Besondere Lernleistung

Textklassifizierung mittels künstlicher Intelligenz mit verschiedenen maschinellen Lernalgorithmen, z.B. neuronalen Netzen und Naive Bayes-Algorithmen, am Beispiel eines Spamfilters für Text-Emails visualisiert mit und implementiert in JAVA

Amin Boyaala

Betreuer: Herr Faßbender

Städtisches Gymnasium Rheinbach

2019/2020

Inhaltsverzeichnis

Anmerkung

Vorab ist zu sagen, dass die Zitate, ohne Fußnoten sich immer auf die letzte Abbildung beziehen. Außerdem wurde bei den Zitaten, die ein UML Diagramm zitieren das „+“ bzw. das „-“ weggelassen, damit der Lesefluss nicht zu stark beeinflusst wird.

Part 0: Das Projekt

Diese Lernleistung hat als Thema die „Textklassifizierung mittels künstlicher Intelligenz mit verschiedenen maschinellen Lernalgorithmen, z.B. neuronalen Netzen und Naive Bayes Algorithmen, am Beispiel eines Spamfilters für Text Emails visualisiert mit und implementiert in JAVA“. Das Ziel dieses Projektes ist es einen Spamfilter in Java zu implementieren, der auf Basis maschineller Lernalgorithmen Emails in Spam oder Nicht-Spam, auch Ham genannt, zu klassifizieren.

Part 1: Theorie

Für die Umsetzung dieses Projektes habe ich als Basis zwei Modelle genutzt. Das erste Modell kommt aus der NLP („Natural Language Processing“) und ist das „Bag of Words“ Modell. Das zweite Modell kommt aus dem „Deep Learning“ und ist ein neuronales Netzwerk mit drei Hidden Layer. Das „Bag of Words“ Modell nutzt eine bestimmte Auswahl von Wörtern und schaut in einem Dokument nach ob und wie oft diese Wörter auftauchen, dabei interessiert sich dieses Modell nicht für die Relation, in der die Wörter zu einander stehen, sondern nur für deren Anzahl. Auf dieser Weise kann ein Dokument in einen Vektor von Zahlen umgeschrieben werden und vom Computer weiterverwendet werden. Um dieses Modell zu verdeutlichen dient die untere Tabelle:

	Dokument 1	**Dokument 2**	**Dokument 3**
Gewinn	3	0	2
Start-Up	2	0	3
Anlage	1	0	2
Überweisen	1	2	0
Hilfe	0	3	1

Die Auswahl von Wörtern, die diese Umsetzung des Modells benutzt, sind Gewinn, Start-Up, Anlage, Überweisen und Hilfe. Wenn man in den einzelnen Dokumenten die Wörter und ihre Anzahl, in der sie in dem Dokument auftauchen zählt, können die Dokumente in die Vektoren [3,2,1,1,0], [0,1,0,0,3] und [2,3,2,0,1] umgeschrieben werden. Nun kann der Computer mit diesen Dokumenten weiterarbeiten.

Als nächstes wird ein neuronales Netzwerk daraufhin trainiert, mithilfe solcher Vektoren, Dokumente in Spam und Ham einzuteilen. Dafür wird ein angebrachter „Bag of Words“ benötigt. Dieser wird mithilfe einer großen Menge von Emails erarbeitet. Das Programm soll eine große Menge von Spam Emails nehmen und aus diesen die an den meist aufkommenden Worten raussuchen, dabei ist es nicht wichtig wie oft die Wörter insgesamt auftauchen, sondern in wie vielen Emails das Wort auftaucht. Die Bedeutung der Anzahl eines Wortes aus dem „Bag of Word“-Modell pro Dokument kann vom neuronalen Netzwerk selbst erarbeitet werden.

Nachdem das Neuronale Netzwerk trainiert wurde, kann dieser benutzt werden, um Emails in Spam oder Ham klassifizieren zu können. Dazu muss man das gegeben Dokument, dass klassifiziert werden soll mittels des „Bag of Word“ Modells in einen Vektor umwandeln. Dieser Vektor, welcher die zu klassifizierende E-Mail darstellt, kann mittels des neuronalen Netzwerks klassifiziert werden.

Am Ende soll das Programm mittels einer importierten Datei in der Lage sein, das „Bag of Words“ Modell und das neuronale Netzwerk direkt nutzen zu können, ohne bei jedem Start diese Modelle neu anbauen zu müssen.

Part 2: Textverarbeitung

Das wichtigste Element in diesem Projekt sind die Textdaten mit der die Modelle trainiert werden. Für die benötigten Textdaten bzw. E-Mail-Daten habe ich den Enron Datensatz[1] ausgewählt. Die Daten müssen erst gereinigt werden und dann

[1] Quelle: “http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/“

in verwendbarer Form, als „Bag of Words“-Modell gebracht werden. Um die Daten in ein „Bag of Words“ Modell umzuwandeln, wird eine simple Analyse der Daten verwendet. Auf Ergebnis dieser Analyse wird ein dann ein „Bag of Words“ Modell angefertigt:

Part 2.1: Textreiningung

Das Reinigen der Textdaten als Vorbereitung für die Analyse der Daten wird mittels der Klassen „Create_Data“ und „Stopwords“ realisiert.

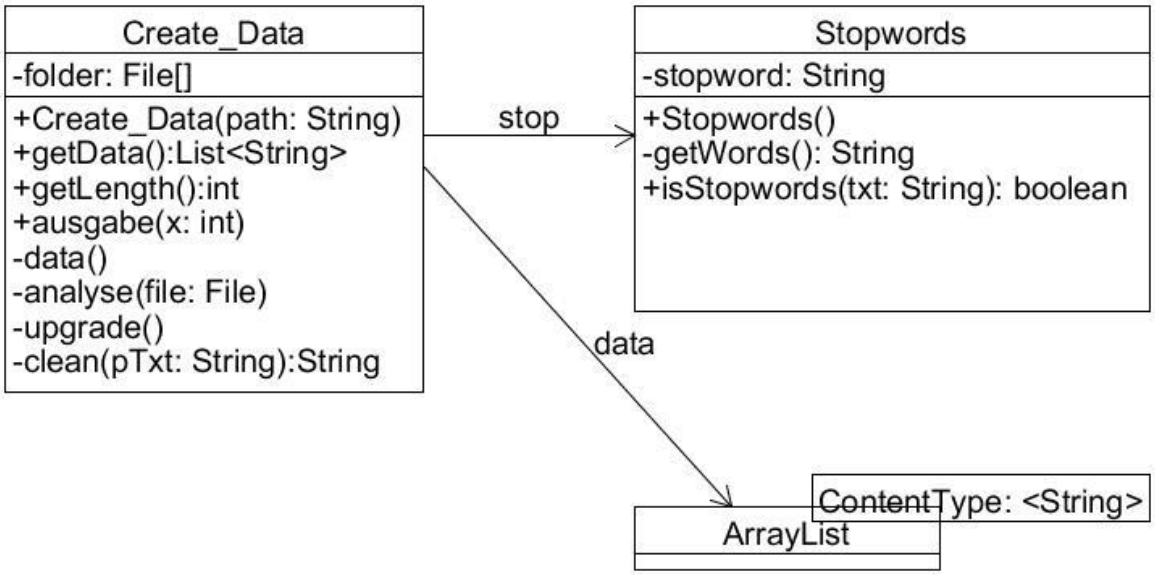


Abbildung 1

Create_Data ist eine Klasse, die darauf spezialisiert ist, die Enron Daten zu reinigen und abrufbereit zu speichern. Die Klasse speichert die Daten Ordnerweise ab, so dass jedes Element in der List „data“ eine E-Mail aus einem Ordner darstellt und als String speichert. Ein Objekt der Klasse Create_Data speichert einen Ordner, voraussichtlich voller Emails und speichert sie in angebrachter Form ab:

Der Pfad zum Ordner wird als Parameter der Klasse Create_Data über den Konstruktor übergeben. Beim Aufrufen des Konstruktor wird jede Datei in dem übergebenen Ordner, mittels der „analyse(file: File)“ Methode eingelesen und dann nach dem Reinigen, realisiert durch die „clean(pText: String):String“ Methode der List data hinzugefügt.

Beim Reinigen wird der Text jeder Datei erst in Zeilen eingeteilt und zeilenweise abgearbeitet. Ist die jeweilige Zeile für die weitere Textverarbeitung wichtig wird sie beibehalten, sonst wird sie verworfen. Die Enron Datensätze benutzen die Windows Escape Sequenz, um Zeilen zu trennen und haben am Anfang der Zeilen Schlüsselwörter, wie zum Beispiel „Subject:“ oder „from:“. Diese Informationen sind wohl für die E-Mail an sich essenziel, aber für die Textverarbeitung auf, die in diesem Projekt abgezielt wird, störend. Dieses Projekt soll mittels der Inhalte der Emails in der Lage sein eine Spam-E-Mail zu erkennen. Methoden wie dem Erkennen von Spam Emails mittels des Abgleiches von bereits bekannten fragwürdigen E-Mail-Adressen oder fragwürdigen Betreffen, ist effizient, aber nicht abgezielt. Zusätzlich wird der Text in dieser Methode so normalisiert, dass die E-Mail nur aus Wörtern aus Kleinbuchstaben und Leerzeichen besteht. Satzzeichen, Hashtags usw. werden entfernt.

Anschließend wird die „upgrade()“ Methode die Daten in der Liste erneut normalisiert, hinzu kommt das dieses Mal Apostrophe ebenfalls entfernt werden und die „clean(pTxt: String):String“ Methode wird aufgerufen. In der data Methode wird mittels der Stopwords Klasse Stoppwörter der englischen Sprache und Schlüsselwörter aus der Sprache HTML aus dem String aus der List data entfernt. Um dies zu realisieren dient die Stopwords Klasse. Diese Klasse baut auf Basis einer „Stopword.txt“[2] einen String „stopword: String“ der die Stoppwörter und HTML Begriffe speichert. Die Klasse bietet die „isStopword(txt: String):boolean“ Methode an, welche zurückgibt ob ein übergebenes Wort ein Stoppwort ist oder nicht. Mittels dieser Klasse entfernt die data Methode alle Stoppwörter und HTML Begriffe aus den Daten in der List data.

Abschließend können die Daten und die Länge der Daten mittels der Methoden „getData():List<String>“ und „getLength():int“ auch von anderen Klassen abgerufen werden.

[2] Basiert auf: „http://www.lextek.com/manuals/onix/stopwords1.html“, Modifizierte Version: siehe Anhang

Part 2.2: Analyse

Für das „Bag-of-words“ Modell ist es notwendig zu wissen wie oft welches Wort in einer E-Mail vorkommt, deshalb muss das Programm mittels der Emails eine Statistik wie oft jedes Wort vorkommt erheben. Dies wird durch die Klassen „BagOfWords“, „Word“ und „Data“ realisiert:

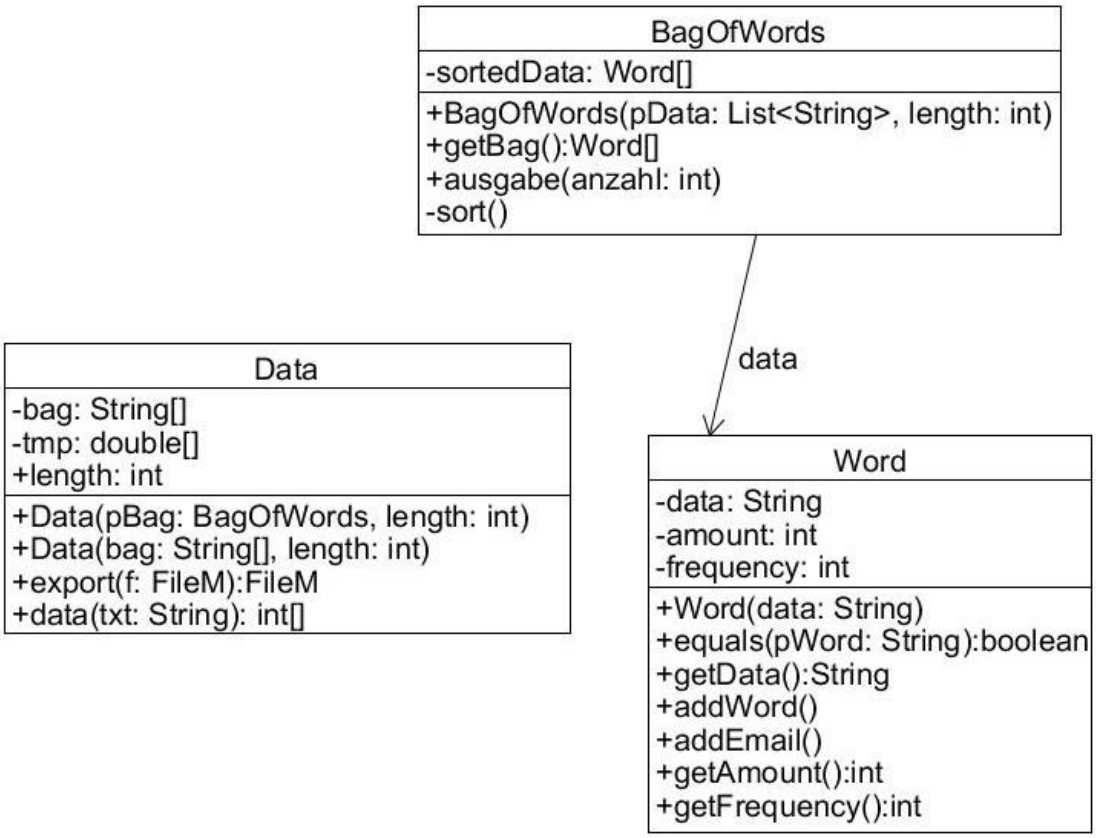


Abbildung 2

Die BagOfWords Klasse analysiert die von der Create_Data Klasse erstellten Daten und erschafft mittels der Word Klasse eine Liste mit den häufigsten Wörtern in den übergebenen Daten. Die Word Klasse speichert ein Wort und dessen Anzahl und Häufigkeit in den Daten. Dabei ist die Anzahl „amount: int“ in dem Fall die Häufigkeit des Wortes in allen Emails und die Häufigkeit „frequency: int“ die Anzahl der Emails in der das Wort vorkommt. Die Klasse speichert diese Werte und erhöht sie bei Bedarf. Die BagOfWords Klasse nutzt eine List von dem ContentType Word namens „data“, welche alle Wörter von

allen Dokumenten speichert und das Words Array „sortedData: Word[]“, welches dieselben Wörter sortiert speichert.

Die Initialisierung der BagOfWords Klasse verlangt eine List von ContentType String „[…]pData: List<String>“ und ein Integer „[…]length: int“. Bei der Initialisierung nimmt die Klasse so viele Strings aus der der übergebenen Liste pData wie der Anzahl von dem length Integer entsprechend und speichert diese in einer temporären Liste ab. Danach wird jedes Wort aus jedem String/Dokument der temporären Liste gelesen und mit der bisherigen Liste von Wörtern „data“ abgeglichen. Ist es ein neues Wort wird es hinzugefügt. Bei jedem Fund eines Wortes in einer neuen E-Mail wird die Häufigkeit erhöht. Anschließend wird erneut die temporäre Liste traversiert und es wird jedes Wort gezählt und auf dieser Weise wird die Anzahl eines Wortes ermittelt. Ist die Anzahl und die Häufigkeit jedes Wortes gefunden wird die Liste data mittels der „sort()“ Methode sortiert und die „sortedData: Word[]“ Liste wird ermittelt. Beim Sortieren wird die List data erst in ein temporäres Array umgewandelt. Dieses temporäre Array wird mittels des „BubbleSort“ sortiert und die List sortedData wird dem sortierten temporären Array gleichgesetzt. Mittels der „getBag(): Word[]“ kann dann das sortierte Array abgerufen werden.

Mit diesem sortierten Array arbeitet die Data Klasse dann weiter. Die Data Klasse realisiert die Arbeit mit dem Bag-of-words Modell. Die Data Klasse kriegt ein Objekt der BagOfWords Klasse „[…]pBag: BagOfWords[…]“ und ein Integer der die gewünschte Länge für das Bag-of-words Modell angibt „[…]length: int)“, übergeben. Mit diesen Daten baut sich die Data Klasse den finale Bag-of-words und speichert diesen in ein String Array „bag: String[]“ und die Länge des Beutels/Arrays in ein Integer „length: int“ ab. Beim Kreieren des Bag-of-words Modells übernimmt die Data Klasse in der Initialisierung den ersten Worten aus dem sortierten Array des Bag-of-words Modells, entsprechend der Anzahl des übergebenen Wertes in dem Parameter length. Dieses Array ist private gespeichert und kann somit nicht von anderen Klassen zugegriffen werden. Wenn eine Klasse das Bag-of-words Modell benutzen will, übergibt es der Data Klasse Methode data „data(txt: String):int[]“ den benötigten Text über.

Der Text wird dann in ein Bag-of-words Modell, wie nach Part 1 umgewandelt und als ein Integer Array zurückgegeben. Das Integer Array enthält die Anzahl der Wörter des Bag-of-words Modells in dem übergebenen Text (vgl. Tabelle 1, Part 1: Theorie).

Die Data Klasse enthält einen zweiten Konstruktor, einen Kopierkonstruktor: „Data(bag: String[], length: int)“. Dieser setzt den eigenen Bag-of-Words und dessen Länge gleich den übergebenen Werten. Zusammen mit der export Methode „export(f: FileM):FileM“, die ein übergebenes Objekt der FileM Klasse mit den eigenen Daten ergänzt und dann zurückgibt, bildet der Kopierkonstruktor die Funktion das Bag-of-words Modell in eine Datei auszulagern und später auch wieder zu importieren.

Nun da Textdateien vektorisiert und somit auch ausgewertet werden können, kann jetzt auch der Motor dieses Projektes trainiert und benutzt werden:

Part 3: Neuronales Netzwerk

Part 3.1: Theorie

Ein neuronales Netzwerk („Neural Network“, NN) oder auch künstliches neuronales Netzwerk („Artificial Neural Network“, ANN) genannt, ist ein maschineller Lernalgorithmus, welches vom menschlichen Gehirn inspiriert wurde.

Maschinelle Lernalgorithmen sind Algorithmen aus dem maschinellen Lernen, welches ein Unterbegriff der künstlichen Intelligenz ist. Maschinelles Lernen ist, wenn ein Programm sich eine bestimmte Kompetenz bzw. Funktion selbst beibringt. In diesem Projekt soll das Programm sich selbst beibringen E-Mails nach Spam und Ham zu klassifizieren. Nachdem mittels einer Analyse von Emails das Bag-of-words Modell gebaut wurde, wird auf Basis des Bag-of-words Modells ein neuronales Netzwerk darauf trainiert E-Mails zu klassifizieren.

Die Vorgehensweise bei maschinellen Lernalgorithmen besteht darin auf Basis von Daten eine Vorhersage zu machen, ist diese Vorhersage falsch korrigiert sich der Algorithmus indem er eigene Parameter anpasst, um näher an die richtige

Vorhersage zu kommen. Nach genügendem Training kann der Algorithmus sichere und passendere Vorhersagen machen. Nun kann der Algorithmus auch zu neuen Daten sichere Vorhersagen machen, obwohl er auf diesen nicht trainiert wurde. Der Zweck von solchen Lernalgorithmen liegt darin Zusammenhänge zu erkennen, die der Mensch nicht erkennen kann. Durch genügend Training wird der Algorithmus dazu gebracht diese Zusammenhänge zu erkennen und auf Basis dieser Vorhersagungen zu machen die Menschenähnliche Intelligenz benötigen würde, deshalb künstliche Intelligenz. Die Zusammenhänge in diesem Projekt bzw. in diesen Daten ist der Zusammenhänge zwischen der Anzahl bestimmter Wörter (Bag-of-words Modell) und der Klassen Spam und Ham.

Input: [5,6,1,0,2,5,8,4,9,1] / Bag-of-words

Output: Klasse 1: Spam: 87%, Klasse 2: Ham: 13%

Um diesen Zusammenhang zu erkennen wird ein neuronales Netzwerk benutzt, dieses ist ein überwachter Lernalgorithmus. Überwachtes maschinelle Lernen ist, wenn ein Algorithmus neben den Daten auch die richtige Antwort bzw. die richtige Klasse übergeben bekommt, auf dieser Weise kann der Algorithmus auf ein bestimmtes Ergebnis „hinarbeiten“. Neben dem überwachten Lernen gibt es ebenfalls das unüberwachte und das bestärkende maschinelle Lernen:

Beim unüberwachten maschinellem Lernen erhält der Algorithmus nur die Daten. Muster, Strukturen und Ausreißer müssen vom Algorithmus selbst gefunden werden. Beim bestärkenden Lernen erhält der Algorithmus nach seiner Vorhersage Belohnungen oder Strafen und passt sich nach diesen an.

Dank des Enron Datensatzes und dem Bag-of-words Modell sind sowohl Daten als auch die Klassen bekannt (Der Enron Datensatz speichert Ham und Spam geteilt). Deshalb bietet sich ein überwachter Lernalgorithmus, in diesem Fall ein neuronales Netzwerk am besten an.

Part 3.2: Aufbau und Funktionsweise neuronaler Netzwerke

Neuronale Netzwerke sind von Gehirnen inspirierten Algorithmen, sie bestehen ebenfalls aus Neuronen, die mittels Synapsen verbunden werden. Bei diesem

Algorithmus wird ein Netzwerk aus solchen Neuronen und Synapsen gebaut, die Neuronen werden dabei mittels der Unterklassen von Node und die Synapsen mittels der Edge Klasse dargestellt.:

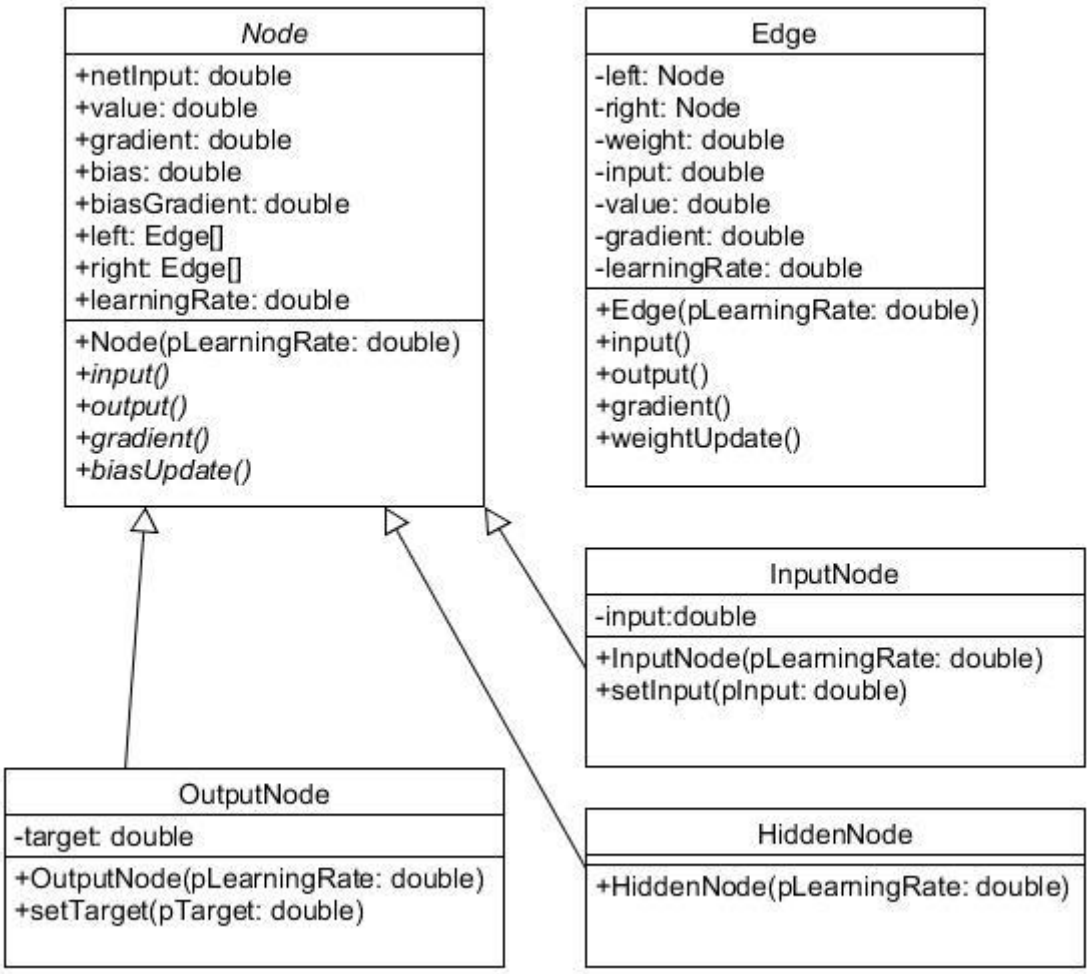


Abbildung 3: Ohne Getter und Setter - Methoden

Ein neuronales Netzwerk hat drei Schichten (/Layers) von Nodes den Input Layer, den Hidden Layer und den Output Layer. Jede Schicht hat eine beliebige Anzahl an Node Objekten, wobei jede Kategorie von Schicht eine andere Aufgabe hat. Jede Node eines Layers ist mit jeder Node der nächsten Schicht über ein Objekt der Edge Klasse verbunden. Da die Edge Klasse verschiedene Kategorien von Nodes verbunden muss, werden die Nodes des neuronalen Netzwerkes mit einer Modellierung mittels einer abstrakten Node Oberklasse und den jeweiligen Unterklassen realisiert. Dadurch das die abstrakte Node

Oberklasse alle notwendigen Methoden und Variablen vorgibt, ist die Edge Klasse in der Lage unterschiedliche Nodes verbinden.

Das neuronale Netzwerk von diesem Projekt benutzt einen Input Layer, drei Hidden Layer und einen Output Layer. Es folgt dieser Form:

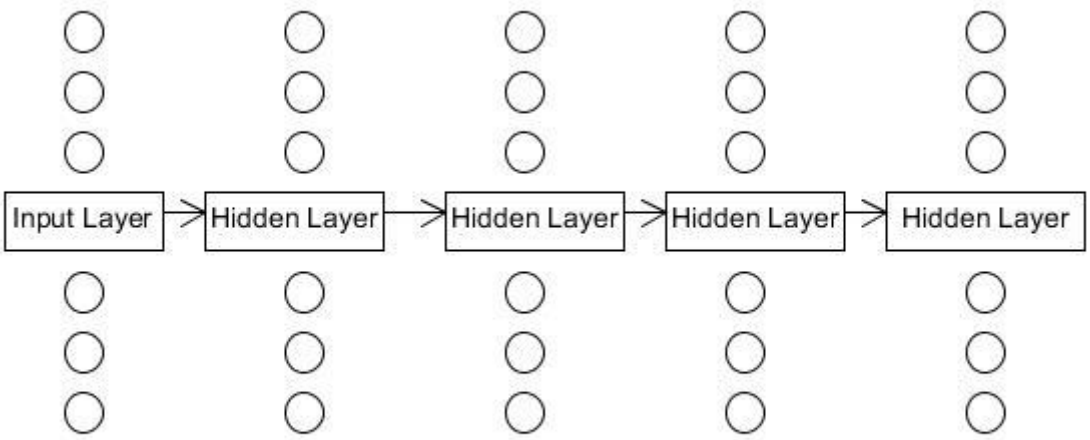


Abbildung 4

Der Input Layer empfängt die Daten, indem jede Input Node des Input Layers einen Datenwert bekommt und „speichert“ „setInput(pInput: double)“. In diesem Projekt würde eine Input Node für ein Wort aus dem Bag-of-words Modell zuständig sein und als Input die Anzahl dieses Wortes in einem Dokument bekommen. Der Input Layer gibt die übergebenen Datenwerte an den nächsten Layer weiter, indem jede Node bzw. Input Node des Input Layers sein Datenwert an jede Node des nächste Hidden Layers weitergibt.

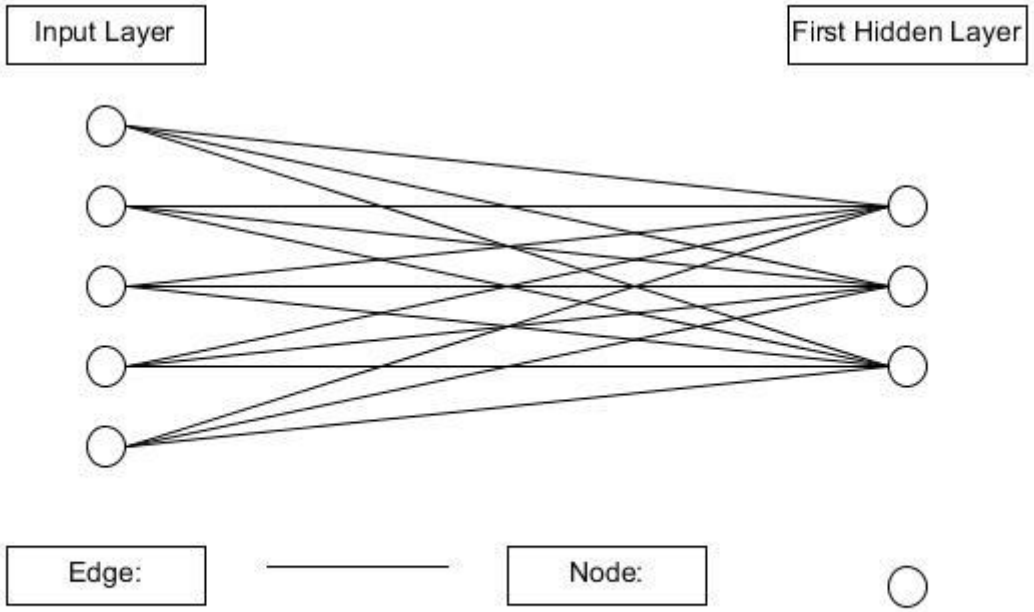


Abbildung 5

Bei dieser Weitergabe geht der Datenwert von der Input Node zur Edge und dann von der Edge zur nächsten Hidden Node. Jede Edge besitzt ein individuelles Gewicht „weight: double“, dieser wird mit dem übergebenen Wert der Input Node multipliziert und dann an die Hidden Node weitergegeben. Dies ist einer der Inputs eines Hidden Nodes, der gesamte Input einer Hidden Node, heißt Net Input „netInput: double“ und dieser entspricht der Summe aller Inputs der vorherigen Input Nodes plus des Bias „bias: double“. Der Bias ist eine zusätzliche Node deren Output immer 1 ist. Jeder Layer, außer der Input Layer besitzt eine zusätzliche Bias Node welche über eine Edge mit jeder Node des jeweiligen Layers verbunden ist. Da der letztendliche Output, der an der Hidden Node ankommt, 1 * Weight (der Edge) ergibt, wird der Bias nicht als eigenes Objekt gespeichert, sondern als double Variable. Diese Double Variable realisiert den Bias Node und den zugehörigen Edge Objekt.

$$Net\ Input = \sum_{i=1}^{Nodes\ des\ vorherigen\ Layers} (Output_i * Weight_i) + Bias$$

Nachdem die Hidden Node sein Net Input mittels der input Methode „input()“ erhalten hat, berechnet sich die Hidden Node den eigenen Output „output()“. Dieser wird mit einer Aktivierungsfunktion a berechnet:

Output = a(Net Input)

Eine Aktivierungsfunktion ist eine Funktion, welche den Net Input in einen Wert zwischen a und b umwandelt. In diesem Projekt wird als Aktivierungsfunktion die Sigmoidfunktion benutzt:

$$Sig(x) = \frac{e^x}{1 + e^x}$$

Die Sigmoidfunktion wandelt jeden Wert in einen Wert zwischen 0 und 1 um und hat die Form einer S-Kurve. Diese Aktivierungsfunktion wurde gewählt, da in diesem Klassifizierungsfunktion nur ein Wert als Output genügt. Wie sehr ist sich das Programm sicher, dass das gegeben Vektorisierte Dokument Spam ist. Dabei wird das Dokument bei Sig(Dokument) = 0 als Ham und bei Sig(Dokument) = 1 als Spam betrachtet. In diesem Projekt wird also für die Lösung dieses Klassifizierungsproblem nur Werte zwischen 0 und 1 benötigt/benutzt. Deshalb benutzt jede Node die gleiche Sigmoidfunktion als Aktivierungsfunktion und dies wird mittels der Function Klasse realisiert

Die Function Klasse besitzt nur eine statische Methode „activation(pX: double):double“, welche die Aktvierungsfunktion, also in dem Fall die Sigmoidfunktion umsetzt.

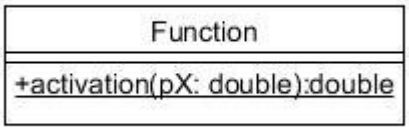


Abbildung 6

Der Output einer Node wird mittels des Net Inputs und der Aktivierungsfunktion berechnet und wird dann an jede Node des nächsten Layers geschickt. Der gleiche Prozess passiert auch im nächsten Hidden Layer. Der Net Input einer Hidden Node des zweiten Hidden Layers entspricht der Summe aller Outputs der

Hidden Nodes des ersten Hidden Layers multipliziert mit der jeweiligen Weight der Edge der die Nodes miteinander verbindet plus den Outputs des Bias Nodes der für den zweiten Hidden Layer zuständig ist. Der Output einer Hidden Node des zweiten Hidden Layers ist ebenfalls die Aktivierungsfunktion von dem Net Input. Das gleiche Spiel findet auch vom zweiten zum dritten Hidden Layer ab und genauso vom dritten Hidden Layer zum Output Layer (fünfter Layer).

Der Output Layer ist für den Output des neuronalen Netzwerkes zuständig. Genauso wie jede Input Node des Input Layers dafür zuständig ist ein Datenwert aus den eingegebenen Daten zu empfangen ist der Output Layer dafür zuständig die Vorhersage des neuronalen Netzes auszugeben. In diesem Klassifizierungsproblem wird nur ein Wert benötigt, wie sicher ist sich das neuronale Netzwerk, das der übergebene Vektor eine Spam-E-Mail darstellt. Deswegen hat das neuronale Netzwerk dieses Projektes nur ein Output Node im Output Layer. Der Net Input de Output Node entspricht der Summe der Outputs der Hidden Nodes im dritten Hidden Layer multipliziert mit der Weight der Edge der die jeweilige Hidden Node mit der Output Node verbindet plus den Output des Bias. Der Output des neuronalen Netzwerkes entspricht dann dem Ergebnis der Aktivierungsfunktion von dem Net Input.

Der Input und Output einer Node wird jeweils mit der Input Methode „input()“ und der Output Methode „output()“, die von der Node Oberklasse geerbt wurde, realisiert. Für diese Umsetzung ist ebenfalls die Edge Klasse wichtig. Neben den eigenen Weight speichert die Edge Klasse die beiden Nodes die sie verbindet „left: Node“ und „right: Node“ als Variablen. Die Edge Klasse besitzt ebenfalls eine Input und Output Methode. Und genauso wie die Edge Klasse besitzen die Node Klassen von der Node Klasse vererbte Left und Right Array „left: Edge[]“ und „right: Edge[]“. Sowohl bei der Node Klasse als auch bei der Edge Klasse funktioniert die Input Methode indem sie den Net Input aus den vorherigen Objekten, also diejenige die über den Namen „left“ gespeichert werden, „einsammelt“. Nur die Input Node Objekte aus dem Input Layer sind eine Ausnahme mit ihrer setInput Methode „setInput(pInput: double)“ ,da diese Objekte keine „Vorgänger“ bzw. keine Objekte haben von denen sie

„einsammeln“ könnten. Bei dem Einsammeln durch die Input Methode übernimmt die Edge Klasse den Output des Left Objektes und speichert diesen als eigenen Input ab „input: double“. Da die Node Klassen Mehrere Objekte haben, von denen sie einsammeln haben sie einen Net Input, bei ihrer Input Methode traversieren sie das Left Array und addieren den Output der Edge Objekte plus den eigenen Bias Wert. Es muss nicht noch extra mit der Weight der Edge Objekte multipliziert werden, da dies schon in der Edge Klasse passiert: In dieser Umsetzung eines neuronalen Netzwerkes wird der Input von den vorherigen Objekten eingesammelt werden und als input/netInput gespeichert werden, der Output muss deswegen nicht an die nächsten Objekte weitergegeben werden. Die Output Methode berechnet den Output der Node bzw. der Edge und speichert diesen unter der Value Variable „value: double“ ab. Diese kann dann von anderen Klassen mittels der „getOutput():double“ Methode (nicht in Abb. 3 abgebildet, siehe UML Diagramm im Anhang) eingesammelt werden.

Auf dieser Weise können Daten in dieser Umsetzung eines neuronalen Netzwerkes traversieren. Zu beachten ist, dass diese Traversierung und Berechnung schrittweise und schichtenweise ablaufen muss. Dies und der Lernprozess des neuronalen Netzwerkes wird von der Neural Network Klasse realisiert. Diese organisiert und trainiert die bisher erklärten Elemente/Klassen:

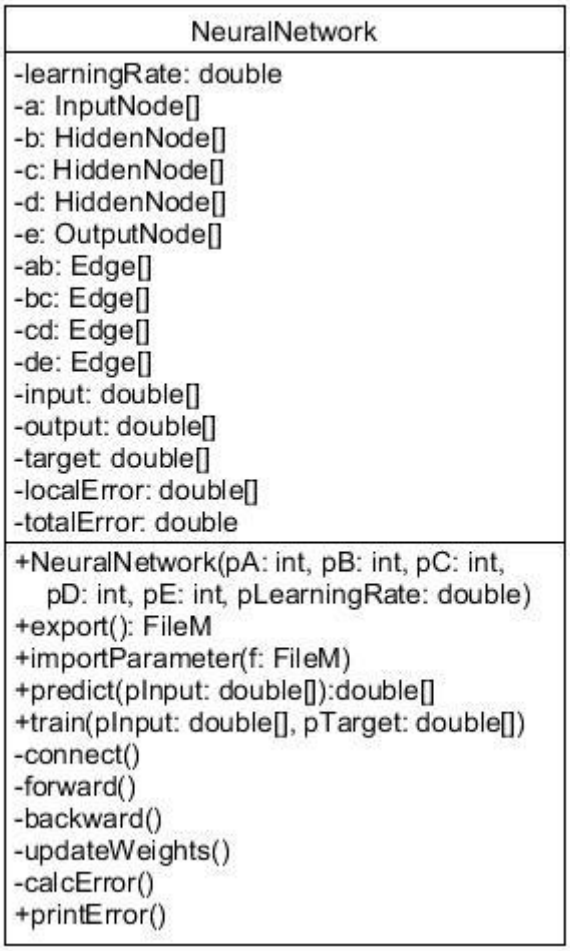


Abbildung 7

Die NeuralNetwork Klasse realisiert das neuronale Netzwerk mittels der Node, Edge und Function Klassen. Die einzelnen Schichten werden mittels Arrays von Nodes umgesetzt, die jeweils mit a, b, c, d und e beschriftet sind. Die Schichten b-d sind die Hidden Layers und a und b sind jeweils der Input und Output Layer.

Die Verbindungen zwischen den Layern, also die Edge Objekte, werden mittels zweidimensionaler Arrays der Edge Klasse geschaffen. Der erste Index bestimmt die linke Node auf die das Edge Objekt verweist und der zweite Index verweist auf das rechten Node Objekt. Ein zweidimensionales Array repräsentiert dabei alle Edge Objekte zwischen zwei Schichten von Node Objekten. Der Input für das Netzwerk wird in dem Input Array „input: double[]“ und der Output, also die Vorhersage des Netzwerkes wird in dem Output Array „output: double[]“ gespeichert. Das Target Array „target: double[]“ speichert das gewünschte Ergebnis zu einem bestimmten Input. Der lokale Fehler „localError: double[]“ speichert die quadratischen Abweichungen von den ausgegebenen Outputs des neuronalen Netzwerkes von dem gegeben Target. Der totale Fehler entspricht der mittleren quadratischen Abweichung (Mean Squared Error) von dem Output und dem Target:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(target_i - output_i)^2$$

Die Lernrate „learningRate: double“ ist ein Wert zwischen 0 und 1 der sagt wie viel das neuronale Netzwerk in jeder Trainingseinheit „lernt“. Dieser Wert muss jedem Objekt innerhalb des neuronalen Netzwerkes (Edge und Nodes) übergeben werden, da das „Lernen“ in jedem Element des Netzwerkes auftritt.

Die Parameter die für die Initialisierung dieser NeuralNetwork Klasse nötig sind, sind die Anzahl von Nodes in den verschiedenen Schichten und die gewünschte Lernrate: „NeuralNetwork(pA: int, pB: int, pC: int, pD: int, pE: int, pLearningRate: double)“. Im Konstruktor wird die Lernrate übernommen und mit ihr werden alle Nodes und Edge Objekte erstellt. Danach wird die Connect Methode „connect()“ aufgerufen, welche allen Edge Objekten ihre jeweiligen Node Objekte und allen Node Objekten ihre jeweiligen Edge Objekte zuweist. Die grundlegendste Methode der NeuralNetwork ist die Predict Methode „predict(pInput: double[]):double[]“ welche Daten übernommen bekommt, diese als Inpupt speichert, dann den schon beschriebenen Prozess der Datenverarbeitung aufruft und daraus folgenden Output zurückgibt. Der Prozess der Datenverarbeitung wird durch die Forward Methode „forward()“ realisiert. Diese Methode arbeitet, mittels der Edge und Node Klassens Input und Output Methoden aller Layer und Schichten von Edge und Node Objekten nacheinander ab. Dadurch traversieren die Daten das Netzwerk und der Input wird zum Output transformiert. Im Sachverhalt bedeutet das, dass der Bag-of-words dem neuronalen Netzwerk als Input übergeben wird und der Output die Vorhersage ist, ob es sich es um eine Spam E-Mail handelt oder nicht:

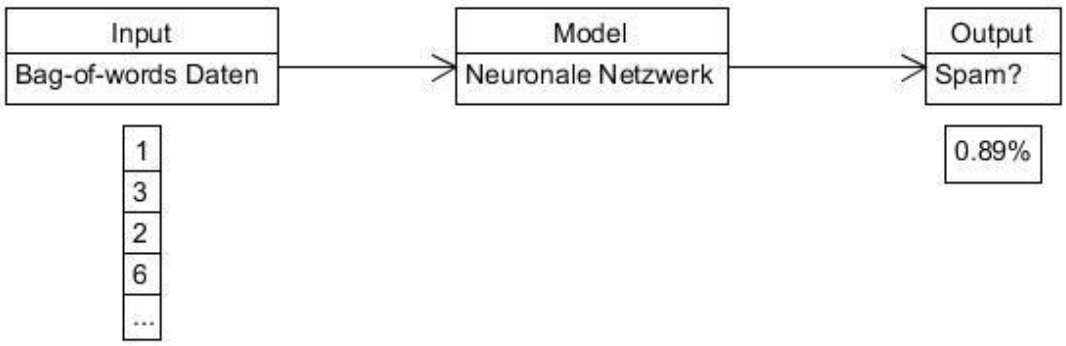


Abbildung 8

Bei der Initialisierung der Edge und Node Klasse wird der Weight der Edge und der Weight der Edge des Bias zufällig gewählt. Deshalb wird das Ergebnis einer Vorhersage mittels eines neu erstellten neuronalen Netzwerkes nichtssagend sein. Es ist deswegen nötig das neuronale Netzwerk zu trainieren und so die einzelne Weights des Netzwerks so zu ändern, dass das neuronale Netzwerk aussagekräftige Vorhersagen treffen kann:

Part 3.3: Der Lernprozess

Neben der Predict Methode gibt es ebenfalls die Train Methode „train(pInput: double[], pTarget: double[])“. Diese Methode erhält genau wie bei der Predict Methode die Daten als Input „[…]pInput: double[]“ übergeben, darüber hinaus erhält sie den Target, also das richtige Ergebnis „[…]pTarget: double[]“ übergeben. Die Methode speichert den Input und das Target ab und ruft dann die Forward Methode ab, welche dann den Input also die Daten verarbeitet. Ist der Output berechnet wurden ruft die Methode die Fehlerrechnung Methode „calcError()“ auf. Diese berechnet den lokalen und den totalen Fehler mittels des mittleren quadratischen Fehler MSE aus. Ist nun auch der Fehler gefunden wird die Backward Methode „backward()“ aufgerufen, in dieser Methode wird das „Lernen“ stattfinden. Sie ist das Gegenteil von der Forward Methode, da die Backward Methode das Netzwerk von rückwärts, also von dem Output Layer zum Input Layer traversiert und entsprechend des berechneten Fehlers die Weights der Edge Objekte anpasst.

Die Frage, die hier gestellt wird, ist „wie stark beeinflusst eine Weight/ ein Gewicht einer Edge den Fehler“. Im Grunde ist ein neuronales Netzwerk eine sehr große Funktion mit sehr vielen Parametern und diese Parameter sind die Weights / die Gewichte der Edges. Um die Frage, wie sehr ein Gewicht ein Gewicht das Ergebnis beeinflusst, beantworten zu können benutzt man die partielle Ableitung.

$$\frac{\partial Fehler\ des\ Netzwerkes}{\partial Gewicht\ einer\ bel.Edge}$$

Die Idee dahinter ist sich den Fehler als eine Funktion vorzustellen, wobei der Fehler auf der y-Achse und der Wert eines Gewichtes einer Edge auf der x-Achse abgebildet ist. Je nachdem wie sich die x-Stelle ändert, so ändert sich auch der y-Wert also der Fehler. Das Ziel ist es einen niedrigen Fehler zu bekommen, also muss man die x-Stelle so ändern, dass sich der y-Wert senkt. Um dies zu erreichen wird die Ableitung an der x-Stelle gesucht, in der sich das Gewicht aktuell befindet. Ist die Ableitung und somit die Steigung gefunden, kann man herausfinden in welche Richtung (+/-, rechts/links auf der x-Achse) sich der Wert des Gewichtes der Edge bewegen muss, damit sich der Fehler weiter verringert. Deshalb wird das Gewicht einer Edge minus der (partiellen) Ableitung für dieses

Gewicht. Ist die Ableitung schon negativ bzw. sinkt der Graph wird durch die Minusrechnung der Wert des Gewichtes größer, wodurch der Fehler kleiner wird. Ist die Ableitung positiv bzw. steigt der Graph an wird der Wert des Gewichtes durch die Minusrechnung kleiner wodurch der Fehler ebenfalls kleiner ist. Bei dieser Minusrechnung kommt die Lernrate ins Spiel. Man zieht nicht die komplette Ableitung ab, sondern multipliziert diesen erst mit der Lernrate. Ist die Lernrate nahe 1, so wird auch mehr abgezogen und das neuronale Netzwerk „lernt mehr dazu“. Ist die Lernrate nahe 0, so wird wenig abgezogen und das Netzwerk „lernt wenig dazu“. Dies ist eine Trainingseinheit und muss daher öfters durchgeführt werden. Und diese Methodik mittels der (partiellen) Ableitung, auch Gradient genannt, den (lokalen) Minimum einer Funktion zu finden, heißt „Gradient descent“ oder auch Gradientenabstiegsverfahren genannt. (Die folgende Abbildung ist nur schematisch)

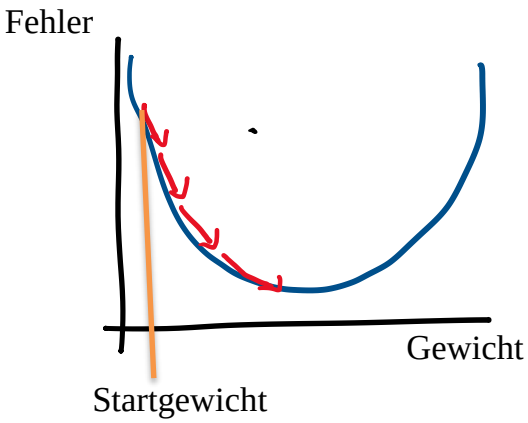


Abbildung 9 Gradientenabstiegsverfahren

Da ein neuronales Netzwerk eine sehr große verschachtelte Funktion ist, muss für die Berechnung der partiellen Ableitung bzw. des Gradienten die Kettenregel angewandt werden. Die Idee dahinter ist nicht zu schauen wie ein Gewicht den totalen Fehler beeinflusst, sondern zu schauen wie ein Gewicht das nächste Element der in der Vorwärtsbewegung beeinflusst und dann zu schauen wie dieses Element das nächste Element beeinflusst usw. bis man an den totalen Fehler erreicht hat. Dies ist mittels einer objektorientierten Umsetzung gut anwendbar, da jedes Objekt seinen eigenen Gradienten errechnen kann indem er den Gradienten von seinen Nachbarn abrufen kann. Dies wird mittels den Variablen „biasGradient: double“ in der Node Klasse und der „gradient: double“ in der Edge Klasse und deren jeweiligen Getter und Setter Methoden (nicht in Abb.3 abgebildet) realisiert. Zu Berechnung dienen die Gradient Methoden in den beiden Klassen „gradient()“ und zum Updaten der Gewichte dienen die Update Methoden „biasUpdate()“ in der Node Klasse und die „weightUpdate()“ in der Edge Klasse. Genau wie die Forward Methode arbeitet die Backward Methode Schichtenweise die Node und Edge Objekte ab, um die Gradienten

auszurechnen. Für das Updaten der Gewichtungen nutzt die Backward Methode die Update Methode „updateWeights()“, welche die Edge und Node Objekte traversiert und ihre jeweiligen Update Methoden aufruft.

Die Methodik der Forward und Backward Methode heißt Forward-Propagation bzw. Back-Propagation.

Für das Auslagern in eine Datei besitzt die NeuralNetwork Klasse die Import und Export Methode „importParameter(f: FileM)“ bzw. „export():FileM“, welche die FileM Klasse benutzt, um sich die Gewichtung der Edge Objekte zu importieren bzw. zu exportieren.

Part 4: Organisation und grafische Nutzeroberfläche

Part 4.1: Organisation

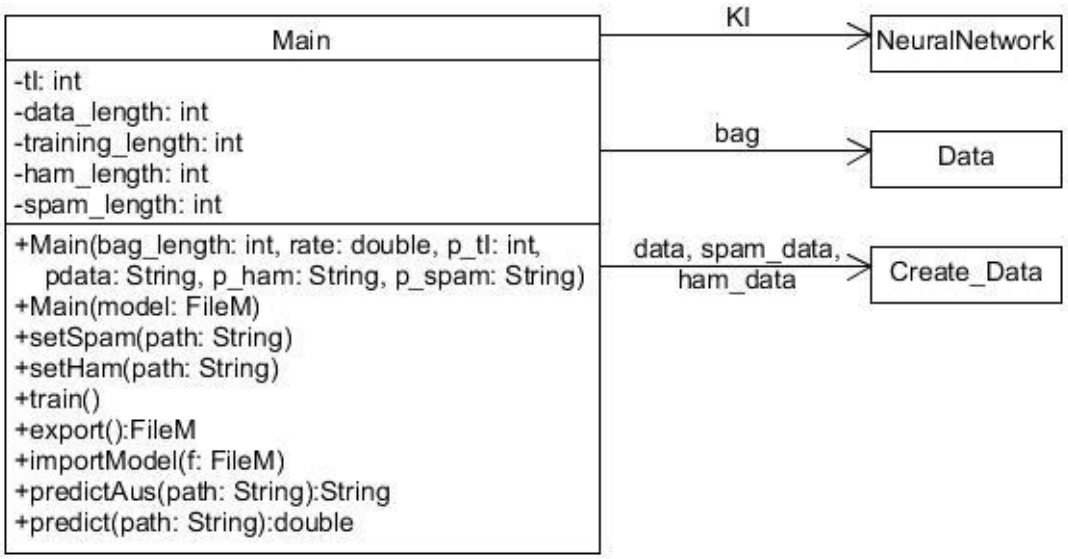


Abbildung 10

Die Main Klasse ist die Klasse die die beiden Konzepte, also das Bag-of-words Modell und das neuronale Netzwerk kombiniert und organisiert. Es speichert die Parameter Trainingsiterationen „tI: int“, Anzahl der Daten für das Bag-of-words Modell „data_length: int“, Anzahl der Trainings E-Mails „training_length: int“, Anzahl der Ham E-Mails (fürs Trainieren) „ham_length: int“ und die Anzahl der Spam E-Mails (fürs Trainieren) „spam_length“. Darüber hinaus speichert sich die Main Klasse ein neuronales Netzwerk als „KI“ und ein Bag-of-Words Modell als „bag“. Die nötigen Daten werden als „data“, „spam_data“ und „ham_data“ gespeichert, wobei das Create_Data Objekt data, die Emails für das Bag-of-words Modell speichert und spam_data bzw. ham_data die jeweiligen Emails fürs Trainieren.

Bei der Initialisierung der Main Klasse müssen die Pfade zu den Ordnern für die Emails (je für data, spam_data und ham_data), die Länge des bag-of-words Modells, die Lernrate für das neuronale Netzwerk und die Anzahl der Trainingsiterationen übergeben werden. Im Konstruktor werden die Parameter übernommen, die Emails als Create_Data Objekte gespeichert, das neuronale Netzwerk initialisiert und das Bag-of-words Modell gebaut. Die Main Klasse besitzt darüber hinaus einen Kopierkonstruktor, der mittels eines Objektes der FileM Klasse sich selbst bauen kann, dazu dient die Import Methode „importModel(f: FileM)“, welche mittels der Import Methode in der NeuralNetwork Klasse und den Kopierkonstruktor der Data Klasse alle Modelle nachbaut.

Die Main Klasse bietet zwei Setter Methoden an die es ermöglichen neue Trainingsemails auszuwählen: „setSpam(path: String)“ und „setHam(path: String)“. Außerdem besitzt sie eine Export Methode, die mittels der Export Methoden in der Data und NeuralNetwork Klasse ein Objekt der FileM Klasse zurückgibt, welche später wieder importiert werden kann: „export():FileM“. Darüber hinaus besitzt die Main Klasse eine Trainings Methode welche das neuronale Netzwerk auf Basis der Ham und Spam E-Mails trainiert: „train()“. Dabei ist zu beachten, dass das neuronale Netzwerk darauf trainiert ist Ham E-Mails mit einem Output von 0 und Spam E-Mails mit einem Output von 1 zu klassifizieren. Also je höher die Vorhersage ist desto eher ist die übergebene E-Mail eine Spam-E-Mail. Die Vorhersagen können mittels der Predict Methode „predict(path: String):double“ abgerufen werden, dazu muss der Pfad zur E-Mail als Parameter übergeben werden. Der ausgegebene Wert ist dabei zwischen 0 und 1, da das neuronale Netzwerk die Sigmoidfunktion zum Verarbeiten der Daten benutzt. Die Methode „predictAus(path: String):String“ gibt das gleiche nur als String aus welcher einen Text und den Output als Prozentzahl zurückgibt.

Bei der Predict Methode benutzt die Main Klasse die Convert_Data Klasse, welche einen übergebenen Pfad für eine Datei annimmt und dessen Text speichert und normalisiert (ähnlich wie die Create_Data Klasse). Der Text und Dateiname können dann über Getter Methoden von anderen Klassen abgerufen werden. (vgl. Part 4.2 UML Diagramm)

Part 4.2: Export und Import

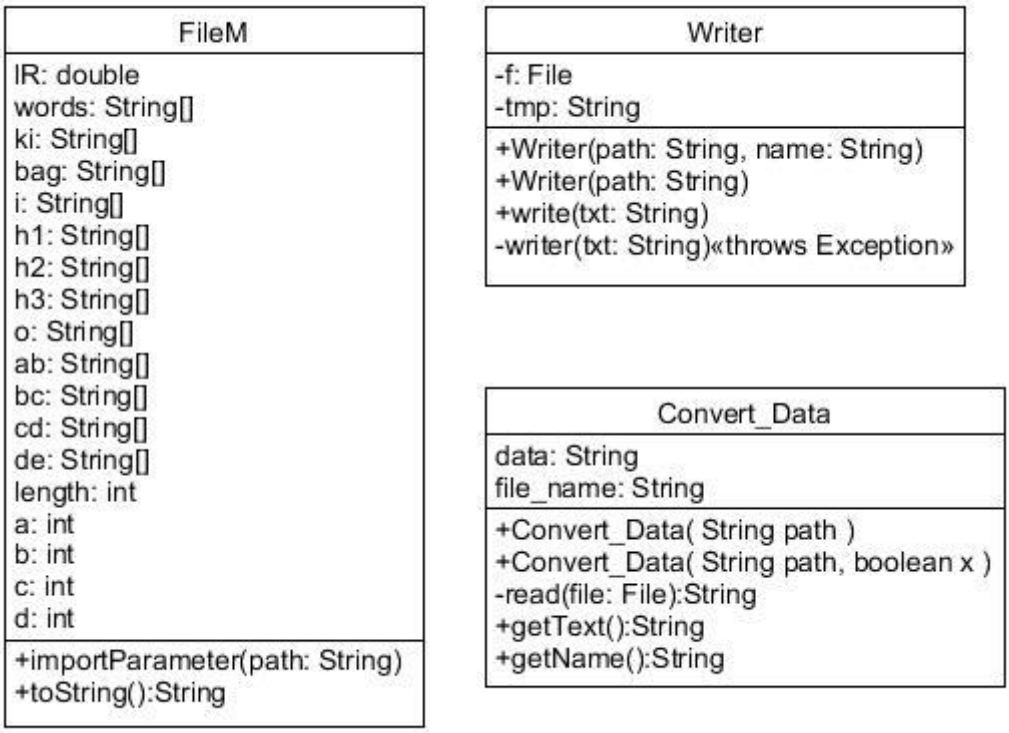


Abbildung 11

Der Export und Import der Modelle und das Schreiben des Projektes in eine Datei werden von der FileM und der Writer Klasse realisiert. Die FileM Klasse speichert alle wichtigen Parameter aus dem neuronalen Netzwerk und dem Bag-of-words Modell. Alle Variablen sind auf public gestellt damit die NeuralNetwork und die Data Klasse ihre Parameter reinschreiben können. Mittels der toString Methode kann dann ein Objekt der FileM Klasse in einem String zusammengefasst werden. Durch die FileM Klasse können die Modelle von Objekten zu Strings umgewandelt werden und so in Dateien gespeichert werden. Diese Dateien werden von der Writer Klasse geschrieben. Diese Klasse ermöglichst es Strings in Dateien zu schreiben. Diese Dateien können später mittels der Convert_Data Klasse gelesen werden und durch die FileM Klasse und den diversen Import Methoden der anderen Klassen wieder in Objekten umgewandelt werden.

Part 4.3: Grafische Nutzeroberfläche

Die GUI dieses Projektes wurde mittels Java Swing programmiert und organisiert die Main Klasse indem sie diese mit den Pfaden und Aktionen versorgt, die der User des Programms vorgibt.

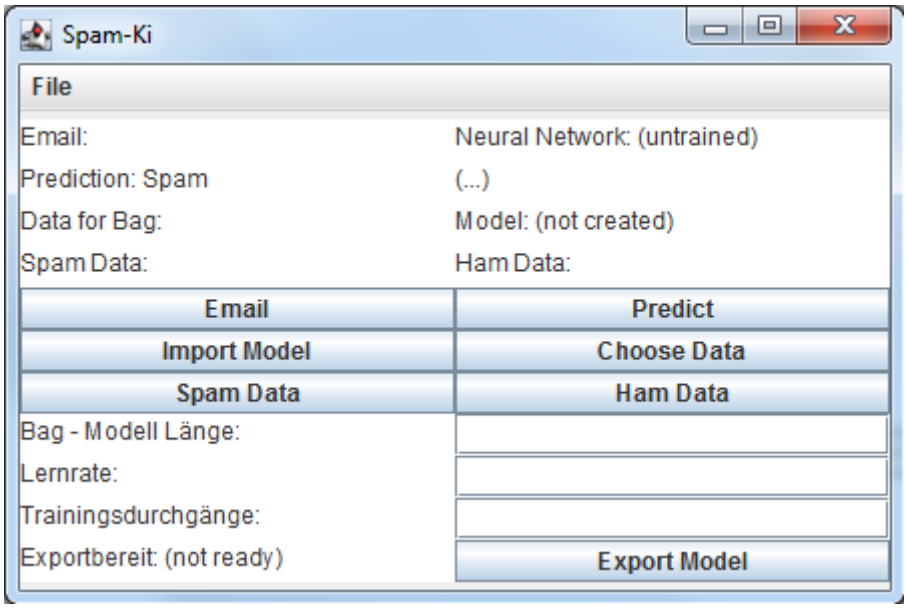


Abbildung 12

Um ein Modell zu importieren muss man „Import Model“ drücken und eine angebrachte Datei auswählen.

Um ein neues Modell anzubauen, muss man je bei „Choose Data“, „Spam Data“ und „Ham Data“ einen Ordner voller angebrachter Emails anwählen. Angebracht wären die Enron Datasätze oder ähnlich formatierte Emails. Danach muss man die Felder neben „Bag – Modell Länge:“, „Lernrate:“ und „Trainingsdurchgänge:“ ausfüllen. In die Textfelder neben „Bag – Modell Länge:“ und „Trainingsdurchgänge:“ müssen Integer stehen. In dem Textfeld neben „Lernrate:“ muss ein Double stehen. Danach muss man bei der Menuleiste über File „New Model“ auswählen, danach kann das Programm etwas laden und „freezen“, das bedeutet das es am Arbeiten ist. Steht nun in der dritten Zeile bei Model in der Klammer „created“, wurde das Model erfolgreich erstellt. Nun muss man wieder über die Menuleiste gehen und Train auswählen, auch hier kann es etwas dauern. Steht nun in der ersten Zeile bei Neural Network in der Klammer „trained“ ist das Modell fertig.

Wenn man ein E-Mail klassifizieren will, wählt man über den „Email“ Button eine Email aus und wählt dann den „Predict“ Button. Nun sollte in der zweiten Zeile in der Klammer eine Prozentzahl stehen.

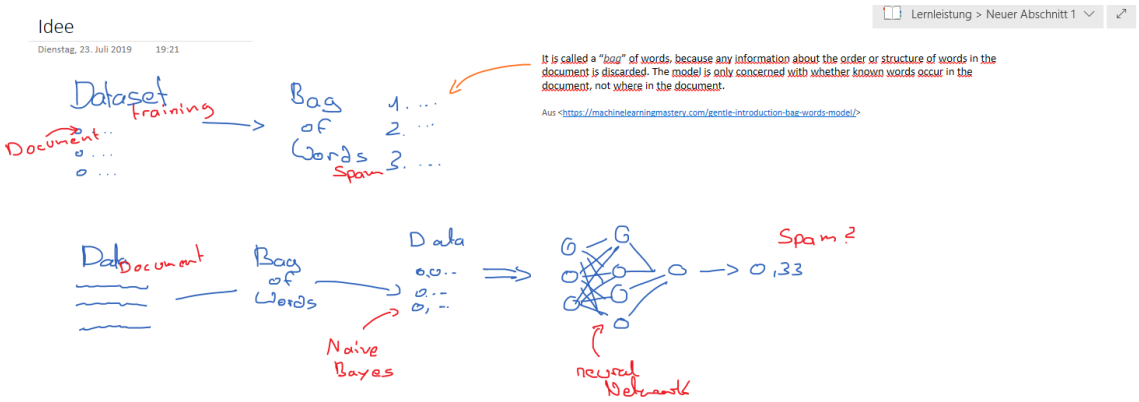
Möchte man sein eigenes Modell exportieren muss man den „Export Model“ Button auswählen und zusätzlich zu der Auswahl des Speicherorts den Dateinamen und Dateiendung hinschreiben.

Part 5: Fazit

Um das Programm zu starten muss man in BlueJ bei der Klasse GUI die static Main Methode „void main(String[] args)“ ausführen.

Die Ergebnisse des Programmes hängen von der Anzahl der benutzten E-Mails, der Größe des Bag-of-Word Modells und der Stärke des neuronalen Netzwerkes ab. Dabei hängt die Stärke des neuronalen Netzwerkes, neben der Größe des Bag-of-Word Modells auch von der Lernrate und der Anzahl der Trainingsiterationen ab. Ein angemessenes Modell besitzt eine Bag-of-Words Länge von 50, eine Lernrate von 0.35 und 10 000 Trainingsiterationen. Zu beachten ist es eine angemessene Anzahl von E-Mails (z.B.je 4 500 E-Mails) für die verschiedene Daten („Data for Bag“, „Spam Data“ und „Ham Data“) zu benutzen.

Ein starkes Modell verlangt eine Bag-of-Words Länge von mind. 500, eine Lernrate von 0.02, je 4 500 E-Mails für die Daten und mind. 100 000 Trainingsiterationen. Die Dauer kann für das Erstellen eines solchen Modells in die Stunden gehen. Bessere Modelle würden mehr E-Mails und Trainingsiterationen, aber dafür auch eine längere Dauer für das Erstellen und Trainieren dieser benötigen.



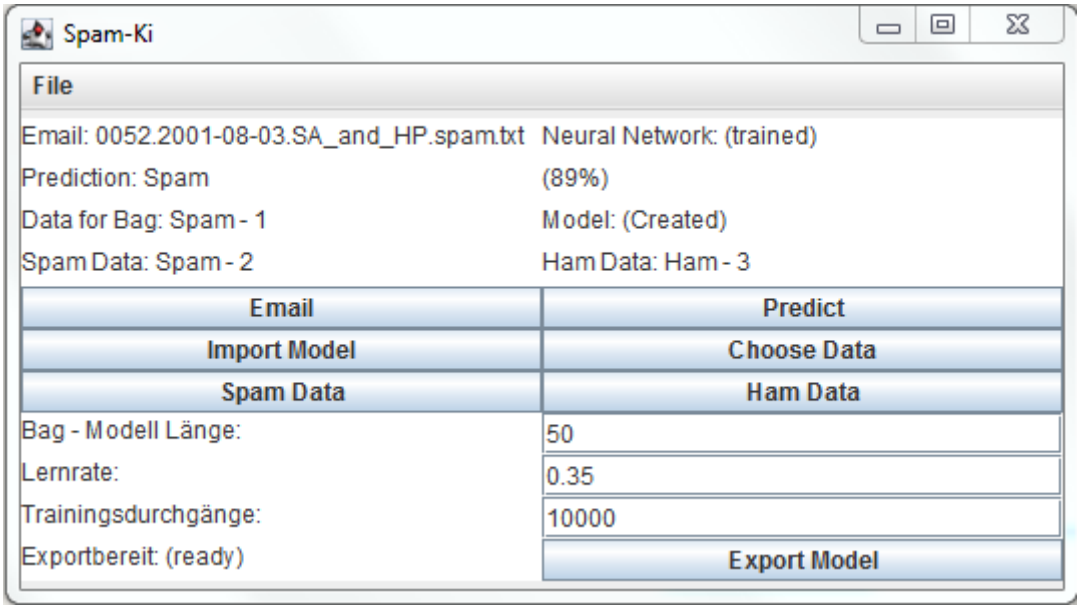


Abbildung 13 Von der Idee zum Programm

Literaturverzeichnis

Enron Datensatz: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/

Suchbegriff: „enron data nlp“ Datum: 30.03.2020

Alle Daten in dem Ordner Daten kommen aus dem Enron Dataset

Stoppwörter: http://www.lextek.com/manuals/onix/stopwords1.html

Suchbegriff: „stopwrods english txt“ Datum: 30.03.2020

Die Stoppwörter wurden nur als Basis benutzt Abweichungen können existieren.

Die modifizierte Version ist im Anhang

Abbildungsverzeichnis

Alle Abbilder wurden von mir entweder mittels Umlet oder mit der Word „Zeichnen“ Funktion angefertigt

Selbstständigkeitserklärung

Versicherung der selbständigen Erarbeitung

Ich versichere, dass ich die vorliegende Arbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

Swisttal (Ort), den 30.03.2020 (Datum)

(Unterschrift)