Duale Hochschule Baden-Württemberg Mannheim

Projektarbeit

Implementierung einer Sentimentanalyse von Nachrichten zur binären Vorhersage von Aktienverläufen

Studiengang Wirtschaftsinformatik

Studienrichtung Data Science

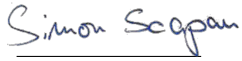
Verfasser und Matrikelnummer:	Jan Niklas Brebeck - 8016697
	Simon Scapan - 6699329
Kurs:	WWI18 - DSB
Studiengangsleiter:	Prof. Dr. Bernhard Drabant
Wissenschaftliche(r) Betreuer(in):	Prof. Dr. Tobias Günther
Bearbeitungszeitraum:	20.01.2012 – 01.02.2021

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Implementierung einer Sentimentanalyse von Nachrichten zur binären Vorhersage von Aktienverläufen* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, den 01.02.2021

Jan Niklas Brebeck



Simon Scapan

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

DJIA	Dow Jones Industrial Average
NLP	Natural Language Processing
DHBW	Duale Hochschule Baden-Württemberg
TF	Term Frequency
IDF	Inverse Document Frequency
WDF	Weighted Document Frequency

1 Einleitung

1.1 Darstellung des Anwendungsfalls

Die Zielsetzung dieses Projekts ist eine Vorhersage von Aktienkursen anhand einer Sentimentanalyse von Nachrichten. Dafür werden zwei Modelle, ein Naiver Bayes Klassifizierer und ein gewichteter Term Frequency Ansatz als Lösungsmöglichkeit implementiert. Die beiden Modelle lernen Anhand des Datenkorpus von Sun [8]. Die ausführliche Datenvorverarbeitung wird im Kapitel 2 beschrieben. Die Modelle lernen somit, welche Nachrichten zu einem Kursabfall beziehungsweise Kursanstieg geführt haben und bewerten einzelne Wörter der Nachrichten dementsprechend positiv, respektive negativ.

Ein Anwendungsfall wäre daraus folgend in zwei Granularitätsstufen gegeben. Zum einen kann das Resultat direkt genutzt werden, um im Hintergrund automatisch die aktuellsten Nachrichten zu bewerten und somit eine Handlungsempfehlung auszusprechen. Eine weitere mögliche Nutzung des Resultates ist die Integration in einen Robo Advisor. Dieser ist ein, Zitat: „Algorithmenbasiertes System, das Empfehlungen zur Vermögensanlage gibt und diese auch automatisiert umsetzen kann.“ [4] Diese arbeiten zum einen anhand historischer Aktienkurse, können aber zusätzlich durch eine solche Sentimentanalyse in der Prognose unterstützt werden.

1.2 Methodik des Projektes

Die Methodik der Bearbeitung stellt sich aus vier Bereichen zusammen. Am Anfang steht eine umfassende und tiefreichende Datenvorverarbeitung. Diese hat zum Ziel eine Datenbasis zu schaffen, welche bestmögliche Ergebnisse der Modelle ermöglicht. Die Daten werden dann in Trainings- und Testdaten aufgeteilt. Zwei Modelle werden anhand des zuvor bereitgestellten Trainingsdatensatzes gelernt. Das ist zum einen ein Naive Bayes Ansatz unter Kapitel 3.1 und eine Implementierung Anhand einer gewichteten Term Häufigkeit 3.2. Die beiden Modelle werden dann in der Schlussfolgerung in Kapitel 4 Anhand der Testdatensätze gegeneinander evaluiert. Dafür werden vorher Vergleichsparameter festgelegt.

2 Daten

2.1 Datenbeschreibung

Für die Umsetzung dieses Projekts, wird der Datensatz „Daily News for Stock Market Prediction, Version 1. Retrieved“ von Sun [8] aus der Kaggle-Plattform verwendet.

Dieser Datensatz enthält die täglichen Top 25 Beiträge des Reddit WorldNews Channels, basierend auf deren erhaltenen Bewertungen, im Zeitraum vom 08.Juni 2008 bis zum 01. Juli 2016. Dies ist in der Datei: „RedditNews.csv“ zu finden. Darüberhinaus sind die Aktiendaten des Dow Jones Industrial Average (DJIA) im Zeitraum vom 08. August 2008 bis zum 01. Juli 2016, basierend auf den Daten der Yahoo Finance Webseite, enthalten. Die Aktiendaten enthalten neben dem Datum, die Kursinformationen für Open, High, Low, Close, Volume, Adj. Close. Dabei steht Open für den Eröffnungskurs, High für den höchsten Kurs des Tages, Low für den niedrigsten Kurs des Tages, Volume für die Anzahl der gehandelten Aktien, Close für den Kurs zum Handelsschluss und Adj. Close für den Schlusskurs unter Berücksichtigungn von Stock Splits und Dividenden. Für die weitere Arbeit mit dem Datensatz, wurden lediglich die Werte des Adj. Close berücksichtigt. Die Daten hierzu befinden sich in der „DJIA_table.csv“ Datei. Zuletzt wird ein weiterer Datenbestand angeboten, welcher die beiden vorher genannten Datenquellen vereint. Dieser ist in der Datei „CombinedNewsDJIA.csv“ zu finden. Dort sind das Datum, das zugehörige Label und die entsprechenden täglichen Top 25 Nachrichten zum entsprechenden Datum zu finden. Das Label besagt dabei mit dem Wert „1“, dass der Kurs gestiegen oder gleich geblieben ist oder mit dem Label „0“, dass der Kurs gefallen ist.

2.2 Datenvorverarbeitung

In der Datenvorverarbeitung werden die vorliegenden Daten bearbeitet, um im Anschluss die bestmöglichen Ergebnisse auf den Daten zu erzielen. Um dies zu erreichen, werden zunächst alle Null-Werte mit leeren Strings aufgefüllt. Anschließend wird ein neues Dataframe mit den folgenden drei Spalten aufgebaut:

Spaltenname	**Datenbeschreibung**
Date	Datum
Label	Aktienkurs mit 1 und -1 gelabelt
News	Nachrichten als ein String

Tabelle 2.1: Dataframe Schema

Für das neue Dataframe werden alle Nachrichten eines Tages zu einem String zusammengefasst. Dieser wird anschließend in die Kleinschreibung transformiert und Satzzeichen, wie zum Beispiel Punkte, werden entfernt. Danach wird die Nachricht mit dem „word_tokenizer“ der „nltk.tokenize“ Bibliothek in einzelne Wörter gesplittet, sodass eine Liste aus allen Wörtern entsteht. Über diese Liste wird iteriert und mit Hilfe der „lemmatize“ Funktion aus der „word_forms.lemmatizer“ Bibliothek, die Wörter auf ihren Wortstamm gebracht. Diese Implementierung liefert für das „Stemming“ beziehungsweise die „Lemmatization“ die besten Ergebnisse aus den getesteten Bibliotheken: [nltk PorterStemmer und SnowballStemmer; nltk WordNetLemmatizer; spacy Lemma_ Funktion]. Mit jeder dieser Bibliothek wurde jeweils der gleiche, zusammengesetzte String bearbeitet und manuell betrachtet. Dabei stellte sich der „word_forms.lemmatizer“, bei der manuellen Betrachtung der Ergebnisse, als die beste Implementierung heraus.
Nach dem die Wortstämme gebildet wurden, werden alle Wörter die aus weniger als drei Buchstaben bestehen, in dem „nltk.corpus“ für englische Stoppwörter enthalten oder eine Zahl sind, herausgefiltert. Abschließend wird der zusammengesetzte String dem neuen Dataframe hinzugefügt.

3 Ansätze der Implementierung

3.1 Umsetzung eines Naive Bayes Klassifizierer

In diesem Kapitel wird der Ansatz der Implementierung eines Dokumentenklassifizierers mit Hilfe von Naive Bayes beschrieben. Der Naive Bayes Klassifikator ist abgeleitet vom Satz von Bayes.

$$P(A|B) = \frac{(P(B|A) * P(A))}{P(B)} \tag{3.1}$$

Unter Berücksichtigung der folgenden „naiven“ Annahmen, kann der Satz von Bayes zu dem folgendem „naiven“ Klassifizierer umgeformt werden:

- „Bag of Words“: Die Position eines Wortes in einem Satz hat keine Auswirkung
- Bedingte Unabhängigkeit: Es wird angenommen, dass die Merkmale keine Abhängigkeit zur Klasse haben

[6]

$$\hat{y} = \underset{i}{\operatorname{argmax}}(\prod_{k=1}^{p} P(x_k|y_i)P(y_i)) \tag{3.2}$$

[7, S.406]

Dieser Ansatz wird im folgenden verwendet um den Klassifizierer zu implementieren.

3.1.1 Implementierung des Naiven Bayes Klassifizierers

Für die Umsetzung des Naiven Bayes Klassifizierers werden zunächst die Wahrscheinlichkeiten für die Klassen, sowie die Häufigkeiten der Wörter innerhalb der Klassen ermittelt. Dazu wird die `train_naive_bayes()` Funktion verwendet. Diese iterriert über alle Zeilen und die darin enthaltenen Nachrichten des Trainingsdatensatzes und zählt für die entsprechende Kategorie, die Worthäufigkeit für alle vorkommenden Wörter. Nachdem dadurch zwei

Dictionaries für jede Klassen erstellt wurden, können diese zur Klassifizierung verwendet werden.

Dies wurde in der `calc_document_probabilities()` Funktion implementiert. Hierbei gibt es die Optionen einen Threshold und/oder Laplace Smoothing zu verwenden. Des Weiteren gibt es die Möglichkeit Wörter, die nicht im Trainingsdatensatz enthalten sind, nicht für die Klassifizierung zu berücksichtigen. Für das Laplace Smoothing werden zwei neue Parameter in der Berechnung der Wortwahrscheinlichkeit für eine Klasse eingeführt, sodass sich die Berechnung wie folgt ändert:

Von: $\frac{\text{Worthäufigkeit in der Klasse}}{\text{Wortanzahl der Klasse}}$

Zu: $\frac{\text{Worthäufigkeit in der Klasse}+\textbf{alpha}}{\text{Wortanzahl der Klasse}+\textbf{beta}}$

[3]

Neben dem Laplace Smoothing, gibt es auch die Option, Wörter, welche nicht im Trainingsdatensatz enthalten sind, nicht für die Klassifizierung zu verwenden. Darüberhinaus kann der bereits erwähnte Threshold angewendet werden, wenn Wörter, die nicht im Trainingsdatensatz enthalten sind, nicht für die Klassifizierung berücksichtigt werden. Beim Einsetzen des Threshold, muss die Wortwahrscheinlichkeit, die mit der Formel vor dem Laplace Smoothing berechnet wird, einen gegebenen Threshold überschreiten, damit die Wahrscheinlichkeit für die Klassifizierung berücksichtigt wird. Ist dies nicht der Fall, wird das Wort nicht für die Klassifizierung verwendet.

Zur Evaluation des besten Klassifizierers, wurden alle Optionskombination getestet. Bei den Durchläufen, ohne Berücksichtigung der nicht vorhandenen Wörter im Trainingsdatensatz, wurden Thresholds im Bereich von 0.0001 bis exklusive 0.1 mit einer Schrittweite von 0.0001 betrachtet. Die Ergebnisse der Evaluation sind wie folgt:

Variante	Threshold	Accuracy
Mit nicht vorhandenen Wörtern / Ohne Smoothing	0	55,16%
Mit nicht vorhandenen Wörtern / Mit Smoothing	0	55,16%
Ohne nicht vorhandene Wörter / Mit Smoothing	0.0003	56,93%
Ohne nicht vorhandene Wörter / Ohne Smoothing	0.0018	57,43%

Tabelle 3.1: Evalutation Naive Bayes Optionen

Für den späteren Vergleich der beiden Ansätze in Kapitel „Gegenüberstellung der Ergebnisse“ wurde die letzte Variante verwendet.

3.2 Umsetzung Anhand gewichteter Dokument Häufigkeit

In diesem Kapitel wird der Ansatz unter Benutzung einer relativen gewichteten Häufigkeit näher betrachtet. Dabei wird eingangs auf die grundsätzliche Überlegung eingegangen. Weiter werden dann die einzelnen Schritte von der Term Häufigkeit bis zum Resultat, der gewichteten Häufigkeit anhand der Implementierung erläutert.

3.2.1 Grundsätzliche Überlegung

Die Idee der Implementierung anhand relativer Häufigkeiten entstammt einer Lehrveranstaltung zum Thema Natural Language Processing (NLP) an der Duale Hochschule Baden-Württemberg (DHBW) Mannheim. In dieser Veranstaltung wurde zum Thema Document Retrieval Vorgelesen. Dem Prinzip von Occam's razor besagt, dass immer die simplere, zweier Theorien einzusetzen ist. [4] Diesem Prinzip folgend, wurde die Überlegung angestrebt das Konzept des übergeordneten Bereichs Information Retrieval zu Nutzen und auf den gegebenen Anwendungsfall anzupassen.

Im Grundsatz wird dabei für jeden Satz in einem Dokument die Worthäufigkeit berechnet. Diese setzt sich zusammen aus: [2]

$$Worthäufigkeit_{\,x} = \frac{Anzahl\,Wörter\,x}{Anzahl\,aller\,Wörter} \tag{3.3}$$

Des Weiteren wird die übergeordnete inverse Dokumenthäufigkeit benötigt, welche eine Aussage über die relative Häufigkeit der Anzahl an Dokumenten mit einem gegebenen Wort beinhalten, trifft. Die inverse Dokumenthäufigkeit wird berechnet durch:[2]

$$inverse\,Dokumenthäufigkeit_{\,x} = log(\frac{Anzahl\,aller\,Dokumente}{Anzahl\,Dokumente\,mit\,x}) \tag{3.4}$$

Um nun zwei Sätze miteinander zu vergleichen, wird deren Worthäufigkeit auf die Dokumenthäufigkeit projiziert. Die hierbei zu wählende Rechenoperation ist die Multiplikation. Das hat den Vorteil, dass jegliche miteinander zu vergleichende Sätze, welche unterschiedliche Dimensionen der Worthäufigkeit aufweisen, durch die Vereinheitlichung beziehungsweise Aggregation mit der Dokumenthäufigkeit der selben Dimension sind. Im Falle des

Information Retrieval könnten so zwei Sätz eines Dokumentes anhand zum Beispiel der Euklidischen Distanz auf ihre Ähnlichkeit zueinander Untersucht werden.

Der letzte Schritt, der Vergleich, ist dabei für den Vorliegenden Anwendungsfall nicht Notwendig. Die eben gelegten Grundlagen des Information Retrieval werden in den nachfolgenden Unterkapiteln modifiziert und an die vorliegende Fragestellung angepasst.

3.2.2 Berechnung der Worthäufigkeit

Aus der grundsätzlichen Überlegung im Kapitel 3.2.1 wurde bereits die Formel 3.3 für die Berechnung der Häufigkeit eines Wortes (englisch: Term Frequency (TF)) in einem Satz aufgestellt. In diesem Kapitel wird sich der konkreten Implementierung gewidmet. Dazu wurde folgender Quelltext geschrieben:

```
def get_tf(message, label):
    tf_dict = {}

    for word in message:
        if word not in tf_dict:
            tf_dict[word] = (message.count(word)/len(message)) * int(
                                                    label)

    return tf_dict
```

Die übergebenen Parameter (`message, label`) sind zum Einen die zu betrachtende Nachricht als Liste von Wörtern aus der Datenvorverarbeitung, zum Anderen wird ein Label übergeben. Dieses ist Numerisch und gibt an, ob eine Nachricht positiv oder negativ konnotiert ist. Für die spätere Erstellung des Weighted Document Frequency (WDF) ist dies von Relevanz. Zur Berechnung der TF für den Testdatensatz wird der Wert 1 übergeben, welcher keine Veränderung des TF zur Folge hat.

Im inneren der Funktion wird zuerst ein Dictionary für das spätere befüllen mit den einzelnen Wörtern und deren Häufigkeiten als `tf_dict` initiiert. Folgt wird über jedes Wort in der Liste der Wörter iteriert. Dabei wird geprüft, ob das Wort bereits im Dictionary vorhanden ist. Falls das nicht der Fall ist, wird die Häufigkeit mit nachfolgender Formel berechnet:

$$Gewichtete\ Häufigkeit_x = \frac{Anzahl\ Wörter\ x}{Anzahl\ Wörter\ in\ Liste} * Gewicht \tag{3.5}$$

Der Stamm der Funktion ist Analog 3.3, mit dem Zusatz, dass hier ein Gewicht hinzugefügt wird. Das Gewicht hat im Fall des vorliegenden Projektes die Ausprägungen 1, respektive -1. Das bedeutet, dass somit nur das Vorzeichen gewechselt wird.

Eine `else` Schleife wird in diesem Fall nicht benötigt, denn Falls ein Wort nochmals in der Liste vorkommt wurde es bereits beim ersten Aufkommen mit Betrachtet, das im Code in Zeile 6 die Anzahl der gleichen Wörter gezählt wird. Mit `len(message)` wird die Anzahl der Einträge in der übergebenen Liste determiniert.

Die Rückgabe der Funktion ist das befüllte Dictionary. Das Wort wird als Schlüssel und die Häufigkeit als Wert festgehalten. Das Dictionary beinhaltet alle Wörter der Liste, welche der Funktion übergeben wurden.

3.2.3 Berechnung der inversen Dokumenthäufigkeit

Die Berechnung der inversen Dokumenthäufigkeit wird in zwei Schritten implementiert. Dafür wird der Funktion `get_idf` ein Dataframe mit den einzelnen Wort-Listen der Nachrichten übergeben. Dies ist dem folgenden Quelltextausschnitt zu entnehmen:

```
def get_idf(data):

    idf_dict = {}

    for index, row in data.iterrows():
        for word in row[3]:
            if word not in idf_dict:
                idf_dict[word] = 0

    doc_count = data.shape[0]

    for word in idf_dict:
        count = 0
        for index, row in data.iterrows():
            if word in row[3]:
                count += 1
            else:
                count = 1
        idf_dict[word] = math.log(doc_count / count)

    return idf_dict
```

Im ersten Schritt wird das übergebene Dataframe auf alle einzigartigen Wörter untersucht. Diese werden in das davor initiierte Dictionary `idf_dict` als Schlüssel geschrieben. Der Wert wird erst in einer späteren Berechnung determiniert und deshalb auf 0 gesetzt.

Ist der Korpus der Dokumenthäufigkeit erstellt, kann mit der Berechnung für jeden einzelnen Eintrag weiter verfahren werden. Dazu wird im Quelltext in Zeile 12 über jedes Wort in ebendiesem Dictionary iteriert. Für jedes Wort wird nun iterativ geprüft, in wie vielen Nachrichten des übergebenen Datensatzes es mindestens ein Mal vorkommt. Einmal über alle Einträge des Datensatzes iteriert, wird die inverse Dokument Häufigkeit Analog der Formel 3.4 und in Quelltextzeile 20 berechnet. Diese berechnete Häufigkeit wird direkt als Wert des betrachteten Wortes in das Dictionary geschrieben.

Nach der Iteration über alle Wörter im Dataframe wird dieses aus der Funktion zurückgegeben.

3.2.4 Berechnung der gewichteten Dokument Häufigkeit

Der letzte Schritt in der Berechnung des Information Retrieval Ansatzes ist die Kombination der einzelnen Wort Häufigkeiten mit der übergeordneten Dokument Häufigkeit. Da gleiche Wörter mit unterschiedlicher Häufigkeit vorkommen und des Weiteren auch alternierende Gewichte haben können, werden diese im folgenden Schritt Kombinieren. Der dazugehörige Quelltext sieht wie folgt aus:

```
def calc_wdf(data):
    idf_dict = get_idf(data)
    tf_comb = {}

    for index, row in data.iterrows():
        tf_dict = get_tf(row[3],row[2])
        for word in tf_dict:
            if word not in tf_comb:
                tf_comb[word] = [ tf_dict.get(word), 1 ]
            else:
                tf_comb[word] = [ tf_comb.get(word)[0] + tf_dict.get(
                                                        word), tf_comb.get
                                                        (word)[1] + 1 ]

    for word in idf_dict:
        idf_dict[word] = idf_dict.get(word) * ( tf_comb.get(word)[0]
                                              / tf_comb.get(word)[1] )
```

```
    return idf_dict
```

In Zeile 2 wird zuerst die im Kapitel 3.2.3 erläuterte Dokumenthäufigkeit für den übergebenen Datensatz berechnet. Danach teilt sich die Berechnung in zwei Bestandteile, zum einen die Kombination der einzelnen Worthäufigkeiten aus dem Datensatz, zum anderen die Multiplikation eben dieser Worthäufigkeit mit der Häufigkeit des Vorkommens im Datensatz, der Dokumenthäufigkeit.

Im ersten Schritt wird für jede Datenreihe die Worthäufigkeit, aus Kapitel 3.2.2, berechnet und für jedes Wort, zu sehen in Quelltextzeile 9-12, in ein neues Dictionary abgelegt. Die Besonderheit liegt darin, dass alle einzelnen Häufigkeiten eines Wortes aufaddiert werden und die Anzahl der einzelnen Einträge mitgezählt wird. Hierfür wird als Wert des Dictionaries eine Liste übergeben. In Zeile 14 und 15 werden abschließend für jedes Wort die Dokument Häufigkeit mit der Term Häufigkeit iterativ multipliziert. Dabei ist anzumerken, dass in diesem Schritt, die Wort Häufigkeit über die den zuvor eingebauten Zähler relativiert wird. Damit stehen alle Wort Häufigkeiten in gleicher, gewichteter, Relation zueinander.

Die Rückgabe der Funktion beinhaltet somit die in Relation gesetzte gewichtete Wort Häufigkeit in Abhängigkeit aller Dokumente des übergebenen Datensatzes.

4 Schlussfolgerung

4.1 Vergleich der Ergebnisse beider Modelle

Im ersten Unterabschnitt diesen Kapitels werden notwendige Vergleichsparameter aufgestellt, um die beiden implementierten Modelle mit den selben Metriken vergleichen zu können. Die einzelnen Parameter werden erläutert und deren Aussage dargestellt. Nachdem die Grundlagen gelegt wurden, werden die beiden Modelle anhand der Ergebnisse basierend auf dem Testdatensatz miteinander verglichen. Ziel des Vergleiches ist, das bessere Modell festzustellen.

4.1.1 Aufstellen der Vergleichsparameter

Das behandelte Problem wird in die Gruppe der Klassifizierungsprobleme eingeordnet. Eine Metrik für diese Art der Probleme ist die Confusion Matrix. Im Folgenden wird der Aufbau dieser Matrix erläutert und auf einige besondere und wichtige Kennzahlen eingegangen. Der allgemeine Aufbau einer Confusion Matrix nach Abhigyan [1] ist:

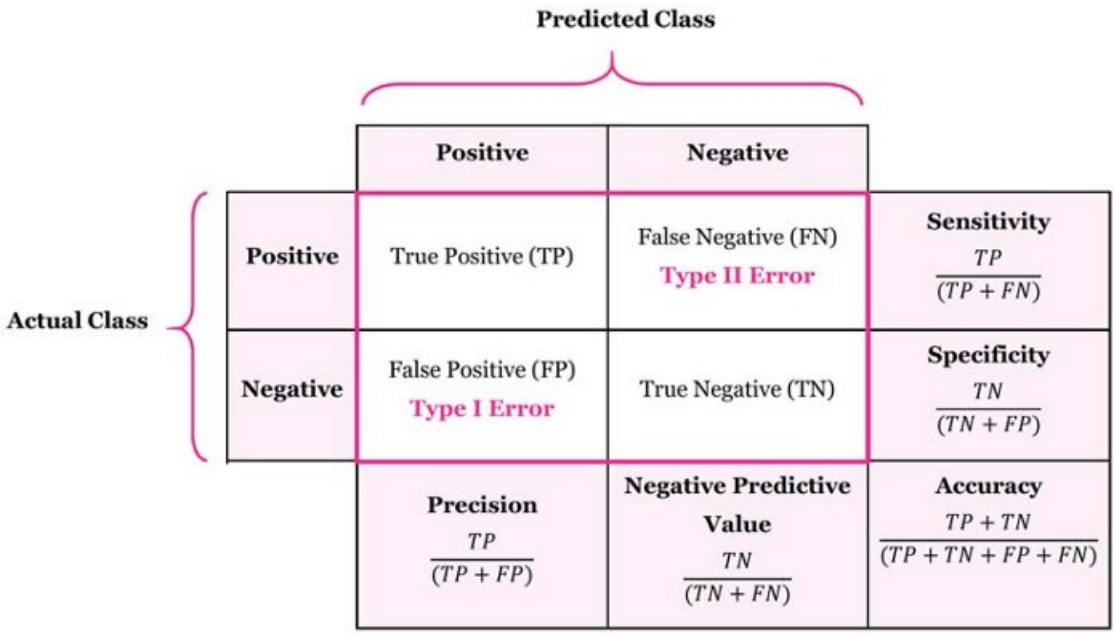


Abbildung 4.1: Confusion Matrix nach Abhigyan [1]

Bei der Betrachtung der Confusion Matrix sind zwei Gruppen von Kennzahlen zu erkennen. Zum einen die direkten Ergebnisse der Klassifizierung, wie zum Beispiel die True Positve Rate, welche sich in den weiß hinterlegten Zellen befinden. Diese entstammen direkt der Klassifizierung. Die True Positive Kennzahl setzt sich zusammensetzen aus der Anzahl Positiv Klassifizierter Datenfelder. Wie beispielhaft anhand der True Positive Kennzahl, setzten sich die weiteren drei Analog zusammen. Auf der anderen Seite, sind rechts neben und unter der Confusion Matrix, in der Abbildung 4.1, erweiterte Kennzahlen zu sehen. Diese setzen sich ausschließlich aus den direkten Kennzahlen zusammen. Es werden die Aussagen folgender vier Kennzahlen nach Narkhede [5] jeweils kurz erläutert:

Precision Anteil der korrekt positiven Vorhersagen, aller positiven Vorhersagen.

Sensitivity (Recall) Anteil der korrekt positiven Vorhersagen, aller tatsächlich positiven Datenpunkte.

Specificity Anteil der korrekt negativen Vorhersagen, aller tatsächlich negativen Datenpunkte.

Accuracy Anteil der insgesamt richtigen Vorhersagen.

Dabei sollten alle Kennzahlen höchstmögliche Werte vorweisen. [5]

4.1.2 Gegenüberstellung der Ergebnisse

In diesem Kapitel werden die Ergebnisse der beiden Modelle tabellarisch gegeneinander gestellt. Das Modell mit den höchsten Werten der erweiterten Kennzahlen ist als das bessere Modell anzusehen.

Aus der nachfolgenden Tabelle sind die Kennzahlenwerte zu entnehmen, welche beide Modelle anhand der Testdaten erzielt haben:

Modell	**Naive Bayes**	**Information Retrieval**
Anzahl Datenpunkte	397	397
True Positive Rate	176	211
True Negative Rate	52	6
False Positive Rate	126	172
False Negative Rate	43	8
Precision	0.583	0.551
Recall	0.804	0.963
Specificity	0.292	0.034
Accuracy	0.574	0.547

Tabelle 4.1: Evaluation der Ergebnisse beider Modelle anhand des Testdatensatzes

4.2 Fazit

Aus der Tabelle 4.1 ist klar zu sehen, das der Naive Bayes Klassifikator deutlich bessere Ergebnisse erzielt, als der Information Retrieval Ansatz. Bei Zweiterem kann Anhand der Daten die Aussage getroffen werden, dass dieser eher dazu tendiert Nachrichten als Positiv zu bewerten.

Beim genaueren Hinblick auf die Accuracy ist zu erkennen, dass beide Modelle mit ca. 55% beziehungsweise ca. 57% nur knapp als Signifikant einzustufen sind. Das kommt aber nicht dadurch, dass die Modelle eine Allgemein geringe Güte aufweisen, sondern, weil der Datensatz keine besseren Informationen beinhaltet. Sun, welcher die Daten auf der Plattform Kaggle zur Verfügung gestellt hat, geht von einer maximalen Accuracy von 62-63% aus.[8] Verglichen damit kann die Aussage getroffen werden, dass die vorliegenden Modelle sehr gute Aussagen treffen. Das bessere der beiden Modelle ist aber deutlich der Naive Bayes Ansatz, da dieser bei den erweiterten Kennzahlen besser abschneidet.

Die Implementierung des Projektes ist unter folgendem Link zu erreichen:
`https://github.com/SimonScapan/NLP-Project`

Literaturverzeichnis

[1] Abhigyan. *Calculating Accuracy of an ML Model*. Juli 2020. URL: https://medium.com/analytics-vidhya/calculating-accuracy-of-an-ml-model-8ae7894802e.

[2] Tobias Günther. *Document Retrieval*. URL: https://moodle.dhbw-mannheim.de/pluginfile.php/341092/mod_resource/content/1/tf_idf.jpg.

[3] Olli Huang. *Applying Multinomial Naive Bayes to NLP Problems: A Practical Explanation*. Juli 2017. URL: https://medium.com/syncedreview/applying-multinomial-naive-bayes-to-nlp-problems-a-practical-explanation-4f5271768ebf.

[4] Prof. Dr. Andreas Mitschele. *Definition: Robo-Advisor*. Mai 2020. URL: https://wirtschaftslexikon.gabler.de/definition/robo-advisor-54214/version-379056.

[5] Sarang Narkhede. *Understanding Confusion Matrix*. Jan. 2021. URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[6] Abhinav Rai. *Text Classification in NLP — Naive Bayes*. Jan. 2017. URL: https://theflyingmantis.medium.com/text-classification-in-nlp-naive-bayes-a606bf419f8c.

[7] S. Ranganathan, K. Nakai und C. Schonbach. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*. Elsevier Science, 2018. ISBN: 9780128114322.

[8] J. Sun. *Daily News for Stock Market Prediction, Version 1. Retrieved*. 1. Aug. 2016. URL: https://www.kaggle.com/aaron7sun/stocknews (besucht am 21. 01. 2021).