown. Only "student" really stands out, which will likely be well enough represented by the age. This could be checked with a correlation test, but considering that the vast majority of topics are unknown, it will be better to ignore this feature altogether. This column will be dropped.
- The signs are pretty well balanced. Analysing astrological signs, that is astrology, is a pseudoscience (Carlson, 1985) and therefore, I do not consider the sign to be a relevant feature and the column will be dropped. It could however be relevant if we wanted to know their exact birthdays.
- Almost all entries were written in 2004, with a few entries in the year before. We may assume that there will no major differences in language. The year column will be dropped.

	id	*gender*	*age*	*text*
0	2059027	male	15	Info has been found (+/- 100 pages...)
1	2059027	male	15	These are the team members: Drewe...
2	2059027	male	15	In het kader van kernfusie op aarde...
3	2059027	male	15	testing!!! testing!!!
4	3581210	male	33	Thanks to Yahoo!'s Toolbar I can ...

Table 6-3: The data frame after dropping the irrelevant columns

```
df_preprocessing = df_no_duplicates.copy()
df_preprocessing.drop(columns=["topic", "sign", "date"], inplace =
True)
```

Code sample 6-6: Removing duplicate entries

6.3.3 Age categories

The data has been pre-split into 3 categories:

- Age 14-17, the "10s". They will get label 0
- Age 23-27, the "20s". They will get label 1
- Age 33-39, the "30s". They will get label 2

As seen in Code sample 6-7, I locate the correct rows with three if statements and assign the label value to a new column named age_category. The result of this can be seen in Table 6-4, showing the output of `df_preprocessing[[“age“, “age_category“]]`.

```
df_preprocessing.loc[df_preprocessing["age"] < 20 , "age_category"] = 0
df_preprocessing.loc[(df_preprocessing["age"] > 20) &
(df_preprocessing["age"] < 30)  , "age_category"] = 1
df_preprocessing.loc[df_preprocessing["age"] >30 , "age_category"] = 2
```

Code sample 6-7: Assigning age categories

	age	*age_category*
0	15	0
1	15	0
2	15	0
3	15	0
4	33	2
...	...	...
681281	23	1
681282	23	1
681283	23	1

Table 6-4: Adding the age_category label for each entry, comparing it to the age

After assigning the labels, the split is a lot more even as seen in Figure 6-3. As expected, following the line of thought in the research articles, there are more teenagers online than adults. This is also why most studies were capped around the age of 40. They could not get the needed number of entries for the older age categories.

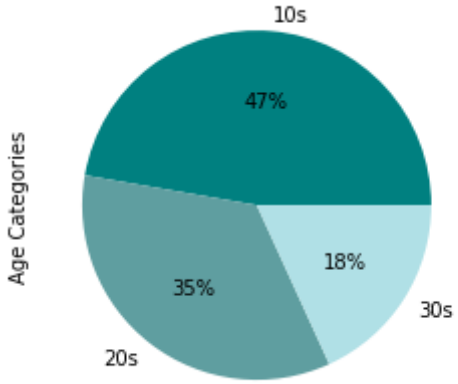


Figure 6-3: Pie chart of age categories in the selected dataset

6.3.4 Further preprocessing

As discussed in 5.4: Preprocessing data, the cleaning of the data is an important part of the machine learning process. It can be argued that good data with a slightly worse model can still achieve better results than worse data with a slightly better model.

```
df_text['text'] = df_text['text'].str.lower()
```

Code sample 6-8: Lowercase

```
df_text['text'] = df_text['text'].apply(lambda x: re.sub('(([a-z]+(\'|-
)[a-z]+)|[a-z]*)(.+?)', '\\1 ', x ))
```

Code sample 6-9: Regex for removing special characters

```
df_text['text'].apply(lambda x: re.sub('urllink', ' ', x.strip()))
df_text['text'] = df_text['text'].apply(lambda x: re.sub('nbsp', ' ',
x))
```

Code sample 6-10: Removing hyperlinks

```
df_text['text'] = df_text['text'].apply(lambda x: re.sub(' +', ' ', x))
df_text['text'] = df_text['text'].apply(lambda x: x.strip())
```

Code sample 6-11: Removing leading, trailing and multiple spaces

I decided to perform several cleaning methods, namely turning everything into lower case, removing any non-text characters, including hyperlinks and non-breaking spaces, while keeping apostrophes in words so words like *what's, it's, and*

they're are not affected. Removing the hyperlinks was relatively simple as they have previously been replaced with the word "urllink" and no regular expression was needed. Similarly for the nbsp (non-breaking space) entity. And finally, strip trailing and leading whitespaces and reduce multiple spaces to one. (Code sample 6-8 to Code sample 6-11)

0	info has been found pages and mb of pdf files ...
1	These are the team members drewes van der laag...
2	in het kader van het kernfusie op aarde maak je ei...
3	testing testing
4	Thanks to yahoo s toolbar i can not capture th...

Table 6-5: The resulting cleaned texts

6.3.5 Removing short entries

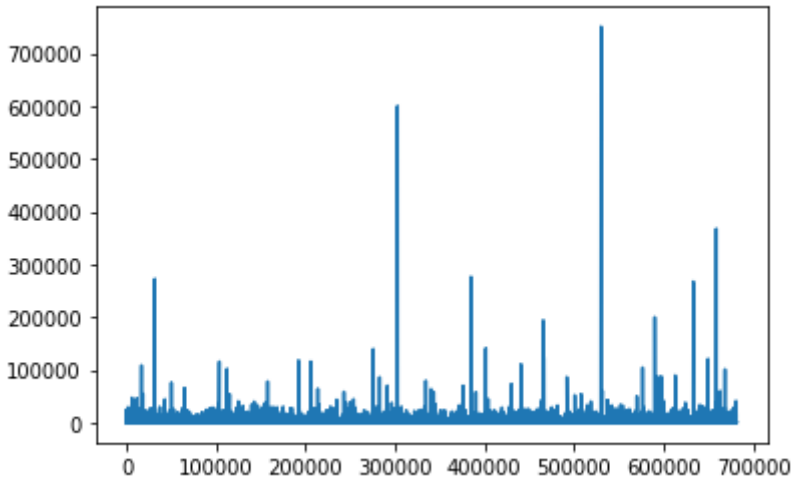


Figure 6-4: Entry length, x= text length, y = occurences

After calculating the median length of a blog post and the standard deviation, any extreme outliers can be removed or edited if need be. After looking at some of the entries, it seemed like a good idea to remove the clear "testing" and "spam" entries before, so as to not influence the median too much. The longer entries can be trimmed using the found sizes. After some experimentation with the constant `CUT_OFF_LENGTH`, the length of the shortest entries to be excluded, the ideal value seemed to be 40, removing 11757 entries, under 2% of the entire dataset. Higher values started to contain entries with more semantic meaning.

Text length attributes:
Minimum: 0
Maximum: 750735
Median: 580.0
Mean: 1036.9508940316111
Standard deviation: 2194.781090618009
Minimum extreme outlier: -10393.905453090045
Minimum outlier: -6004.343271854026
Maximum outlier: 7164.343271854026
Maximum extreme outlier: 11553.905453090045

Some are empty entries, such as entry 150 seen in Table 6-6, which don't add any value to the experiment. Other entries that are cut when using a cut-off length of 40 are equally irrelevant.

3	testing testing
150	(empty entry)
235	i drunk myself with win
237	one chance
240	wounded soul
...	...
681214	die and choke on jason cucumber
681245	peace
681254	dear susan i hope you die patrick
681257	dear susan why do you hate black people
681265	susan i don't like you haha

Table 6-6: Data entries with a text length less than or equal to 40

Comparing this to Table 6-7, where entries between a length of 40 and 60 are shown, they already contain more content that could be used to create a profile on the author.

138	we are drinking and talking of unrequited love n
149	yeah so i've been thinking a bit but not typing n
181	what sort of weird are you this quiz by orsa
238	it was her face which etched clearly in my mind
312	argh arghhhhhhh no time to look for one either
…	…
681135	ah kin hahaz listening to mp har so suang
681141	still not feeling oky i'm losing my voice
681161	dear susan you a fat wombat i hate you tha…
681173	dear susan you were an accident and a mistake …
681241	dear susan you just aren't worth it anymo…

Table 6-7: Data entries with a text length between 40 and 60 inclusive.

The entries cut are more or less the same as the spread of the whole data set. Rechecking all the graphs showing the different distributions, it is shown that removing short entries (≤40 characters) did not affect the age distribution of the corpus nor any major effect on any of the distributions in general. This means that the short message length is not an important representative point of this dataset and they can safely be removed.

There are no minimum outliers, but there are numerous extreme maximum outliers. (With lengths 102 268, 104 352, 108 633, 110 774, 115 677, 115 709, … being significantly larger than the extreme outlier limit of 11 554) They will be left for now as these entries do contain valid and useful information and I don't want to remove them in full. Instead, this will be resolved in a later step.

6.3.6 Splitting data into training, testing and validation sets.

I decided to use the classic 7:2:1 split using Sklearn's train_test_split. The difference here was the first line, where the indices had to be reset due to all the dropped entries.

```
df_preprocessed = df_preprocessed[num_train_examples:].sample(frac=1,
random_state = SEED).reset_index(drop = True)

train_text, test_text, train_labels, test_labels =
train_test_split(df_preprocessed["text"][num_train_examples:].to_numpy(
),
df_preprocessed["age_category"][num_train_examples:].to_numpy(),

test_size = 0.20, random_state = SEED ))
```

Code sample 6-12: Train: test split of data using scikit-learn's train_test_split method

	Data entries	*Percentage*
Training set	403 236	71.11%
Testing set	100 809	17.78%
Validation set	63 006	11.11%

Table 6-8: The data split

6.3.7 Tokenisation, vectorisation and embedding

As covered in section 5.4.1, tokenisation is used to split the given text into smaller pieces on a word, character or subword level.

Although a character level tokenisation would result in a lot lower vocabulary size (26 if the text only contains English characters), it would be harder for the neural network to find connections between the letters. This does not necessarily need to be a problem for deep neural networks, but as I use relatively basic models, I decided to instead use word-sized tokenisation with an out-of-vocabulary character [UNK] replacing the least numerous words. The third option is subword level tokenisation, based on n-grams, where n is the length of each part. Although the idea is interesting, I would have to iterate over the entire corpus several times to find the best character sequences. As the text is very large, this doesn't seem feasible to perform on an everyday use laptop.

So finally I settled on tokenisation on a word level with a vocabulary size of 50 000 words. I do not know the original vocabulary length, as the program crashes when the vocabulary size is set to None (unlimited, include all), but around 50 000, the words started to become uncommon - misspelt, not readily used in a everyday life or even strange. Some of the last included words in the vocabulary include:

['ganga','galz','gaby','futsal','furlong','funs','funnn','freshfaced','freeform','freakshow', 'fortify','forster','flinders','flattening']

Anything being less common than the above words, likely won't add much value to the written text. Further experiments to reduce the vocabulary size could be performed. Comparing that to the most frequent words in the vocabulary:

['','[UNK]','the','i','to','and','a','of','that','in','it','my','is','you','for','was','on','me','but']

Unsurprisingly, empty characters and the sum of all other words not included in the vocabulary, the OOV character [UNK], are at the top followed by typical personal pronouns and conjunctions.

In the middle of the vocabulary, we find everyday words like:

['guys', 'girl', 'gonna', 'used', 'family', 'thinking', 'live', 'hate']

Although words at the top of the vocabulary are important in the sense that they are used often, we would expect them across all age groups and they don't add much value in that sense. Because of this, it can be worth considering filtering stop words - words that are filtered out before the actual natural language processing and before creating the vocabulary and it is often recommended to remove them for classification problems.

Probably the most widespread way to remove stopwords is by using NLTK. As removing stopwords decreases the corpus size, the overall processing time tends to be lower. On the other side, wanting to remove stop words would result in having to use NLTK and iterating over the text manually, rather than some other currently faster methods.

Therefore I will keep the stop words for now and instead experiment with it in the end, once I have found the most effective neural network model for our data set and use a different, quite effective approach for now.

Only recently out of keras.experimental is the TextVectorization layer. It allows us to take any text and it will tokenise and vectorises it, outputting an array of numbers.

Padding the too short entries. The output length is based on the average text length and the average word length, so handling of any (extreme) upper outliers.

The final preparation step before the actual machine learning algorithmic application is embedding the text. It finds correlations between different words and represents these similarities and differences as vectors.

For example, the following text found in the data set:

tomorrow i will be wearing my orange pants and red shoes i will be totally stylin haha yes today was pretty lame again jeff showed meghan and me his skate video thing and i told matt that i want to marry jon it was funny my bus didnt come till it was painfully cold outside too but while we were waiting me and natalie and katarina were watching the band and tom waved at us hurray for nice drummers

```
text_vectorizer = layers.TextVectorization(max_tokens = 50000,
output_sequence_length = OUTPUT_LENGTH)
```

Code sample 6-13: Preparing text vectoriser

After vectorising using the Code sample 6-13, it turns into the following:

```
<tf.Tensor: shape=(1, 207), dtype=int64, numpy=
array([[  280,     3,    50,    23,   892,    11,  1817,  1210,     5,
          571,  1092,     3,    50,    23,   572, 32841,   358,   233,
           97,    15,   183,  1861,   132,  1921,  1173,  9762,     5,
           17,    59,  5279,   855,   128,     5,     3,   211,  1190,
            8,     3,    90,     4,  2434,  2190,    10,    15,   323,
           11,   781,   131,   150,   640,    10,    15,  8086,   626,
          497,    96,    18,   147,    24,    69,   539,    17,     5,
         5274,     5, 34349,    69,   440,     2,   537,     5,  1399,
         7295,    25,   119, 10967,    14,   208, 23020,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
                                        ...
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0]]
)>
```

```
embedding_layer = layers.Embedding(input_dim = vocab_length, output_dim
= 256,)
embedding_layer(text_vectorizer([sample_text]))
```

Code sample 6-14: Preparing embedder, running vectorization and embedding on sample_text

And finally, it gets embedded using Code sample 6-14, returning:

```
<tf.Tensor:   shape=(1,   207,   256),   dtype=float32,   numpy=
array([[[ 0.02513913,  0.03178935, -0.0057463 , ..., -0.04402475,
                                    0.00963997,      -0.03580179],
        [ 0.02286556,  0.01752985, -0.01490762, ...,  0.01668891,
                                    0.03585861,      -0.01533937],
        [-0.02062292,  0.02874741, -0.01069456, ..., -0.02900656,
                                    0.01664155,       0.02478272],
                                                                ...,
        [ 0.03735739, -0.03139231, -0.02102159, ...,  0.04548396,
                                   -0.00557458,       -0.02985016],
        [ 0.04815376, -0.016858  ,  0.02365566, ...,  0.04610132,
                                    0.04403371,      -0.00902497],
        [ 0.0184358 ,  0.01818255,  0.04221144, ..., -0.00287368,
         -0.01065179, -0.00709324]]], dtype=float32)>
```

6.4 Building the neural network models

Neural networks are always built using the same setup:

1) Make a model using different layers, deciding the structure.
2) Fit the model to training data, optionally perform cross-validation, that is validate the results using validation data
3) Make predictions using testing data
4) Analyse the results
5) Make changes and repeat

6.4.1 Feed-forward neural network

Feed-forward neural networks are commonly built using the sequential API, such as the examples shown in code samples Code sample 6-15 and Code sample 6-16, as it is a straightforward model with all the layers being processed in sequence. However, as recurrent models can't be built using this, I decided to use the functional API for all three, making the differences clearer.

```
model = tf.keras.models.Sequential([
tf.keras.layers.Dense(64, activation = "relu"),
tf.keras.layers.Dense(3)
])
```

Code sample 6-15: Sequential API example 1

```
model = tf.keras.models.Sequential
model.add(tf.keras.layers.Dense(64, activation = "relu"))
model.add(tf.keras.layers.Dense(3))
```

Code sample 6-16: Sequential API example 2

Functional APIs on the other hand, allow for a non-linear topology. The difference will be shown in the graphs plotting the models later on. (*The Functional API*, n.d.)

```
inputs = layers.Input(shape=(1), dtype = tf.string)
x = text_vectorizer(inputs)
x = embedding_layer(x)
x = layers.GlobalMaxPool1D()(x)
outputs = layers.Dense(3, activation = "softmax")(x)
model_1 = tf.keras.Model(inputs, outputs, name = "1_feed_forward")
```

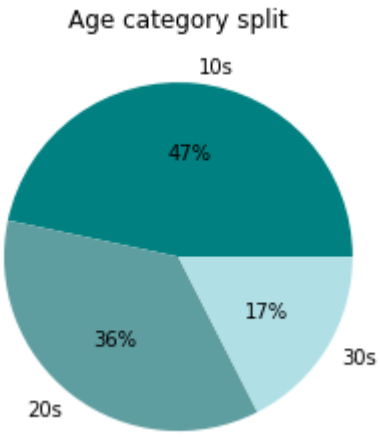
Code sample 6-17: Feedforward model using the functional API

The basic NN consists of a few layers, first, the texts (inputs) are put through the text vectorizer and the embedder, ensuring that all the data going into the neural network are numerical. This data is then passed through a global max-pooling layer to reduce dimensionality (can't be passed into a dense layer without) followed by the final Dense layer of size 3, representing the 3 possible categories.

The initial results were quite appalling, and when ran on a subset of 1000 data points, it just continuously guessed label 1 (the 20s). Adding an additional dense layer at least made it guess in each category, but the results don't beat the baseline and aren't anything of greater interest:

Accuracy	53.54%
Precision	59.15%
F1	55.67%
Recall	53.54%

Table 6-9: FNN result with 2 dense layers



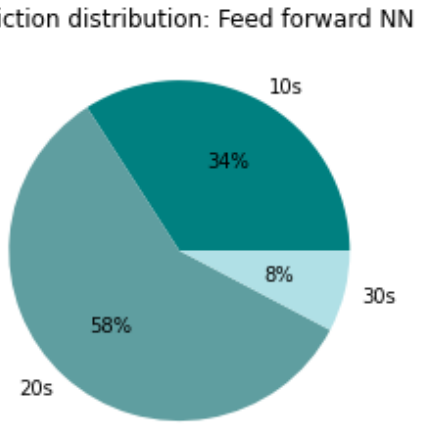
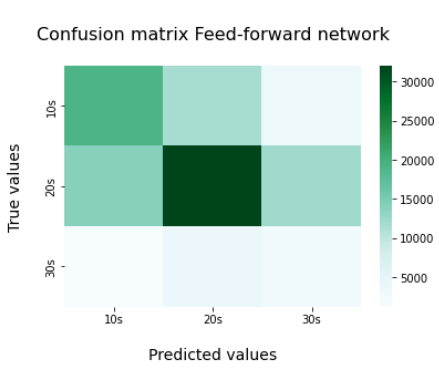


Figure 6-5 (left): Expected age category split

Figure 6-6 (right): Best achieved results using FNN so far

The feed-forward network seems unable to filter out the important features and looking at its confusion matrix (Figure 6-7), there doesn't seem to be much of a system, confirmed by its history diagram (Figure 6-8). We can see that neither the validation loss nor the accuracy improve over time. Initially, I thought the model was memorising the data shown, but adding early stopping and dynamically changing the learning rate had little to no impact on the results, still achieving a training accuracy of over 99%, but a validation and testing accuracy of 54%. Clearly, it is not learning the correct features.



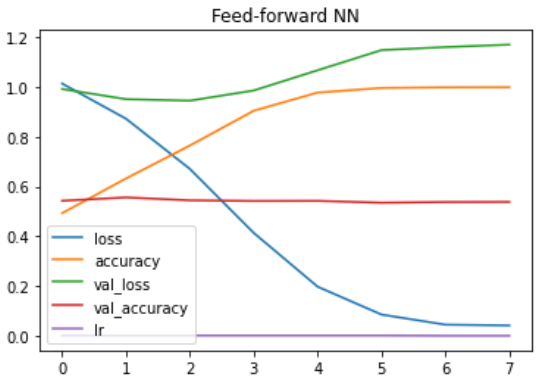
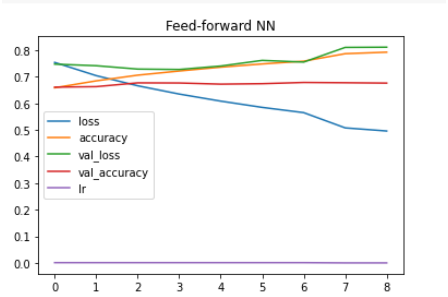


Figure 6-7 (left): Confusion matrix FNN

Figure 6-8 (right): History FNN, x=epochs, y = size

I added two more Dense layers, one with 32 neurons before and with 128 neurons after the existing 64 neuron layer, increasing the validation accuracy, but the model

is still not learning and improving its results over time, with the validation loss increasing rather than decreasing over time (Figure 6-9Figure 6-10) and in a later experiment, except the loss, none of the variables changed at all. (Figure 6-10)



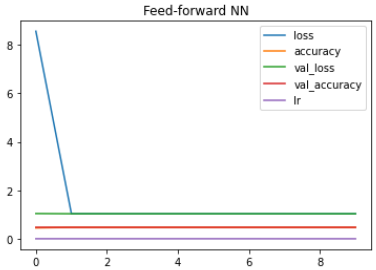


Figure 6-9 (left): Increasing validation loss

Figure 6-10 (right): (Validation) accuracy and validation loss don't change over time

6.4.2 CNN

I decided to leave the Feedforward network for a minute, assuming that it would achieve the worst results, and modelled, fit and compiled a convolutional neural network. The process is very similar, with the addition of a 1D convolutional layer, the number of neurons, stride and padding set arbitrarily.

```
inputs = layers.Input(shape = (1,), dtype="string")
x = text_vectorizer(inputs)
x = embedding_layer(x)
x = layers.Conv1D(64,4,1, padding="same")(x)
x = layers.GlobalMaxPool1D()(x)
outputs = layers.Dense(3, activation = "softmax")(x)
model_2 = tf.keras.Model(inputs, outputs, name="2_CNN")
```

Code sample 6-18: first CNN model

Although better than the FNN results, the findings weren't anything to be excited about. Although the resulting accuracy, F1 and recall were higher (see results in Table 6-10) than the baseline (64.25%, 58.16% and 64.25% respectively), the model is not learning and improving on those results (Figure 6-11).

Accuracy	67.75%
Precision	67.45%
F1	67.55%
Recall	67.75%

Table 6-10: CNN initial experiment results

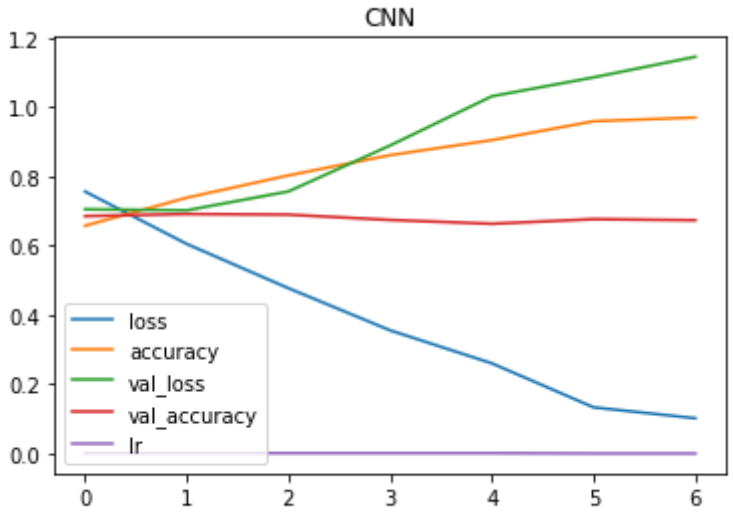


Figure 6-11: The training loss is decreasing and the accuracy increasing as expected, validation results aren't conforming.

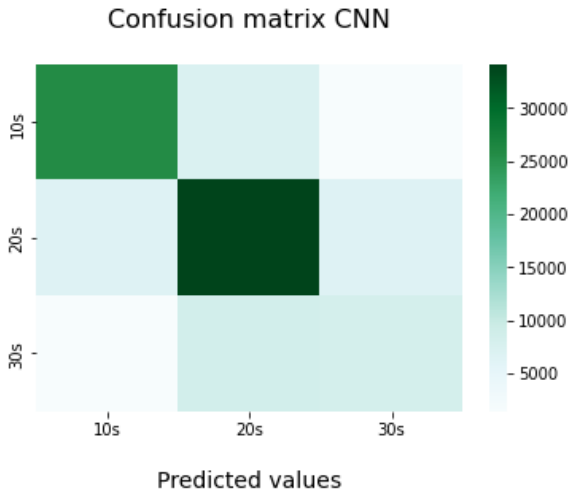


Figure 6-12: Confusion matrix CNN, confuses 30s for 20s

The model seems to pick out authors in their 10s and 20s quite well, although this seems to be more of an accident than any real, scientific reason, as changing the parameters barely affects those results.

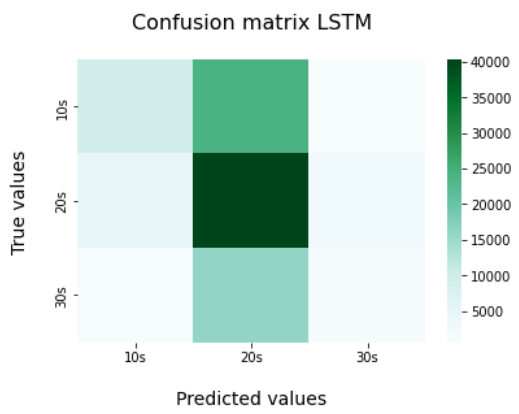
6.4.3 LSTM

After a handful of CNN experiments, I tried my hand at long short-term memory networks. Again, the same principle applies, we provide input and output shapes and the layers between, in this case using a special LSTM layer instead of the Conv1D layer.

```
inputs = layers.Input(shape = (1,), dtype = "string")
x = text_vectorizer(inputs)
x = embedding_layer(x)
x = layers.LSTM(64)(x)
outputs = layers.Dense(3, activation = "softmax")(x)
model_3 = tf.keras.Model(inputs, outputs, name = "model_3_LSTM")
```

Code sample 6-19: LSTM initial model

As accustomed to by now, the model works well on neither validation nor testing data. In the case of the initial LSTM model, it predicted the 20s (label 1) in 80% of the cases, only trying to maximise its statistically correct number of answers, not learning age-specific features.



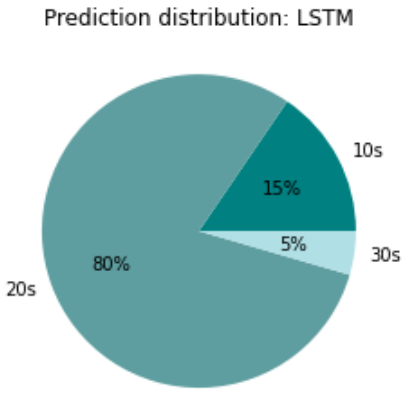


Figure 6-13 (left): LSTM confusion matrix, guessing 20s in 80% of the cases

Figure 6-14 (right): LSTM prediction distribution, guessing 20s in 80% of the **cases**

Accuracy	51.09%
Precision	50.65%
F1	45.30%
Recall	51.09%

Table 6-11: LSTM initial model results

6.4.4 Enhancing model accuracy

After running a few more experiments on each type of model, receiving equally terrible results - around 50% when run on a smaller subset - with the models not really learning no matter what I changed. When looking at the model summaries (such as the one shown in Figure 6-15) I realised they all had something in common, most of the trainable features are not in the neural network but in the embedding layer.

```
Model: "model_3_LSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1)]               0

 text_vectorization (TextVec  (None, 207)              0
 torization)

 embedding (Embedding)       (None, 207, 64)           3200000

 lstm (LSTM)                 (None, 64)                33024

 dense (Dense)               (None, 3)                 195

=================================================================
Total params: 3,233,219
Trainable params: 3,233,219
Non-trainable params: 0
_________________________________________________________________
```

Figure 6-15:LSTM model summary, 3 200 000 of the 3 233 219 trainable parameters are in the embedding layer.

```
embedding_layer = layers.Embedding(input_dim = vocab_length,
                                   output_dim = 256,
                                   )
```

Code sample 6-20: Original embedding layer that could be causing problems

Removing the embedding layer and replacing it with a normalization layer indeed caused the network to learn when ran on a subset jumping from a loss of 200 (validation: 86) all the way down to 0.98 (val: 1.28), but the resulting accuracies were a lot lower, with 50% on the training data and 46% on the validation. (increased from 38% and 34% respectively). Once run on the full dataset, the accuracy starts stronger around 45% (val: 47%) with a loss of 8.5 (1.0) but barely gets better, reaching 47% accuracy with a loss of 1.03 on both training and validation sets. It could be argued that in reality, it learns a lot more, as the networks work in batches of 32, updating the weights as they go. It may just find a well-working collection of weights very early on, but as the predictions are all over the place, further investigation is needed.

Recompiling and rerunning the feed-forward model, it can be seen that the initial loss is a lot higher and the accuracy a lot lower, scoring a loss of 351.9744 and accuracy of 0.3693.

Substituting the embedded layer back, using a lower parameter (8, 16, 32, 64) and a lower vocabulary size (10 000, 20 000 words) finally showed growth in validation accuracy, but only 2 per cent over a span of 10 epochs. Running the model on testing data however showed a precision of 1.0 – we are back to guessing just 20s and receive no false positives, meaning a statistically smart guess.

Next, I prepared the layers in the model in a separate step, so that .get_weights() could be run on them. As seen in Code sample 6-21, the weights cover a relatively wide range of values and they are not all evenly distributed as the results could implicate. Just in case, I decided to add some dropout layers, using 0.1 as the dropout. Randomly nulling 10% of the weights, forcing the model to adapt and the other nodes to catch the slack. Unfortunately, this did not affect the results by more than a few per cent combined with other changes.

```
#prepare new layers
conv_layer = layers.Conv1D(64,4,1, padding="same")
output_layer = layers.Dense(3, activation = "softmax")
global_pooling_layer = layers.GlobalMaxPool1D()
#use layers
inputs = layers.Input(shape = (1,), dtype="string")
x = text_vectorizer(inputs)
x = embedding_layer(x)
x = conv_layer(x)
x = global_pooling_layer(x)
outputs = output_layer(x)
model_2 = tf.keras.Model(inputs, outputs, name="2_CNN")
```

Code sample 6-21: Separating layer definition from their usage

```
dense_layer_3.get_weights()
```

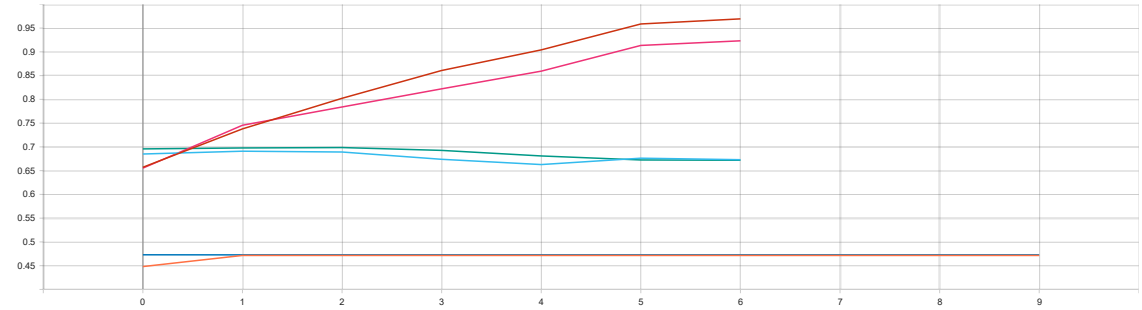
```
[array([[ 0.09230202,  0.05559348, -0.15029868, ..., -0.04299978,
         -0.08087286, -0.07671764],
        [-0.0238497 , -0.00609609, -0.05927556, ..., -0.14589755,
         -0.17585634, -0.07961053],
        [ 0.06238399, -0.03362613, -0.06703521, ...,  0.13699299,
         -0.05027273, -0.08784644],
        ...,
        [-0.10273495, -0.16665657,  0.14050847, ..., -0.14844745,
         -0.12319705, -0.00936369],
        [-0.03622045, -0.02444219, -0.03673013, ...,  0.07300149,
          0.03701668, -0.02344011],
        [ 0.10978453,  0.00485663, -0.00476059, ...,  0.05006269,
          0.05037927,  0.02321975]], dtype=float32),
 array([-4.99891788e-02, -3.80776733e-01, -8.44740495e-02, -1.28097013e-01,
        -2.51790464e-01, -9.68514849e-03, -3.07693630e-01, -3.56721491e-01,
        -7.31859952e-02, -8.24742988e-02, -1.63703933e-01, -3.08662087e-01,
        -1.05961010e-01, -3.67176309e-02, -5.13489416e-04, -4.08391833e-01,
        -1.10069543e-01, -2.88312435e-01, -9.31289867e-02, -1.71314716e-01,
        -3.11688364e-01, -1.61650464e-01, -5.20793140e-01, -4.10489559e-01,
        -2.35761225e-01, -3.01748633e-01, -7.58873820e-02, -1.53958127e-01,
```

Figure 6-16: A subset of the weights of dense_layer_3 in the initial FNN

```
inputs = layers.Input(shape=(1), dtype = tf.string)
x = text_vectorizer(inputs)
x = embedding_layer(x)
x = global_pooling_layer(x)
x = dense_layer_1(x)
x = dropout_1(x)
x = dense_layer_2(x)
x = dropout_2(x)
x = dense_layer_3(x)
outputs = output_layer(x)
```

Code sample 6-22: Adding two dropout layers with layers.Dropout(0.1)

Next, I updated my code to use TensorBoards, a visual tool provided by TensorFlow to compare experiments, in case my graph drawing was the cause of the error. This was not the case.



Name	Smoothed	Value	Step	Time	Relative
1_feed_forward/50000_vocab/14/07-2022-08:52:55/train	0.4715	0.4715	3	Thu Jul 14, 10:58:08	3m 53s
1_feed_forward/50000_vocab/14/07-2022-08:52:55/validation	0.4728	0.4728	3	Thu Jul 14, 10:58:08	3m 53s
2_convolutional/50000_vocab/14/07-2022-09:06:29/train	0.8608	0.8608	3	Thu Jul 14, 11:13:37	5m 17s
2_convolutional/50000_vocab/14/07-2022-09:06:29/validation	0.6739	0.6739	3	Thu Jul 14, 11:13:37	5m 17s
3_LSTM/50000_vocab/14/07-2022-09:19:04/train	0.8222	0.8222	3	Thu Jul 14, 11:31:30	9m 21s
3_LSTM/50000_vocab/14/07-2022-09:19:04/validation	0.6926	0.6926	3	Thu Jul 14, 11:31:30	9m 21s

Figure 6-17: A subset of the experiments shown simultaneously using TensorBoard.dev.

I ran another approximately 100 experiments changing the following parameters:

- CUT_OFF_LENGTH – including short texts I may not consider valuable.
- OUTPUT_LENGTH –number of characters of each text being compared.
- MAX_TOKENS – the vocabulary size, the number of different words allowed.
- Use of embedding layer vs. a normaliser.
- Embedding layer output_dimension
- EPOCHS – first changed manually, early stopping callback added for automation.
- BATCH_SIZE – the number of examples passed through the network before backpropagating.
- The number of neurons in each of the layers.
- The number of layers in the network.
- The order of the layers.
- Optimizers - Adam vs SGD
- Activation functions - Tanh, Relu, Sigmoid

I changed one parameter at a time, to see what had a positive or negative effect on the experiments, but hardly any of the changes had any impact on the results at all. Increasing the batch size to 256, reducing the max tokens to 20 000 and with early stopping and dynamic learning rate, I managed to get the feed-forward network results up by 10%, but the learning is barely visible. Going from 59.83% accuracy to 71.16 (val: 63.01 to 65.51) and the loss decreasing from 86.10% to 66.30% (val: 0.80, flattening out around 0.78), finally obtaining a graph looking more like my expectations (Figure 6-18).

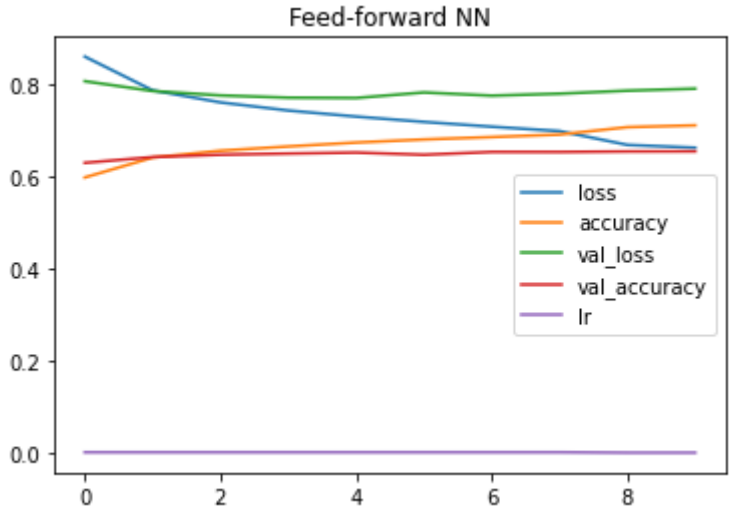


Figure 6-18: Losses decreasing and flattening out, accuracies increasing.

7 Discussion

This chapter discusses the results achieved in Chapter 6, comparing them to the hypothesis made in section 2.1 and finally provides ideas for improvements and further research.

7.1 Achieved results, summary

My hypothesis about the naive Bayes results, expecting a result of 60-70% was correct, achieving 64%.

The expected neural network accuracies was 70-80% and was only just missed, reaching 68% with one of the CNN models.

The hypothesis that LSTM would outperform the other models turned out to be wrong and it performed the worst of all, reaching only a maximum accuracy of 51%. LSTMs are focused on both long-term and short-term information. When reading some of the texts, there is little relation between the beginning and the end of a blog post. The authors' thoughts don't seem as concise as in different forms of text and could have impacted the results negatively. Also, the many linguistical errors may have impacted this.

Instead, the CNN performed the best, likely because the feature extraction process is close to the methods used by linguists when manually categorizing texts. Just like naive Bayes, it focused more on individual textual features, and a few misspelt words would not affect this as much.

Despite my best efforts, I could not get this neural network to learn in the typical sense of the word and I have to conclude that I will not, in a reasonable timeframe, find a learning neural network model that would beat the baseline set using naive Bayes calculations. Although several of my models beat the 64% accuracy baseline, the neural network models were simply making statistically smart decisions rather than learning textual features and basing their decisions on those.

Because of this reason, I have to conclude that on this corpus using this preprocessed data, naive Bayes is a more appropriate method.

7.2 Further investigation

There is a high likelihood that a learning neural network beating the naive Bayes baselines exists, I just did not find one in a reasonable timeframe. Some examples of what could be investigated in further detail are:

Different kinds of layers: for instance a Gated Recurrent Unit, a recurrent alternative to LSTM, could be used.

Using a pre-trained model: either using a model trained on similar texts and applying it to this model or applying feature extraction. This is where you take a pre-trained model, freeze the layers where it learned important textual features, and change the output layer to fit your needs. Many pre-trained models can be found on TensorFlow Hub. Alternatively, a pre-trained encoder could be used instead of manual encoding. The universal sentence encoder is a good example of such an encoder that could be used. (Cer et al., 2018)

Further parameter changes: There are an infinite number of combinations, and I may simply have missed the ideal combination. Ideally, a table would be created, updating one parameter at a time, from the minimum to the maximum considered value and saving all the relevant metrics for each run. The results would be more structured and any affecting factors would be more easily recognisable. This, within the set bounds, would be an exhaustive method and require more time, but an optimal solution in the set range would indubitably be found. Further checks would have to be performed to ensure the best results are created by a learning model and not a statistically optimised guessing model, but those cases would easily be eliminated.

Speed up the code: my code ran relatively slow at points. I already optimised the data loading, using a GPU and 8-based numbers, but there were still parts where my code ran slower than I would like. Mainly the regular expressions required a lot of time to run. Removing the punctuation required approximately a minute. In the past, it was possible to use tf.keras.preprocessing.text.Tokenizer, simply filtering the characters, but it has been deprecated. It got replayed with

tf.keras.layers.TextVectorization but its features are more limited to options such as “lower_and_strip_punctuation”. Although this is enough in many cases, it may not always satisfy the needs. This would also have removed my ability to filter out shorter messages early on. Alternatively, the texts could have been filtered using a stream, token not in the python-specific string.punctuation, which contains the most common punctuation types:

*! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~*

Removing additional sources of bias: As the models often ended up guessing only the 20s, it could be argued that the data is biased towards the 20s. However, as we saw during the data analysis, the 20s are only just overrepresented with 35% of the whole data set, compared to 33% which it would have been in an ideal situation. If bias was the issue, I would have expected the results to be biased towards option 0, the 10s which is represented by a full 47%, a full 14% more than an ideal representation. This could however explain the models where both the 10s and 20s were correctly represented and it only struggled with the 30s. Removing additional entries to more evenly represent the age categories could be further investigated. It may also be worth removing frequent writers, so as to not learn individual peoples’ styles. An alternative rendition of the scientific article “Mining the Blogosphere: Age, gender and the varieties of self-expression thesis (Argamon et al., 2007a) contains several tables with a detailed analysis of the use of certain topics and vocabulary in the different age groups that can be used for inspiration.

The effect of using lowercase letters only: Capitalization is an important part of our language and something we learn over time. I assume that adults are more likely to capitalize correctly and this could be another feature used to classify the texts correctly. A similar statement is true for improper punctuation use. With my limited computing power, I wanted to limit myself in the vocabulary size, and this was a sacrifice I was willing to make, although it for sure is an interesting concept to look into. This concept is mentioned in, among other papers, Stylometric analysis of Bloggers’ age and gender: “As blogs are informal writing, the bloggers’ may not abide by the grammatical and editorial rules (…). There are grammatical errors in the blog

writing like improper use of full stop (.), exclamation marks (!) and capital letters." (Goswami et al., 2009)

Using n-grams: I could have experimented with n-grams and rather than looking at only one word at a time, using among the most simple tokenization methods. Some additional context could have been extracted using n-grams.

NLTK: In the future, I would use the NLTK options, and not the just-out-of-experimental Embedding layer. This limited my options, and actions such as trying to remove stop words, lemmatization and similar NLP preprocessing techniques would cause the embedding to throw unsolvable errors. The Keras TextVectorization process could be replaced with SciKit learn's preprocessing options, giving the experimenter more flexibility. I still think that the text-to-number conversion and meaningless words may be the reason for the disappointing results and having the ability to fine-tune more of those features could make a large difference.

Transformers: As the state-of-the-art method for natural language processing, using transformers could have constituted in better result. As they are built on harder concepts, I did not consider them a viable option for this project, but they are an interesting concept that can be explored in the future.

Shuffling: Reshuffling the training data before each forward pass, so the model does not learn the order.

Age as a variable: The article *Why gender and age prediction from tweets is hard* (Nguyen et al., 2010) used age and gender as sociological variables, rather than fully scientific variables. They found that 10% of the tweets used were written by users where the gender does not agree with the sex. They also found that the language use was heavily influenced by the surrounding people. This concept may flow into age too, and an additional experiment could be created, taking into account the age of the closest 5 people in their lives.

Text augmentation: For images, image augmentation can be used to perform a few simple transformations (rotation, translation, zoom and reflection) to artificially grow the dataset and provide more similar yet slightly different examples. I am not aware of similar augmentation methods for texts, but further research may.

7.3 Organisational standpoint

Some of the other changes I would implement are not related to further research, but rather to changes from an organisational perspective.

Organised code: I would have split the code into different files earlier on. I am already using a main file and a helper functions file, but this was not enough. I should have saved my preprocessed data set and then had 3 separate code files for each type of model. I had to keep scrolling back and forth, trying to find the correct pieces of code, related to the model I was working on.

Computing power: The combination of my laptop and Colab was not ideal for such a large corpus. I was not able to open the file in Excel in advance, due to a lack of RAM and all the corpus familiarization had to be done through Colab, where I could only load in a few lines at a time. Additionally, as the code took some time to run as a whole, I would often perform other tasks in between, but Colab only stays active for a certain time until it deactivates. I would often see the results of the last run, only to change one line and have to run the whole code again as Colab had gone inactive.

Code readability: I would have kept experimental parts more separate and functionalised more of the repeated code. My Python knowledge was not good enough for this at the start, but it is something to consider for the future.

TensorBoard: I only found out about the TensorBoard callbacks towards the end of my thesis, after already having run well over 50 experiments. A visual representation of all of them on one diagram, easily comparable, rather than my notes all over the place would have pushed my thesis to the next level.

Extent: I largely underestimated how much work would go into each model and my initial idea of using 5 models (the 3 used now, GRU and a pre-trained feature

extractor) quickly changed to 3, where 2 probably would have been sufficient for a bachelor thesis.

8 Sources

Adobe Photoshop. (2021, May 17). *Photoshop- Using filters*. https://helpx.adobe.com/photoshop/using/filter-basics.html

AISBX. (2019). *AISBX - Loebner Prize*. https://web.archive.org/web/20190822190535/http://aisb.org.uk:80/events/loebner-prize

Amani, A., & Soleimany, A. (2018). *Intro to deep learning—MIT 6.S191*. http://introtodeeplearning.com/

Argamon, S., Koppel, M., PenneBaker, J. W., & Schler, J. (2007a). Mining the Blogosphere: Age, gender and the varieties of self-expression. *First Monday*, *12*(9 (September 2007)). https://firstmonday.org/ojs/index.php/fm/article/view/2003/1878

Argamon, S., Koppel, M., PenneBaker, J. W., & Schler, J. (2007b). Mining the Blogosphere: Age, gender and the varieties of self-expression. *First Monday*, *12*(9). https://firstmonday.org/ojs/index.php/fm/article/view/2003/1878

Argamon, S., Koppel, M., PenneBaker, J. W., & Schler, J. (2009). *Automatically Profiling the Author of an Anonymous Text*. https://sci-hub.se/10.1145/1461928.1461959

Aruchamy, V. (2021, September 29). *How To Plot Confusion Matrix In Python And Why You Need To?* https://www.stackvidhya.com/plot-confusion-matrix-in-python-and-why/

Bailey, P., Delgado, J., & Hyttsen, M. (2019, March 2). *Intro to TensorFlow for Deep Learning*. https://classroom.udacity.com/courses/ud187

Beri, A. (2020, May 14). *Stemming vs Lemmatization*. https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language processing with Python* (1st ed.). O'Reilly Media, Incl. http://117.3.71.125:8080/dspace/bitstream/DHKTDN/6460/1/Natural%20Language%20Processing%20with%20Python.4149.pdf

Bonaros, B. (2019, August 5). Keep Punctuation In NLP Tasks With Python. *Predictive Hacks*. https://predictivehacks.com/tokenizer-for-nlp-tasks/

Bonta, R. (2018). *California Consumer Privacy Act (CCPA)* [Government website]. State of California Department of Justice. https://oag.ca.gov/privacy/ccpa#:~:text=The%20California%20Consumer%20Privacy%20Act,how%20to%20implement%20the%20law.

Bre, F. (2017). *Artificial neural network architecture (ANN i-h 1-h 2-h n-o).* https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051

Brighton, H., & Selena, H. (2007). *Introducing Artificial Intelligence*. Icon Books Ltd.

Brownlee, J. (2017, September 22). What Is Natural Language Processing? *Machine Learning Mastery*. https://machinelearningmastery.com/natural-language-processing/

Brownlee, J. (2019, July 5). *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

Brownlee, J. (2020, January 9). A Gentle Introduction to the Rectified Linear Unit (ReLU). *Machine Learning Mastery*. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Brownlee, J. (2021a, February 1). *Difference Between Backpropagation and Stochastic Gradient Descent*. https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/#:~:text=Specifically%2C%20you%20learned%3A-,Stochastic%20gradient%20descent%20is%20an%20optimization%20algorithm%20for%20minimizing%20the,a%20neural%20network%20graph%20structure.

Brownlee, J. (2021b, February 2). *How to Use Word Embedding Layers for Deep Learning with Keras*. https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Brownlee, J. (2021c, October 12). *Code Adam Optimization Algorithm From Scratch*. https://machinelearningmastery.com/adam-optimization-from-scratch/

Carlson, S. (1985). A double-blind test of astrology. *Nature*, *318*. https://muller.lbl.gov/papers/Astrology-Carlson.pdf

Cer, D., Kong, S., & Hua, N. (2018). *Universal-sentence-encoder*. TensorFlow Hub. https://tfhub.dev/google/universal-sentence-encoder/4

Cesare, N., Grant, C., Nguyen, Q., Lee, H., & Nsoedie, E. O. (2017). *How well can machine learning predict demographics of social media users?* https://arxiv.org/abs/1702.01807

Chollet, F. (2015). *Keras*. https://keras.io/

Chris Firth, Bekinschtein, T., Bor, D., Jarrett, C., Kanai, R., O'Shea, M., & Ward, J. (2014). *30-second brain* (A. Seth, Ed.; 1st ed.). Ivy Press.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. https://arxiv.org/pdf/1412.3555v1.pdf

CNN: Convolutional Neural Networks Explained—Computerphile. (2016, May 20). Computerphile. https://youtu.be/py5byOOHZM8

Conti, M. (Director). (2017, February 28). *The incredible inventions of intuitive AI*. TED.

Cournapeau, D. (2007). *Scikit-learn*. https://scikit-learn.org/stable/

Dixon, L., Li, J., & Jeffrey Sorensen. (2018, 3/2). *Measuring and mitigating unintended bias in text classification*. AIES'18, New Orleans. https://dl.acm.org/doi/pdf/10.1145/3278721.3278729

Duc Pham, D., Tran, G. B., & Pham, S. B. (2009). *Author Profiling for Vietnamese Blogs*. 2009 International Conference on Asian Language Processing, Hanoi, Vietnam.

Facebook help center. (2022). *What is data scraping and what can I do to protect my information on Facebook?* https://www.facebook.com/help/463983701520800

Fleisch, D. (Director). (2011, November 21). *What's a Tensor?* https://www.youtube.com/watch?v=f5liqUk0ZTw

French, Dr. C. (2014). *Why demographic data matters*. https://extension.unh.edu/sites/default/files/migrated_unmanaged_files/Resource004765_Rep6784.pdf

Gandhi, R. (2018, May 5). *Naive Bayes Classifier*. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

GeeksForGeeks. (2019, October 15). *String.punctuation in Python*. GeeksForGeeks. https://www.geeksforgeeks.org/string-punctuation-in-python/

GeeksForGeeks. (2022, May 18). Removing stop words with NLTK in Python. *GeeksForGeeks*. https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

Godoy, D. (2018, November 21). *Understanding binary cross-entropy / log loss: A visual explanation*. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Gómez, R. (2018, May 23). *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. https://gombru.github.io/2018/05/23/cross_entropy_loss/

Google Brain. (2015). *TensorFlow*. https://www.tensorflow.org/

Google Cloud. (2016, May). *Cloud Tensor Processing Units (TPUs)*. https://cloud.google.com/tpu/docs/tpus

Goswami, S., Sarkar, S., & Rustagi, M. (2009, 2020). *Stylometric Analysis of Bloggers' Age and Gender*. 3rd international AAI conference on weblogs and social media, San Jose California. https://ojs.aaai.org/index.php/ICWSM/article/view/13992/13841

Guimaraes, R. G., Rosa, R. L., de Gaetano, D., Rodriguez, D. Z., & Bressan, G. (2017). *Age Groups Classification in Social Network Using Deep Learning*. https://e-tarjome.com/storage/panel/fileuploads/2019-01-23/1548237695_E11599-e-tarjome.pdf

Gupta, K. (2019, June 19). How to Load Kaggle Datasets Directly into Google Colab? *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/06/how-to-load-kaggle-datasets-directly-into-google-colab/

Hinds, J. (2018). *What demographic attributes do our digital footprints reveal? A systematic review*. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207112

Holst, A. (2021, June). *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025*. Statista. https://www.statista.com/statistics/871513/worldwide-data-created/

Huang, X., & Paul, M. J. (2019, 7/6). *Neural User Factor Adaptation for Text Classification: Learning to Generalize Across Author Demographics*. Eighth Joint Conference on Lexical and Computational Semantics, Minneapolis. https://aclanthology.org/S19-1015.pdf

Hunter, J. D. (2003). *Matplotlib*. https://matplotlib.org/

IBM Cloud Education. (2020, July 2). Natural Language Processing (NLP). *IBM Cloud Learn Hub*. https://www.ibm.com/cloud/learn/natural-language-processing

Internet usage statistics. (2021, July 3). https://www.internetworldstats.com/stats.htm

Intersoft Consulting. (2018, May 23). *General Data Protection Regulation—GDPR*. Intersoft Consultin. https://gdpr-info.eu/

Ježek, B. (2021, May). *Počítačová vidění 2- Fitrace Obrazu*.

Johnson, J. (2021, February 6). *Pretrained Word Embeddings using SpaCy and Keras TextVectorization*. https://towardsdatascience.com/pretrained-word-embeddings-using-spacy-and-keras-textvectorization-ef75ecd56360

Keras. (n.d.). *LSTM layer*. https://keras.io/api/layers/recurrent_layers/lstm/

Khanna, C. (2021, August 13). Byte-Pair Encoding: Subword-based tokenization algorithm. *Towards Data Science*. https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

Kim, M., Xu, Q., Qu, L., Wan, S., & Paris, C. (2017). *Demographic Inference on Twitter using Recursive Neural Networks*. 471–477. https://aclanthology.org/P17-2075.pdf

Kingma, D. P., & Lei Ba, J. (2017, January 30). *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. ICLR 2015. https://arxiv.org/pdf/1412.6980.pdf

Koehrsen, W. (2018, October 2). *Neural Network Embeddings Explained*. Towards Data Science. https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

Kostadinov, S. (2017). Understanding GRU Networks. *Towards Data Science*. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Lopez, dProgrammer. (2019). *RNN, LSTM & GRU*. http://dprogrammer.org/rnn-lstm-gru

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Information retrieval*. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

McKinney, W. (2008). *Pandas*. https://pandas.pydata.org/

Myatt, G. J. (2007). *Making sense of data*. John Wiley & Sons, Inc.

Neagoie, A., & Bourke, D. (Directors). (2020, November). *TensorFlow Developer Certificate in 2022: Zero to Mastery*. https://www.udemy.com/course/tensorflow-developer-certificate-machine-learning-zero-to-mastery/

Nguyen, D., Trischniigg, D., Dogruöz, A. S., Gravel, R., Theune, M., Meder, T., & de Jong, F. (2010). *Why Gender and Age Prediction from Tweets is Hard: Lessons from a Crowdsourcing Experiment*. https://aclanthology.org/C14-1184.pdf

niDirect. (2022). *Social media, online gaming and keeping children safe online* [Government website]. https://www.nidirect.gov.uk/articles/social-media-online-gaming-and-keeping-children-safe-online#:~:text=cyberbullying%20(bullying%20using%20digital%20technology,seeing%20offensive%20images%20and%20messages

Nielsen, M. (2019). *Neural networks and deep learning*. http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

NLTK. (2001). *NLTK documentation*. https://www.nltk.org/

Olah, C. (2015, August 27). Understanding LSTM Networks. *Colah's Blog*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Olisa, S. (2018). *Original image (b) Haar (c) Laplacian, (d) Gabor, (e) Sobel (f) Laplacian of Gabor [proposed] (g) Sobel of Gabor and [proposed] (h)*

Laplacian of Sobel of Gabor [proposed]. https://www.researchgate.net/figure/Original-image-b-Haar-c-Laplacian-d-Gabor-e-Sobel-f-Laplacian-of-Gabor_fig2_324911043

OpenAI. (2020a). *Open AI - GPT-3 Examples*. https://beta.openai.com/examples/

OpenAI. (2020b). *OpenAI*. https://openai.com/api/

Pai, A. (2020, May 26). What is Tokenization in NLP? Here's All You Need To Know. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

Peersman, C., Daelemans, W., & Van Vaerenbergh, L. (2011, October 28). *Predicting Age and Gender in Online Social Networks*. Proceedings of the 3rd International CIKM Workshop on Search and Mining User-Generated Contents, Glasgow, UK. https://www.researchgate.net/publication/221615645_Predicting_age_and_gender_in_online_social_networks

Peralta, B., Figueroa, A., Nicolis, O., & Trewhela, Á. (2021). *Gender Identification From Community Question Answering Avatars*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9625020

Potthast, M., Rosso, P., Stamatatos, E., & Stein, B. (2009). *PAN conference*. https://pan.webis.de/

Powers, D. M. W. (n.d.). *The Total Turing Test and the Loebner Prize* [Flinders Universitry of South Australia]. https://aclanthology.org/W98-1235.pdf

Raising Children Australia. (2006). *Internet safety: Children 6-8*. https://raisingchildren.net.au/school-age/play-media-technology/online-safety/internet-safety-6-8-years

Redmon, J. (Director). (2017, August 18). *How computers learn to recognize objects instantly*. TED. https://youtu.be/Cgxsv1riJhI

Reed, S., Żołna, K., Gómez Colmenarejo, S., Novikov, A., Barth-Maron, G., Giménez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Razavi, A., Edwards, A., Heess, N., Chen, Y., Radsell, R., Bordbar, M., & de Freitas, N. (2022). *A Generalist Agent*. https://www.deepmind.com/publications/a-generalist-agent

Ritvikmath (Director). (2020, November 11). *TFIDF : Data Science Concepts*. https://youtu.be/OymqCnh-APA

Romero, A. (2021, May 24). *A Complete Overview of GPT-3—The Largest Neural Network Ever Created*. https://towardsdatascience.com/gpt-3-a-complete-overview-190232eb25fd

Russo, V. (Director). (2019, April 10). *Python and TensorFlow: Text Classification—Part 4* (Vol. 4). Lucid Programming. https://youtu.be/tC1amOAkojE

Saaty, T. L. (2000). *The Brain- Unravelling the mystery of how it works* (1st ed.). RWS Publications.

Sanderson, G. (Director). (2017a, October 16). *Gradient descent, how neural networks learn* (Vol. 2). 3Blue1Brown. https://youtu.be/IHZwWFHWa-w

Sanderson, G. (Director). (2017b, November 3). *Backpropagation calculus* (Vol. 4). 3Blue1Brown. https://youtu.be/tIeHLnjs5U8

Schler, J., Koppel, M., Argamon, S., & PenneBaker, J. W. (2005). *Effects of Age and Gender on Blogging- Blog authorship corpus*. https://u.cs.biu.ac.il/~koppel/BlogCorpus.htm

Scikit learn. (2011a, September 2). *Naive Bayes*. https://scikit-learn.org/stable/modules/naive_bayes.html

Scikit learn. (2011b, September 2). *Sklearn.naive_bayes.MultinomialNB*. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

Scikit learn. (2019a). *Sklearn.metrics.accuracy_score*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Scikit learn. (2019b). *Sklearn.metrics.precision_recall_fscore_support*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

Scikit learn. (2019c). *Confusion matrix without normalization, lilies*. https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

Scikit learn. (2019d, October 23). *Sklearn.metrics.recall_score*. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

Scikit learn, 2.9.2011. (2011c, September 2). *TfidfTransformer*. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html#:~:text=The%20formula%20that%20is%20used,document%20frequency%20of%20t%3B%20the

Scott, T. (Director). (2019, August 26). *I'm Not A Robot*. https://www.youtube.com/watch?v=o1zNIm8GVPY

Seaborn. (2012). *Seaborn.heatmap*. Seaborn. https://seaborn.pydata.org/generated/seaborn.heatmap.html

Sethi, A. (2020, March 6). One-Hot Encoding vs. Label Encoding using Scikit-Learn. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/

Singh, S. (2019, August 21). *NLP Essentials: Removing Stopwords and Performing Text Normalization using NLTK and spaCy in Python*. https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/

Starmer, J. (Director). (2020, June 3). *Naive Bayes, clearly explained*. https://youtu.be/O2L2Uv9pdDA

Sundman, J. (2003, February 26). Artificial stupidity. *Salon*. https://web.archive.org/web/20120720014628/http://www.salon.com/2003/02/26/loebner_part_one

Tatman, R. (2017, August 15). *Blog authorship corpus*. https://www.kaggle.com/rtatman/blog-authorship-corpus

Team NLTK. (2001). *Natural Language Toolkit*. https://www.nltk.org/

TensorFlow. (2017a, June 19). *Embedding projector*. Embedding Projector. https://projector.tensorflow.org/

TensorFlow. (2017b, June 19). *Word embeddings*. https://www.tensorflow.org/text/guide/word_embeddingsWord embeddings

TensorFlow. (2021, January 6). *Visualizing Data using the Embedding Projector in TensorBoard*. https://www.tensorflow.org/tensorboard/tensorboard_projector_plugin

TensorFlow. (2022, February 10). *Training a neural network on MNIST with Keras*. https://www.tensorflow.org/datasets/keras_example

The Functional API. (n.d.). TensorFlow. Retrieved 21 May 2021, from https://www.tensorflow.org/guide/keras/functional

Twitter. (2022). *Twitter API*. https://developer.twitter.com/en/docs/twitter-api

United States Census. (2022). *Age and Sex*. United States Census Bureau. https://www.census.gov/population/cspro/userregistration.html?resource=cspro75.exe

Urmson, C. (Director). (2015, June 26). *How a driverless car sees the road*. TED. https://youtu.be/tiwVMrTLUWg

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Kaiser, L. (2017). *Attention is all you need*. https://arxiv.org/pdf/1706.03762.pdf

Walber. (2014). *Precisionrecall* [Digital]. https://en.wikipedia.org/wiki/File:Precisionrecall.svg

Wang, J. D. (n.d.). *Python sklearn.metrics.accuracy_score() Examples*. Program Creek. https://www.programcreek.com/python/example/84417/sklearn.metrics.accuracy_score

Waskom, M. (2012). *Seaborn*. https://seaborn.pydata.org/

9 Appendix A

9.1 Glossary

ANN = Artificial Neural Network

CNN = Convolutional Neural Network

RNN = Recurrent Neural Network

LSTM = Long Short Term Memory, a type of RNN

GRU = Gate Recurrent Unit, a type of RNN

TP = True Positive, model correctly predicts the positive class

TN = True Negative, model correctly predicts the negative class

FP = False Positive, model wrongly predicts the positive class

FN = False Negative, model wrongly predicts the negative class

AI = Artificial Intelligence

ML = Machine Learning

FNN = Feed-Forward Neural Network

9.2 Attachments

Attached to this thesis is one .zip file named Anouk_Wilstra_Text_classification_with_artificial_neural_networks.zip containing three code files: age_classification_nn.ipynb, helper_functions.ipynb and activation_functions.ipynb.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Anouk Wilstra**

Studium: I2000441

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Klasifikace textu pomocí umělých neuronových sítí**

Název bakalářské práce AJ: Text classification with artificial neural networks

Cíl, metody, literatura, předpoklady:

Goal: Research data manipulation and neural network algorithms, and combine them to create an accurate text classification model.

1) Introduction to neural networks

2) Related research and state of the art results

3) Selecting a data set

4) Data preparation and manipulation

5) Building the neural network model

6) Enhancing model accuracy

7) Discussion of the achieved results

Buduma, N., & Locascio, N. (2017). *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms* (First edition). O'Reilly Media.

Demmelmaier, G., & Westerberg, C. (2021). *Data Segmentation Using NLP: Gender and Age* [Uppsala Universitet]. http://uu.diva-portal.org/smash/get/diva2:1527530/FULLTEXT01.pdf

Zadávající pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Ing. Milan Košťák

Datum zadání závěrečné práce: 12.1.2021