Can stylometric features help in the detection of spun content?

Patrick Twellmann

13. März 2020

Inhaltsverzeichnis

1 Motivation

Plagiarism is an ongoing problem that has always plagued the scientific community. Because of plagiarism it is more difficult to identify talented researchers and with that also diminishes the trust in the scientific community as a whole. This is not the only problem plagiarism creates, but is enough to understand the fight against it. Unfortunately, as plagiarism detection improves, so does plagiarism itself. The focus of this project was on the act of "spinning" through readily available tools.
"Spinning" refers to text processing that replaces words with synonyms of that word, thus altering the text while the syntax and semantics stay the same. Utilizing spinning, the newly created document would not be considered plagiarism by traditional plagiarism detection tools, since the text is not the same as it was pre-spinning.
Previously, together with a colleague (Florian Hölken), we worked on improving a web based tool for intrinsic plagiarism detection. This form of plagiarism detection aims to utilize the differences in writing style that occur when someone copy's part of a text and integrates it into their own text. Two usually have two different writing styles, and those differences can be quantified by stylometric features. These cover four categories of features (at least in our tool) and range from simple "average word length" to measurements for the ease of reading.

The idea behind this project was to use our intrinsic plagiarism tool to detect documents that have been altered through a spinning tool. At the beginning of the project there had been a rudimentary machine learning test run by my colleague. This neural network took the scores of all the stylometric features as an input to determine if the text was spun or not. Sadly the accuracy of the prediction was only around 60%. In order to improve this, I want to analyze the stylometric features in regards to the impact they have for the prediction, so that the neural network can be run again with a cleaned input to hopefully increase the accuracy.

2 Method

The idea behind the evaluation is simple: take a large number of text documents, spin them, run them through our intrinsic plagiarism tool (IntrinsicPD) with every feature and then look at the scores to determine the impact of the feature.

2.1 Prerequisites

So the first step is to get a large number of documents and spin them. I was provided around 2000 Wikipedia articles which have also be spun using the spinning tool "Spinbot.com" for a total of about 4000 articles. With that number of articles the results will be statistically significant.
Before I talk about the main project it is important to understand the basics of the IntrinsicPD tool. First you need to upload the document that you want to analyze, then you select the "atom" type from the choices of paragraph, sentence and nchars (nchars will be ignored in this project since I don't see value in decreasing the atom size below sentence, especially when some of the features are originally designed to be run on the text as a whole). Lastly you select the features to use on the document. 30 features from four different categories are available to choose from. This are the basics as needed in this project. There are two more options (Threshold value and clustering method) which will be ignored in this project to reduce the number of variables for this first analysis. I used "sentence" as atom type and left the rest of the options at their default value.

2.2 Python Scripts

In order to run 4000 documents through our tool I needed to develop a script, since running them manually through the tool is not feasible. There exists a command line tool for the IntrinsicPD website that I could have used, but instead I build my script on top that so that I could automate some simple statistical calculations (and change them later if need be). This script, written in python, took up the majority of my time working on this project. As with most programming task small details tend to need a lot of time to correctly implement. The full script is located in the appendix A, here I will describe the different parts of my script: First, in loadDocuments() the paths to the documents will be saved and sorted into original and spun for easier access in the script.
CreateFileStructure() is the main method of this script (I didn't know what a more fitting name should be since it grew bigger than originally anticipated). The folders get created as soon as they are needed in the script. The final file structure is shown in figure1. It should be noted that the script will delete and recreate the whole folder on each run, so successful runs need to be saved manually (I renamed the base folders).
The script functions through a three times nested loop: The first loop iterates

through all categories as defined in the beginning of the script. The second loop iterates through all features in a given category as defined in the beginning of the script. The third loop iterates through all documents, first the originals and then the spun documents. Here command_line.py gets called and the script gets a .csv file with the scores for the current feature and document.
Following the base analysis, a .csv file will be created for every document pair (original + spun) containing the differences of the atom scores. These differences are the main metric for this analysis. The idea being, if there is a large difference in the atom scores, this feature is useful in determining whether a text was spun or not.
At the end of every loop iteration values necessary for statistical calculations will be stored. As of writing, for a lack of a better metric, the average of the scores will be calculated on different levels: average of pair atom scores inside every feature and average of those averages across all features inside a category.
There are a number of helper methods to make the script a little more readable. FindFile returns the wanted document, analysis calls command_line.py and analyseDifferences creates the .csv with the atom score differences for each pair.
Besides this main script exists a second script B. This was designed to do some pre-processing, but ultimately failed. Due to time constraints I was not able to fix this and I will just briefly describe what it was supposed to do for completion sake.
ProcessDocuments is a helper method who aimed to equalize the original and the spun texts, as the spinning tool created some noise which the IntrinsicPD tool picks up as additional paragraphs. Therefore, to run an analysis with the atom type paragraph, this method was needed. CountWords was supposed to do as the name suggests. The idea being to compare the vocabulary of the original and spun texts to get more insight into the spinning tool. Analysis is a helper method that prepares the documents for use with the neural network and evaluationPreperation was supposed to prepare the word comparison analysis.

2.3 Analysis Idea

Now that the processing of the documents is automated I can talk about the idea behind the analysis. Since this is just the first exploration of the topic, I took a simple approach to the data to get a general idea if this is something valuable to research further. If there is a significant difference in atom scores after spinning, the feature is likely useful for the neural network. For this I needed to generate the averages, since going through the data manually is not practical (too much data). Using averages shouldn't be a problem for this analysis, since the documents represent whole plagiarized documents, and not just plagiarized passages. This helps us to understand the importance of the different features, however in the future more realistic scenarios need to be explored.
In order to see whether a smaller feature input helps improve the accuracy I will rerun the neural network from the previous project with a reduced input

and compare this to the former results. The neural network itself is a simple one consisting of the input and output layers, and two hidden layers with 128 neurons each (fully connected).

3 Results

After running the script I compiled a table with the feature averages. C Additionally I added information regarding the scale of the feature (if I could find such information) to get a better understand of the results. Then I color coded some easy identifiable results. Red colored features are insignificant. The highest difference average is -0.09 which is small in comparison to the rest of the results. Green coded features have an exceptionally high difference, and yellow features show a high difference, but not as big as the green features (the highest yellow feature difference is 3.69, the lowest green difference is 30.29).
Now I needed to choose the features for the neural network. Since this is the first analysis of the topic I decided to keep it simple and just have one category of features. The readability category had the highest density of yellow and green features which is the reason I chose those features as input for the neural network.
With the input features selected I first re ran the network with the full selection of features in order to confirm my baseline readings. A total of four runs were completed to counter any weird deviations.D After those runs I reduced the features to the selected readability features and ran the network four times again. E

The initial runs confirmed the 60% accuracy from the previous project. Unfortunately the reduced feature runs had a slightly lower accuracy of 59.5%. While this is disappointing, there are positives: first and foremost, the input is drastically reduced. Originally 30 features were input, this number dropped down to eight. This shows, that most of the features are unnecessary for the network, so it is possible to reduce the amount of data without reducing the accuracy. Even though there was a drop of 0.5% in the accuracy, this is likely due to the simple selection of the features. For example one of the green coded features was left out of the neural network input. With a proper feature selection a drop in accuracy can probably be prevented.

4 Conclusion

This first analysis regarding the viability of the IntrinsicPD tool to detect spun content showed that not all features are useful (when used in a neural network). While there was no improvement in accuracy, there was a drastic reduction of the input. Future projects could expand upon this in different ways:
The feature selection in this project was simple and "dumb". Without limiting to one category of features the accuracy should at least match the baseline again, however a gain in accuracy is also plausible.
Another possible improvement could come from the neural network. In this project I reused the network form the previous project for a better comparison. However, this network is rather simple and someone with extended knowledge in machine learning may be able to produce a neural network which is better suited for this task.
An investigation into the spinning tool (and different spinning tools) could also be helpful. There could be some exotic words introduced by those tools which may be used as an additional input to the neural network.
Lastly it would be interesting to rerun the analysis with the atom type "paragraph". Many of the stylometric features are designed to be used on a whole text and not just snippets. A bigger atom size might yield more accurate results for those features.

Anhang

A IPDAnalysis Script

```
import sys
import re
import platform
import os
import csv
import shutil
import numpy

runBaseAnalysis = True

path = ""
basePath = ""
seperator = "\\"

lexicalFeatures = ['punctuation_percentage', 'average_word_length','average_sent
syntacticFeatures = ['stopword_percentage','avg_internal_word_freq_class','avg_e
vocabularyRichness = ['honore_r_measure','hapax_legomena','hapax_dislegomena','si
readabilityFeatures = ['coleman_liau_index','automated_readability_index','linsea

allFeatures = {"LexicalFeatures":lexicalFeatures,"SyntacticFeatures":syntacticFe

segmentation = "paragraph"    #später Dateien aufräumen (ersetzen: linebreak -> l

originalDocuments = {}
spunDocuments = {}

def loadDocuments():
    global originalDocuments
    global spunDocuments
    global path

    for document in os.listdir(path):
        if(re.search("ORIG",document)):
            originalDocuments[document] = path + seperator + document
        else:
            spunDocuments[document] = path + seperator + document

def createFileStructure():
    global allFeatures
    global seperator
```

```
    global basePath
    global originalDocuments
    global spunDocuments
    global runBaseAnalysis

    basePath += seperator + "Analysis"
    if runBaseAnalysis:
        if(os.path.exists(basePath)):
            shutil.rmtree(basePath,True)
        os.makedirs(basePath,True)

    for category,features in allFeatures.items():
        categoryPath = basePath + seperator + category
        if runBaseAnalysis:
            os.makedirs(categoryPath, True)
        categoryAverageFileName = category + "Averages" + ".csv"
        featureAverages = []
        with open(categoryPath + seperator + categoryAverageFileName, 'w') as ca

            for feature in features:
                featureAverageFileName = feature + "Averages" + ".csv"

                featureSum = 0
                featureAtomCount = 0
                averages = []
                featurePath = categoryPath + seperator + feature
                if runBaseAnalysis:
                    os.makedirs(featurePath,True)
                with open(featurePath + seperator + featureAverageFileName,'w')
                    if runBaseAnalysis:
                        originalDirectoryPath = featurePath + seperator + "Origi
                        os.makedirs(originalDirectoryPath, True)
#hier die Analysen durchführen?
                        for documents,documentPath in originalDocuments.items():
                            analysis(documentPath,originalDirectoryPath,feature)

                        spunDirectoryPath = featurePath + seperator + "Spun"
                        os.makedirs(spunDirectoryPath, True)
                        for documents,documentPath in spunDocuments.items():
                            analysis(documentPath,spunDirectoryPath,feature)

                    featureAnalysisPath = featurePath + seperator + "Deviation"
                    os.makedirs(featureAnalysisPath,True)
                    #os.chdir(featureAnalysisPath)
                    documentIDs = []
                    for document in os.listdir(originalDirectoryPath):
```

```
                documentID = document.split("-")[0]
                documentIDs.append(documentID)
                #spunDocument = findFile(documentID, spunDirectoryPath)
                averages.append(analyseDifferences(originalDirectoryPath

            csvwriter = csv.writer(featureAverageCSV)
            csvwriter.writerow(["Document","Average"])
            for element in averages:
                csvwriter.writerow([str(documentIDs[featureAtomCount]),s
# Format: Document,Atom Score Average
                featureAtomCount += 1
                featureSum += element
            sys.stdout.write("Feature " + feature + " complete" + "\n")

        featureAverages.append(numpy.mean(averages))
        #featureAverages.append(featureSum/featureAtomCount) #ist hier d

    csvwriter = csv.writer(categoryAverageCSV)
    csvwriter.writerow(["Feature","Average"])
    counter = 0
    for element in featureAverages:
        csvwriter.writerow([str(features[counter]),str(element)])
# Format: Feature,Atom Score Average
        counter += 1
    sys.stdout.write("Category" + category + " complete" + "\n")

def findFile(documentId,directory):
    for document in os.listdir(directory):
        currentID = document.split("-")[0]
        if(currentID == documentId):
            return document

def analysis(documentPath,savePath,feature):
    global path
    global segmentation
    os.system("python command_line.py " + documentPath + " -f " + feature + " -s
" -l " + savePath) #funktioniert das so?
    #muss hier noch was hin?

def analyseDifferences(csvOriginal,csvSpun,name):
    differences = []
    with open(csvOriginal, 'r', newline='') as f:
        readerOriginal = csv.reader(f, delimiter=',')
        with open(csvSpun, 'r', newline='') as f:
            readerSpun = csv.reader(f, delimiter=',')
            firstRow = True
```

```
            for (row,spunRow) in zip(readerOriginal,readerSpun):
                #zum besseren Analysieren in Listen/Arrays zwischenspeichern?
                #spunRow = readerSpun.__next__()
                if(firstRow):    #header nicht auslesen
                    firstRow = False
                    continue

                #beide Reader sollten die gleiche Anzahl an Zeilen haben
                #rowElements = row.split(",")
                #spunRowElements = spunRow.split(",")
                #differences.append(rowElements[2] - spunRowElements[2]) #erste
                differences.append(float(row[2]) - float(spunRow[2]))
# erste Spalte: atom no, zweite Spalte: start index, dritte Spalte: Feature scor

    sum = 0
    atomNumbers = [i+1 for i in range(differences.__len__())]
    with open(name+".csv","w") as analysisCSV:
        csvwriter = csv.writer(analysisCSV)
        counter = 0
        csvwriter.writerow(["Atom No","Atom Score Difference"])
        for element in differences:
            csvwriter.writerow([str(atomNumbers[counter]),str(element)])
#Format: Atom No,Atom Score Difference
            counter+=1
            sum += element

    return numpy.mean(differences)  #float(sum) / float(differences.__len__())


def main():
    global seperator
    global path
    global basePath
    global runBaseAnalysis
    if (re.match("(l|L)inux", platform.platform(terse=True))):
        seperator = "//"
    path = sys.argv[1]  #Speicherort der zu analysierenden Dateien
    basePath = sys.argv[2] #Speicherort der Analyse
    if runBaseAnalysis:
        loadDocuments()
    createFileStructure()

if __name__ == '__main__':
    main()
```

B Preprocessing Script

```
import sys
import re
import platform
import os
import csv
import shutil
from collections import OrderedDict
from operator import itemgetter

seperator = "\\"

def processDocuments():
    global seperator
    documentPath = "D:\Patrick\Desktop\IPDAnalysis\documents"
#wird nicht auf Linux funzen
    proccessedDocumentsPath = "D:\Patrick\Desktop\IPDAnalysis" + seperator + "pr

    if (os.path.exists(proccessedDocumentsPath)):
        shutil.rmtree(proccessedDocumentsPath, True)
    os.makedirs(proccessedDocumentsPath, True)

    for document in os.listdir(documentPath):
        text = ""
        with open(documentPath + seperator +document,"r",encoding="utf-8") as un
            sys.stdout.write(document + "\n")
            text = unprocessedDocument.read()
        text.replace("\n","\n\n")
        with open(proccessedDocumentsPath + seperator + document,"w",encoding="u
            processedDocument.write(text)

def countWords():
    global seperator
    documentPath = "D:\Patrick\Desktop\IPDAnalysis\documents"
# wird nicht auf Linux funzen
    proccessedDocumentsPath = "D:\Patrick\Desktop\IPDAnalysis" + seperator + "wo

    if (os.path.exists(proccessedDocumentsPath)):
        shutil.rmtree(proccessedDocumentsPath, True)
    os.makedirs(proccessedDocumentsPath, True)

    wordCount = {}
    for document in os.listdir(documentPath):
        with open(documentPath + seperator + document,'r',encoding="utf-8") as t
```

```
            textString = text.read()
            tokens = re.split("(\s|\.|,|!|\?|:|;|\(|\)|-||\"|\[|\])",textString,
            for token in tokens:
                if token in wordCount:
                    currentCount = wordCount[token]
                    wordCount[token] = currentCount + 1
                else:
                    wordCount[token] = 1

        wordCount = OrderedDict(sorted(wordCount.items(), key=itemgetter(1)))

        with open(proccessedDocumentsPath + seperator + document + ".csv","w",en
            csvwriter = csv.writer(wordCountCSV)
            csvwriter.writerow(["Word", "Count"])
            for key,value in wordCount.items():
                csvwriter.writerow([key, value])


def analysis(documentPath,savePath):
    features = "coleman_liau_index automated_readability_index linsear_write_forr
    allFeatures = features + " punctuation_percentage average_word_length averag
    segmentation = "sentence"
    os.system("python command_line.py " + documentPath + " -f " + allFeatures +
" -l " + savePath) #funktioniert das so?
    #muss hier noch was hin?

def evaluationPreperation():
    global seperator
    documentPath = "D:\Patrick\Desktop\IPDAnalysis\documents"
# wird nicht auf Linux funzen
    proccessedDocumentsPath = "D:\Patrick\Desktop\IPDAnalysis" + seperator + "ev

    if (os.path.exists(proccessedDocumentsPath)):
        shutil.rmtree(proccessedDocumentsPath, True)
    os.makedirs(proccessedDocumentsPath, True)

    originalPath = proccessedDocumentsPath + seperator + "original"
    spunPath = proccessedDocumentsPath + seperator + "spun"

    os.makedirs(originalPath, True)
    os.makedirs(spunPath, True)

    for document in os.listdir(documentPath):
        fullDocumentPath = documentPath + seperator + document
        if(re.search("ORIG",document)):
            analysis(fullDocumentPath,originalPath)
```

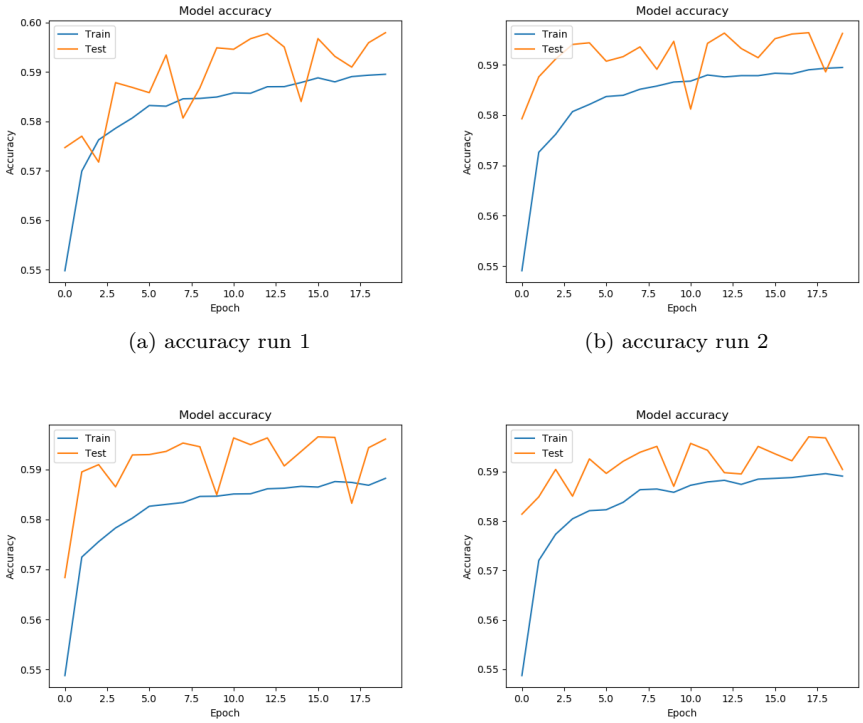
```
        else:
            analysis(fullDocumentPath,spunPath)

def main():
    global seperator
    if (re.match("(l|L)inux", platform.platform(terse=True))):
        seperator = "//"

    #processDocuments()
    #countWords()
    #evaluationPreperation()
    analysis("D:\Patrick\Desktop\IPDAnalysis\documents","D:\Patrick\Desktop\IPDA

if __name__ == '__main__':
    main()
```

C Averages

Feature	Avg Deviation - sentence	Scale	Category
punctuation_percentage	0.0021430845388008143		lexical
average_word_length	-0.12775205183455676	Englisch: 4.7	lexical
average_sentence_length	-1.04869249546782	Englisch: 15-20	lexical
average_syllables_per_word	-0.03275897035497655		lexical
alpha_chars_ratio	-0.003977218608954367		lexical
digit_chars_ratio	0.0013065522880773817		lexical
upper_chars_ratio	-0.0015641527448878646		lexical
white_chars_ratio	0.0027512263628052523		lexical
polysyllablcount	-0.37424283268728453	#words with #syllables > 3	lexical
coleman_liau_index	30.287042071273007	US school grade	readability
automated_readability_index	-0.5986997813065018	US school grade, designed for score 1-14	readability
linsear_write_formula	-0.9253524071006992	US school grade	readability
gunning_fog_index	-1.1198749832627757	years of Education required, designed for score 6-17	readability
dale_cahll_readability_score	-0.20917935370096816	years of Education required, designed for score 6-17	readability
smog_index	-0.5162859883256578	years of Education required, requires 30-sentences?	readability
flesch_reading_ease	3.6927064759281163	0-100	readability
flesch_kincaid_grade	-0.757595014220708	US school grade	readability
stopword_percentage	0.0020013215949394704		syntactic
avg_internal_word_freq_class	-0.08595015098922926		syntactic
avg_external_word_freq_class	-0.08588719434813716		syntactic
syntactic_complexity	-0.2034618749694078	modified index of syntactic complexity	syntactic
honore_r_measure	-95.09324493613033	share of unique words in the atom	vocabulary richniess
hapax_legomena	-0.0009794013311279258		vocabulary richniess
hapax_dislegomena	0.0014991019002989156		vocabulary richniess
simpsons_d	4.667064194601471e-05		vocabulary richniess
sichels_s	0.00015036020977823115		vocabulary richniess
brunets_w	-0.0655178397408864		vocabulary richniess
yule_k_characteristic	1.7712932140591138		vocabulary richniess
yule_i	-4.436428001723888		vocabulary richniess
type_token_ratio	5.6703974620550635e-05		vocabulary richniess

Abbildung 1: Overview of the feature averages with added context information. The calculation for the difference is original atom score - spun atom score.

D Neural Network Baseline



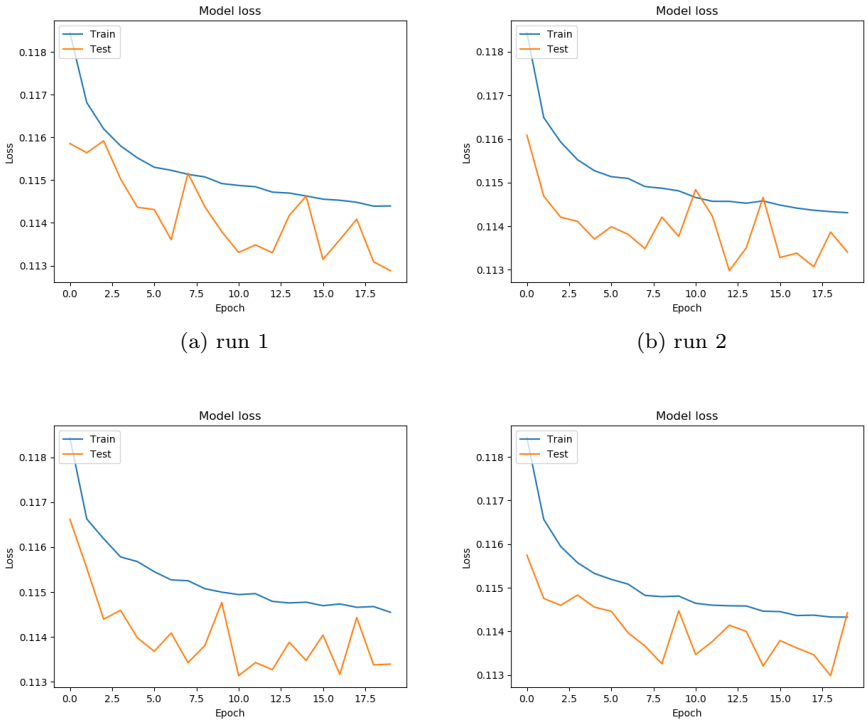
(a) accuracy run 1

(b) accuracy run 2

(c) accuracy run 3

(d) accuracy run 4

Abbildung 2: Neural Network baseline accuracy



(a) run 1

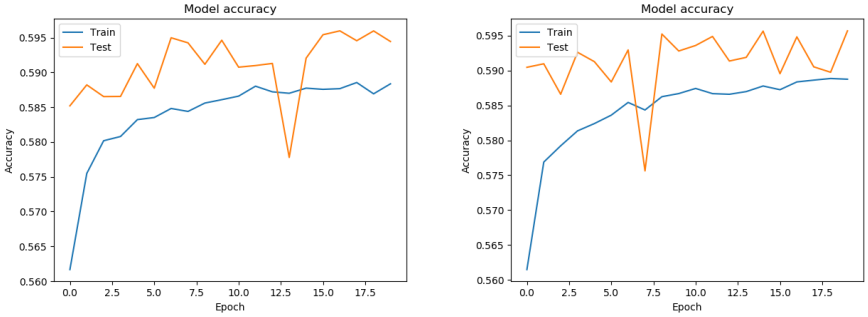
(b) run 2

(c) run 3

(d) run 4

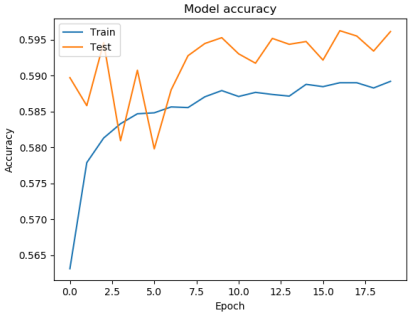
Abbildung 3: Neural Network baseline loss

E Neural Network Reduced Input

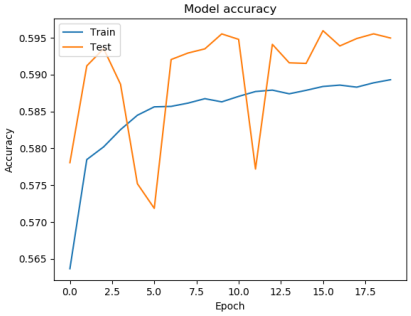


(a) accuracy run 1

(b) accuracy run 2

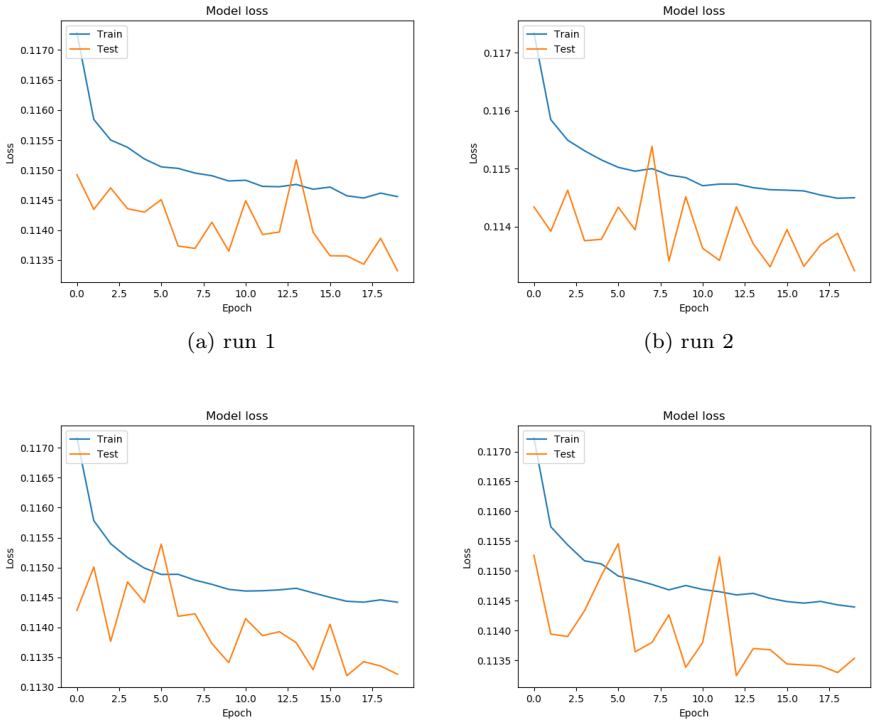


(c) accuracy run 3



(d) accuracy run 4

Abbildung 4: Neural Network baseline accuracy



(a) run 1

(b) run 2

(c) run 3

(d) run 4

Abbildung 5: Neural Network baseline loss