# Contents

**Abstract**

# 1 Einführung

Die folgende Arbeit befasst sich mit der Frage inwiefern sich Feature Engineering bei der Anwendung [8] verschiedener Klassifizierungsmethoden auf die Ergebnisse auswirkt. Der ganze Versuch wird in Zusammenarbeit mit der Firma Osram Opto Semiconductor unternommen.

## 1.1 Motivation

Digitalisierung ist heutzutage eines der meistverwendeten Buzzwords der modernen Industrie. Arbeitsschritte sollen automatisiert oder generell "smarter" gestaltet werden. Um den Wandel kontrolliert und sinnvoll umzusetzen, wurde die Abteilung Digital Transformation ins Leben gerufen. Die Hauptaufgabe der Abteilung besteht im Analysieren von Business Prozessen, die im Optimalfall durch digitale Lösungsansätze effizienter gestaltet werden können. So auch in der Sales-Abteilung bei Osram OS.
Ein großer Bestandteil der Arbeit eines Mitarbeiters im Sales-Department ist die Kundenbetreuung, welche fast ausschließlich über E-Mailverkehr vollzogen wird. Eine Option um den Workflow zu erleichtern oder nachvollziehbarer zu gestalten, ist das Einrichten eines *Customer Relations Management* Systems. Zur Zeit wird ein solches System aufgesetzt. Dieses System kann nicht alle Arbeitsschritte erleichtern, was die Überlegung anregt einer maschinellen Unterstützung durch künstliche Intelligenz ins Auge zu fassen. Im konkreten Fall handelt es sich um das Bearbeiten und Einordnen von Anfragen der Kunden. Welcher Kunde oder interessierter Kunde will eine Auskunft zu welchem Thema? Pro Tag trifft eine große Menge an verschiedenen Kundenanfragen in das zuständige E-Mailpostfach ein. Der Inhalt kann zwischen Quote, Pricing, technische Anfragen oder Beratung, bis zu fachfremdem variieren.
Um dem mühsamen und oft zeitaufwendigen Prozess zu vereinfachen, soll getestet werden wie sehr die Verarbeitung maschinell durchgeführt werden kann. Um vom Eintreffen einer E-Mail bis zum automatisierten Antworten zu kommen, sind einige *Toolbausteine* zu implementieren. Diese Arbeit beschränkt sich auf den Prozess des Sortierens, der unterscheidet ob eine E-Mail zu einer bestimmten Kategorie gehört oder nicht. Das Auslesen der Eckdaten und ihre Weiterverarbeitung im SAP oder anderen Tools soll ebenso Teil dieser großen *Toolchain* werden. Das ideale Endergebnis ist eine automatisch generierte Antwortmail oder ein ausgefülltes Formular, welches alle erfragten Parameter enthält.
Der Baustein der Klassifizierung ist nur einer der Schritte zur vollautomatisierten Beantwortung oder Bearbeitung der Anfragen.

## 1.2 Anforderungen

Um sinnvolle und gute Ergebnisse beim Forschen mit Machine Learning Systemen zu erhalten, ist eine große Datenmenge unverzichtbar. Der Forschungsinhalt der Arbeit legt seinen Schwerpunkt auf das *Preprocessing* der E-Mails und das Verwenden verschiedener Klassifizierungsmethoden. Preprocessing bedeutet dass Datensätze vorab angepasst oder verändert werden, damit leichter oder effektiver mit ihnen gearbeitet werden kann. Dafür muss der vorhandene Datensatz, bestehend aus E-Mails, zum Teil per Hand durchsucht werden um festzustellen welche Eigenschaften sinnvoll dazu beitragen, dass ein Classifier gute Ergebnisse liefert. Solche Eigenschaften werden Features genannt 3.1.
Nachdem sinnvolle *Features* ausgewählt worden sind, werden verschiedene Klassifzierungsarten ausgewählt. Im Vornherein muss geklärt werden, welche Classifier zu sinnvollen und vergleichbaren Ergebnissen führen.

## 1.3 Forschungsfragen

Das Ziel der Forschung liegt in der Frage welche angewandten Kombinationen aus Features mit welchen Classifiern die besten Ergebnisse liefern. Dabei werden nicht nur die Features verglichen sondern auch die Classifier unter sich. Welcher Classifier ordnet mit der geringsten Fehlerrate zufällig ausgewählte Testmails ein?
Desweiteren soll evaluiert werden ob die Anwendung von *Machine Learning* in diesem Szenario sinnvoll ist oder nicht. Der Hintergrund dieser Fragestellung beruht auf der Möglichkeit auf gewisse Schlüsselworte oder Produktnummern in den E-Mails zu achten und diese ohne *Machine Learning* zu sortieren.
Abschließend soll besonderes Augenmerk auf die Performance von Neuronalen Netzen gelegt werden, da Osram OS ein neues *Customer Relations Management* System von Salesforce aufsetzt. Dieses Tool bietet die Möglichkeit eine KI namens *Einstein* basierend auf Neuronalen Netzen zu verwenden. Somit kann der Einsatz eines Neuronalen Netz in diesem Usecase als Feldtest gesehen werden.

# 2 Theoretischer Teil

Um die Problemstellung und deren Lösung nachvollziehen zu können, sollten die Grundlagen von Machine Learning bekannt sein. Im Folgenden werden verschiedene Trainingsarten und Modelle beleuchtet, um ein grundsätzliches Verständnis für die Materie zu entwickeln.

## 2.1 Grundlagen von Machine Learning

Machine Learning ist die Wissenschaft einem Computer zu vermitteln flexibel auf eine Problemstellungen der realen Welt zu reagiert. Dies soll durch das Zuführen von Informationen oder der Betrachtung von Sachverhalten bewerkstelligt werden. Im Grunde soll eine Maschine nicht mehr lineare Probleme lösen, für die eigens erstellte Funktionen oder Programme geschrieben wurden müssten, sondern durch Training und Erfahrung auf Problemstellungen reagieren zu können.

**Definition 1** *"Machine learning is the science of getting computers to act without being explicitly programmed." -Stanford*[1]

Es gibt mehrere Arten von Machine Learning, unabhängig welche Art ein Algorithmus widerspiegelt er muss muss einen Trainingsprozess durchlaufen. Ein Trainingsprozess besteht darin einer Maschine Zusammenhänge zwischen Daten oder Eigenschaften zu vermitteln. Der Mensch lernt beispielsweise in der Schule oder von seinen Eltern, dass eine rote Ampel bedeutet, stehen zu bleiben. Ab jetzt verbindet er eine rote Ampel mit stehen bleiben, also einer Reaktion. Genauso muss eine Maschine trainiert werden, um beim Auftreten gewisser Eigenschaften bestimmte Reaktionen zu zeigen. Ein naheliegendes Beispiel findet man in der Bilderkennung wieder. Nachdem ein Algorithmus mehrere Bilder von Katzen vorgesetzt bekommen hat, kann dieser erkennen ob ein Bild eine Katze ist oder nicht.

### 2.1.1 Trainingsarten

Es gibt verschiedene Ansätze Künstliche Intelligenzen zu trainieren, aber es reicht zwei sehr gegenteilige Arten zu beleuchten. Zum einen Supervised und zum Anderen Unsupervised Learning. Im Folgenden werden beide detaillierter beschrieben.

#### Supervised Learning

*Supervised Learning* oder Überwachtes Lernen ist ein Ansatz, der für das Trainieren von Künstlichen Intelligenzen verwendet werden kann. Zuerst muss ein Datenset vorliegen mit Hilfe dessen eine KI trainiert werden kann. Wie ein Datenset auszusehen hat und wie eines erstellt werden kann, wird in 2.2 genauer beschrieben.

[1] https://www.coursera.org/learn/machine-learning

Als anschauliches Beispiel für Supervised Learning wäre ein Algorithmus, der erkennt ob ein unbestimmtes Objekt ein Apfel oder eine Birne ist. Jedes Trainingsobjekt trägt ein vorab bestimmtes *Label* [3, p. 103]. Das bedeutet jedes Objekt im Trainingsdatenset ist einer bestimmten Klasse zugeordnet. Während des Lernprozesses lernt der Algorithmus anhand der Labels, wann ein Objekt ein Apfel oder eine Birne ist. Die Bezeichnung überwachtes Lernen hat ihren Ursprung in der vorhergehenden Verteilung der Label. Es ist während dem Training permanent bekannt zu welcher Klasse ein Objekt gehört. Kurz gesagt wird Erfahrung benutzt um mehr Erfahrung zu erlangen. Im Idealfall kann der Algorithmus nach der Trainingsphase ein unbekanntes Testobjekt der Klasse Apfel oder Birne zuordnen.

#### Unsupervised Learning

Im Gegensatz zu Supervised Learning verzichtet das Unsupervised Learning auf Label im Datenset [2]. Supervised Learning wird eher verwendet um Ergebnisse vorherzusagen, wohingegen Unsupervised Learning hilft Daten oder Datenstrukturen besser zu verstehen. Es geht darum versteckte Strukturen im Datensatz zu erkennen oder Datenmengen zusammenzufassen. Das Ziel ist es Daten anhand ihrer Gemeinsamkeiten oder Unterschiede zu sortieren und neue Gruppierungen zu entdecken, die vorher eventuell nicht ersichtlich waren. Diesen Vorgang nennt man *Clustering*. Beide Arten verkörpern komplett verschiedene Herangehensweisen, ihr sinnvoller Einsatz hängt von der Problemstellung ab. Diese Arbeit bedient sich des Supervised Learnings, da es nicht um die Gruppierung von Daten geht, sondern das genaue Zuordnen der Objekte zu bestimmten Klassen.

### 2.1.2 Modelle

Nachdem eine Art des Lernens gewählt worden ist, muss die Problemstellung genauer definiert werden. Zur besseren Einschränkung gibt es mehrere Herangehensweisen. Die verbreitetsten sind Regression und Klassifizierung. Es gibt Algorithmen die beide Wege einschlagen können und manche die nur exklusiv einer Seite angehören.

#### Regression

Regressionsanalyse ist ein mathematisches Verfahren welches zum Abschätzen von Ergebnissen verwendet wird [3, p. 105,106]. Im Machine Learning Bereich kommt die Variante der *Linear Regression* zum Einsatz. Als Grundlage betrachtet man einen Vector $x \epsilon \mathbb{R}^n$ als Inputvektor und versucht einen Wert $y \epsilon \mathbb{R}$ zu generieren.
Der Output ist folgenermaßen definiert:

$$y = w \top x \tag{1}$$

[2] http://oliviaklose.azurewebsites.net/machine-learning-2-supervised-versus-unsupervised-learning/

$w$ repräsentiert in diesem Fall die Gewichtung der jeweiligen Inputparametern. Um beim vorherigen Beispiel zu bleiben, soll der Reifegrad eines Apfels anhand des Datums und der Größe des Apfels bestimmt werden. Das Modell soll anhand vorher gelernter Reifegraden entscheiden weclher Reifegrad vorhanden ist. Da es nicht möglich ist die vorausgesagten Werte auf Korrektheit zu prüfen bedient man sich der sogenannten *Loss Functions*. Als Beispiel wird hier die *mean squared error* Funktion aufgeführt. Um einen Wert zu erhalten werden vorab Testobjekte bestimmt und deren Inputparameter dem Algorithmus übergeben. Wenn $z$ die Vorausgesagten Werte für $y$ beinhaltet lautet die Formel:

$$MSE_{test} = \frac{1}{m} \sum_{i} z^{(test)} - y^{(test)} \tag{2}$$

Das Resultat der Funktion geht gegen null je genauer die vorhergesagten Werte den Testwerten entspricht.

#### Klassifizierung

Klassifizierung basiert auf der statistischen *Logistic Regression*, in der die Wahrscheinlichkeit bestimmt wird wann ein Objekt zu einer bestimmten Klasse zu zählen ist. Eine Klassifizierung setzt einen Datensatz voraus, der durch eine Aufzählung von diskreten Attributen beschrieben wird [1]. Bei diesem Verfahren werden Objekte anhand ihrer Eigenschaften zu bestimmten Klassen zugeordnet. Aus dem Datensatz lässt sich ein Trainingsdatensatz erstellen, bei der jedes Objekt vorher einer Klasse zugeordnet wird. Auch hier sind verschiedene Eigenschaften zu beachten. Einige verschiedene Vertreter von Klassifizierungsarten sind numerische, statistische oder parametrische Verfahren. Ziel einer Klassifizierung ist es Regeln aufzustellen, sodass jedes zu klassifiziernde Objekt einer Klasse zugeordnet werden kann.
Es ist relevant wie viele Klassen es gibt, also Binäre Klassifizierung oder *multiclass* Klassifizierung. Andererseits können Datensatzeinträge mehrere Label aufweisen und somit zu mehreren Klassen zugehörig sein, diesen Fall nennt man *Multilabel* [12]. Vor Beginn des Trainings muss festgelegt werden wie viele Labels und Klassen vorhanden sind.
Darüber hinaus muss bei einer Klassifizierung festgelegt werden wie hoch die *Kosten* einer fehlerhaften Klassifizierung sind. Die Kosten werden in der Regel durch eine *loss function* beschrieben. Detailierte Angaben über Auswirkungen und Vorgaben werden in 2.2.1 beschrieben.

## 2.2 Datenvorbereitung

Um eine Künstliche Intelligenz unabhängig welcher Art zu trainieren, ist es notwendig strukturierte Daten zu verwenden. Da in der Realität strukturierte Daten oder Datensätze eher eine Seltenheit sind, ist es zwangsläufig notwendig, die große Menge an unstrukturierten Daten in ein sinnvolles Format zu konvertieren.

### 2.2.1 Startbedingungen des Daten Preprocessings

Ein Datensatz besteht aus einer Sammlung von Objekten, hier E-Mails, welche so aufbereitet werden, dass ein Classifier trainiert werden kann. Es gibt einige Faustregeln um gute und effektive Datensätze zu generieren. Zum einen trifft hier die Regel "je mehr desto besser" vollkommen zu. Je größer der Datensatz den ein Classifier zur Verfügung hat, desto genauer kann er sich auf jede Art von Anfrage einstellen. Beispielsweise beinhalten größere Datensätze öfter Sonderfälle auf diese sich ein Classifier vorbereiten kann, was bei schlankeren Datensätzen nicht möglich wäre. Zum Anderen ist es fast wichtiger einen aussagekräftigen Datensatz zu verwenden, der repräsentativ für die Problemstellung ist. Je genauer diese definiert ist und je besser die Datenobjekte diesen beschreiben, desto bessere Resultate entstehen.
Ein weiterer wichtiger Faktor ist die Balance der Datensätze. Viele Klassifizierungsproblemstellungen für Texte sind unbalanciert, dies führt zu Problemen durch die *Laplace Korrektur*[2] Nehmen wir an eine KI muss zwischen relevanten E-Mails von Kunden oder Spam und unwichtigen E-Mails unterscheiden. Der vorhandene Trainingsdatensatz besteht aber aus 80% relevanten und 20% irrelevanten Mails. Hier kann das sogenannte *class imbalance problem*[9] auftreten. Bei einer fehlenden Balance der Dateien ist es für jeden Classifier schwerer zu erkennen, wann eine E-Mail eine Spam-Mail oder eine Kundenanfrage ist. Das Ergebnis leidet signifikant unter den suboptimalen Startbedingungen, deshalb ist es von großen Wert balancierte Datensätze zur Verfügung zu haben. Ohne ein sauberes Datenpreprocessing wird es schwer präzise arbeitende Classifier zu erstellen.

### 2.2.2 Konvertieren zu Featureraum

Mit diesen Grundsätzen soll nun aus dem oben erwähnten unstrukturiertem "Blob" ein gut aufgesetzter Datensatz werden. Diese Arbeit verwendet E-Mails als Datengrundlage, was bedeutet dass diese textbasierten Nachrichten zu einem maschinenlesbaren oder maschinenverarbeitbaren Konstrukt werden müssen. Dabei werden verschiedene E-Mails verglichen um Eigenschaften zu finden, die einem Algorithmus beim Klassifizierungsprozess weiterhelfen können. Solche Eigenschaften werden *Features* genannt und das Konzept hinter den Fragen "Welche Features gibt es?" und "Welche Features sind sinnvoll?" nennt man Feature Engineering.
In einem realen Umfeld treten häufig symbolische und numerische Features auf, die jeweils eine andere Art Algorithmus zur Verarbeitung benötigen. Darüber hinaus wird ein Datenset in der Realität durch zu viele unwichtige Features repräsentiert. Folglich muss gewährleistet sein, dass alle Features diskreter Natur sind und nicht relevante Features entfernt werden. [8].
Um die Effizienz eines Algorithmus zu verbessern, werden kontinuierliche Features mit *discretization* Algorithmen umgewandelt. Ein Classifier der einen kontinuierlichen Featureraum zur Verfügung hat, arbeitet wesentlich langsamer und

ineffizienter [8]. Nachdem sinnvolle Features ausgewählt wurden, beginnt das eigentliche Preprocessing. Das Ziel der Vorverarbeitung ist es so viele unnötige und störende Informationen aus dem Rohformat der Daten zu entfernen, damit ein Classifier optimal arbeiten kann. [10] Als ersten Schritt entfernt man so genannte *Stopwords* aus dem Fließtext. *Stopwords* sind Wörter, die keinen semantischen Mehrwert liefern. Wörter wie "und, aber, so, indem, etc." helfen nicht Inhalte von Texten besser zu verstehen. Oft trifft eher das Gegenteil zu, da mehr Text zu verarbeiten ist. Solche Störfaktoren werden *noisy features* genannt und interferieren den Prozess.
Das Entfernen von einzelnen Wörtern und Links ist eine Standartprozedur im *Natural Language Processing* Bereich.
Das Produkt ist ein diskreter teilweise optimierter Featureraum. Meistens existieren große Unterschiede zwischen den Größen der einzelnen Features. Die naheliegende Lösung ist eine Normalisierung der Features um einen besseren Vergleich zu erhalten. Vorallem Neuronale Netze und K-Nearest Neighbor Algorithmen profitieren von Normalisierung.

### 2.2.3 Fehlerquellen im Datensatz

Eine weitere Hürde, die auf dem Weg zu einem guten Machine Learning Ansatz genommen werden muss, besteht darin *Overfitting* oder *Underfitting* [3, p. 108,109] zu vermeiden. Overfitting tritt auf wenn, das Machine Learning System zu spezifisch auf die Problemstellung trainiert wurde. Das bedeutet ein Classifier trainiert sich jedes Detail des Trainingdatensatzes so genau an, dass dies negativen Einfluss auf das Ergebnis beim Klassifizieren fremder Daten aufweist. Dieses Phänomen tritt beispielsweise bei schlecht balancierten Datensätzen auf, da schnell eine geringe Diversität zwischen den Daten auftreten kann. Ein weiterer Standardfehler tritt schon bei der Auswahl der Testdaten auf, indem nur Daten ausgewählt werden, die dem Idealzustand entsprechen oder keine signifikanten Abweichungen zu anderen Einträgen im Datensatz aufweisen. Das Zusammensetzen eines guten Datensets baut auf guter Generalisierung auf. Generalisierung beschreibt wie erfolgreich ein Sytsem auf jegliche Art von Daten innerhalb der Domäne reagieren kann. Generalisierung lässt sich anhand eines Beispiels erklären:
Angenommen ein Algorithmus muss zwischen Äpfeln und Birnen unterscheiden, da aber als Trainingsobjekte nur rote Äpfel zur Verfügung stehen, würde der Algorithmus bei Klassifizieren von grünen Äpfeln schlechter abschneiden. Das ist das Resultat einer schlechten Generalisierung, da eine hohe Diversität im Datenset zu Um Overfitting zu vermeiden werden verschieden Methoden angewandt, beispielsweise bei der Verwendung eines Neuronalen Netzes. Ein Ansatz wird *Dropout* [11] genannt und lässt einen Algorithmus den Lernprozess abbrechen wenn ein Rückgang der Lernfortschritte zu verzeichnen ist.
Das Pendant zum Overfitting wird Underfitting genannt.Im Gegensatz zum Overfitting wird hier weder das System ausreichend trainiert, noch ist es ausreichend generalisiert.
Ein Sonderfall der Datenvorbereitung sind sogenannte *Outlier*. Als Outlier beze-

ichnet man beispielsweise falsch gelablete Trainingsdaten oder zu spezifische Sonderfälle. Outlier können enorme negative Auswirkungen auf manche Classifier haben, da alle Ergebnisse auf den Trainingsdaten aufbauen. Die Forschung hat sich intensiv mit dem Handling von Outliern beschäftigt und eine große Menge an Methoden zur Problembearbeitung zur Verfügung gestellt, dies ist aber nicht Inhalt dieser Arbeit.

## 2.3 Wahl der Algorithmen

Um ein Klassifizierungsproblem zu lösen gibt es eine breite Masse an Algorithmen. Diese Arbeit beschäftigt sich mit Neuronalen Netzen, Support Vektor Machines, K-Nearest-Neighbor Algorithmus und dem Naive Bayes Ansatz. Darüberhinaus werden Alternativen kurz angeschnitten.

### 2.3.1 Neuronales Netz oder Multi Layer Perception Classifier

Der Grundgedanke hinter Neuronalen Netzen basiert auf dem menschlichen Gehirn, indem es künstliche Neuronen zur Verarbeitung von Daten zur Hilfe nimmt. Zuerst muss man nachvollziehen können wie ein künstliches Neuron arbeitet. Ein künstliches Neuron bekommt eine bestimmte Anzahl an Inputs und generiert einen einzigen Output, wobei ein Standardneuron generell binäre Eingabewerte verarbeitet. Der generierte Output entsteht wenn die Summe aller Inputs einen Schwellenwert unter- oder überschreiten. Bei Standardneuronen ist der Output folglich entweder Eins oder Null. Inputs sind teilweise gewichtet, da manche Parameter größeren Stellenwert inne haben, dadurch kann die Berechnung des Schwellenwertes maßgeblich beeinflusst werden. Die Berechnung wird später genauer beschrieben.
Da ein Neuronales Netz aus mehreren Neuronen besteht entstehen drei verschiedene Arten von Schichten, die so genannten *Layer*. [3]

-Input-Layer: Nimmt alle eingehenden Umgebungsvariablen ungefiltert auf und leitet diese gefiltert an den Hidden-Layer weiter.

-Hidden Layer: Als Hidden-Layer werden alle Layer bezeichnet, die nicht Output- oder Input-Layer sind. Es können beliebig viele Hidden-Layer hintereinander angeordnet werden. Diese haben durch die Verarbeitung der vorhergehender Layer eine abstraktere Sicht auf die Eingabewerte und verarbeiten genau wie ein Input-Layer mehrere Inputvariablen zu einem Output Wert. Wenn mehrere Hidden-Layer zum Einsatz kommen, spricht man von *Multilayer Perception*.

- Output Layer: Der letzte Layer zeigt das Ergebnis des Algorithmus an.

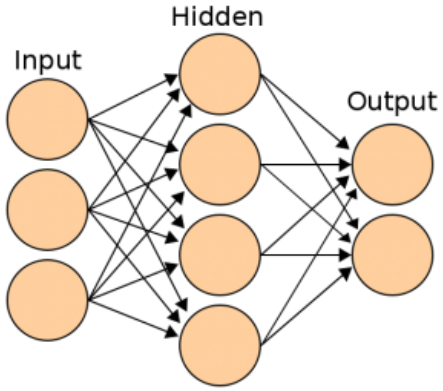


Figure 1: Aufbau eines Neuronales Netz mit neun Neuronen und einem Hidden-Layer

In der Regel wird zusätzlich zum Schwellenwert eines Neurons ebenso eine Gewichtung mit eingerechnet. Diese Gewichtung wird als Delta zur Summe der Inputs gerechnet. Ein Neuron kann einfache logische Operationen durchführen, indem es **AND, OR** und **NAND** als Operatoren zur Verfügung hat. Wie ein Neuron arbeitet lässt sich anhand Figure 2 2 erklären.
In der Abbildung sind beide Kanten gewichtet da sie jeweils den Wert -2 inne haben. Diese Gewichtung fließt in die Berechnung des Outputs mit ein. Angenommen der Input von $x1$ und $x2$ besteht aus 0 und 1, daraus ergibt sich folgender Term:

$$(-2) * 0 + (-2) * 1 + 3 = 1 \tag{3}$$

Das Ergebnis ist positiv also gibt das Neuron eine 1 als Resultat aus. Analog dazu wird bei negativen Werten oder 0 eine 0 ausgegeben. [3]

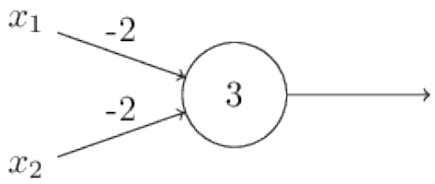


Figure 2: Beispiel eines Perceptrons mit zwei Kanten $x1$ und $x2$ mit der Gewichtung -2 und dem Perceptron selbst mit Gewichtung 3

Ein untrainiertes Neuronales Netz besteht aus ungewichteten Neuronen und Kanten. Wird ein Neuronales Netz trainiert, verändert sich diese Ausrich-

tung der Neuronen, um sich den Eingabeparametern anzupassen. Da die Outputs der Neuronen binärer Natur sind und das Ergebnis der Kalkulation der Eingabeparameter entweder Eins oder Null ergibt, kann dies durch die leichte Anpassung der Gewichtung in ein völlig anderes Ergebnis resultieren. Beispielsweise entscheidet ein Neuronales Netz ob ein Objekt ein Apfel ist oder nicht. Während des Trainings passen sich die jeweiligen Neuronen an und es ändern sich einzelne Gewichtungen. Diese kleinen Änderungen können bei der Entscheidung "grüner" oder "roter" Apfel so viel Einfluss haben, dass ein Wert den Output komplett verändert. Um dem entgegenzuwirken werden *Sigmoid Neuronen* verwendet. Diese Art der Neuronen bedient sich bei der Berechnung, ob ein Schwellenwert überschritten wird oder nicht, der so genannten Sigmoidfunktion.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

Wenn die Sigmoidfunktion mit einfließt, ergibt sich eine Kurve für alle Eingabeparameter. Also können Werte anstatt nur 0 und 1, auch Fließkommazahlen annehmen. Dadurch haben kleinere Änderungen an der Gewichtung der Neuronen, weniger Auswirkung auf das Endergebnis[4].
Während des Trainings muss ein Neuronales Netz so viele verschiedenen Arten der Gewichtung testen, um die beste Kombination zu finden. Diese Durchläufe nennt man *Epochen* und müssen abhängig von der Komplexität des Netzes sehr oft durchlaufen werden. Um die optimale Anzahl von durchlaufenden Epochen zu ermitteln, gibt es viele Ansätze, die aber zum Verständnis Arbeit nicht zuträglich sind.

### 2.3.2 Support Vector Machine

Ein sehr weit verbreiteter Ansatz ist die Verwendung von sogenannten *Support Vector Machines* abgekürzt SVM. SVMsprädestiniert für die Verwendung bei Klassifizierungsproblemen. Jedes vorhandene Trainingsobjekt wird als Vektor in einem N-dimensionalen Vektorraum dargestellt, wobei N gleich der Featureraumgröße. Die Aufgabe des Algorithmus ist es im Vektorraum eine optimale Trennfläche, eine so genannte Hyperebene, zu finden. Diese Hyperebene trennt im Idealfall die vorhandenen Klassen.
Dabei ist zu beachten, dass es mehrere Hyperebenen gibt. Es wird die Hyperebene gewählt, welche den maximalen Abstand zu den anliegenden Vektoren inne hat. Vektoren die nicht im Bereich der Hyperebene liegen nennt man Stützvektoren, da sie gebraucht werden um die Ebene mathematisch exakt zu beschreiben. Eine Voraussetzung um diese Ebene zu errechnen ist, dass alle Objekte linear trennbar sind.
Bei der Verwendung von realen Trainingsobjekten ist das in der Regel nur schwer möglich. Wie in Abbildung x zu sehen, ist eine lineare Trennung der Klassen nicht möglich, deshalb wendet die SVM den *Kernel Trick* an.
Der Kernel Trick beinhaltet das Überführen des Vektorraums in einen höher

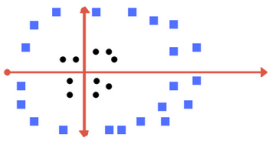

Figure 3: Beispielfall für Kernel Trick

dimensionalen Raum, teilweise bis ins Unendliche. Dadurch können selbst die komplexesten Vektorräume linear getrennt werden. Im Beispiel in Abbildung X wird eine Weitere Achse hinzugefügt, die es ermöglicht eine klare Hyperebene zwischen den Klassen zu ziehen. Danach wird der Vektorraum wieder in seine Ursprungsdimension zurückgeführt, wobei die Funktion der entstandenen Hyperebene noch vorhanden ist. Eine SVM ist durch das ausschließliche Betra-

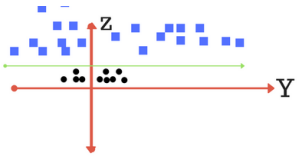


Figure 4: Vektorraum in höhere Dimension überführt und Bildung der Hyperebene

chten der Stützvektoren und der Hyperebene resistent im Falle von auftretenden Outliern, da diese selten beim erstellen der Hyperebene berücksichtigt werden. Die Nachteile der Anwendung finden sich zum Einen im Berechnen der höheren

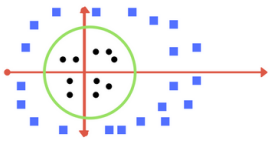

Figure 5: Rückführung zur Ursprungsdimension und beibehalten der Hyperebene

Dimension und zum Anderen beim Verarbeiten von realitätsnahen Datensätzen wieder. Oft liegen die Trainingsobjekte sehr unvorteilhaft im Vektorraum um eine saubere Hyperbene leicht finden zu können. Um dem entgegenzuwirken zu können, gibt es zwei Hyper-Parameter *Gamma* und *Regulierungsparameter*, die für eine höhere Genauigkeit angepasst werden können.

**Gamma** Der Gamma-Parameter legt fest inwiefern ein Trainingsobjekt relevant für das Berechnen der Hyperebene ist. Ein niedriger Wert zieht Objekte in Betracht, die weiter von der Hyperebene entfernt liegen. Große Gamma-Werte arbeiten mit Trainingsobjekten, welche näher an der Hyperbene angesiedelt sind.

**Regulierungsparameter** Dieser Parameter legt fest wie relevant eine fehlerhafte Klassifizierung für den Classifier ist. Ein hoher Wert resultiert in einer höheren Genauigkeit beim Klassifizieren, aber das Gesamtbild des Vektorraums tritt in den Hintergrund. Wird ein niedriger Wert gewählt, nutzt der Classifier einen größeren Blick, selbst wenn das in falsche Klassifizierungen resultiert.

### 2.3.3 K-Nearest Neighbor

K-nearest neighbor, oft KNN genannt, ist eine der älteren Klassifizierungsmethoden und dazu noch eine sehr simple. Das Grundprinzip des Algorithmus basiert auf dem Vergleichen der einzelnen Trainingsobjekten. Diese Trainingsobjekte werden in einen Vektorraum überführt, dabei wird jedes einzelne Trainingsobjekt durch einen Punkt repräsentiert. Die Dimension des Raumes wird durch die Featureanzahl bestimmt. Wenn der Algorithmus ein unbekanntes Objekt einer Klasse zuordnen will, bedient er sich einer vorher festgelegten Anzahl seiner Nachbarn. Da nur die Eigenschaften der abgespeicherten Nachbarn in die Klassifizierung mit einfließen, spricht man von *lazy learning* [3]. Lazy Learning bedeutet das Trainingsdaten nicht verarbeitet, sondern nur abgespeichert werden und bei Bedarf jederzeit zur Verfügung stehen.
Angenommen im Vorfeld wurde festgelegt, dass die fünf nächsten Nachbarn eines unbekannten Objekts, das im Vektorraum repräsentiert wird, die Klasse des Objektes bestimmen. So wird jedes Label der Nachbarn betrachtet und evaluiert welche Klasse am häufigsten unter diesen fünf Nachbarn auftritt. Das Objekt

[3] http://scholarpedia.org/article/K-nearest_neighbor

wird der Klasse zugeordnet, welche am häufigsten unter den betrachteten Nachbarn aufgetreten ist. Die Abstände der Trainingsvektoren werden durch den *Euklidischen Abstand* bestimmt. Der Abstand wird durch eine Funktion $d(x, y)$ beschrieben, wobei $x, y$ zwei mögliche Ergebnisse der Entscheidung Classifiers sind. Der Featureraum wird hier durch $N$ dargestellt. Daraus ergibt sich folgende Formel:

$$d_E(x, y) = \sum_{i=1}^{N} \sqrt{x_i^2 - y_i^2} \tag{5}$$

Ein Ansatz um die Arbeitsweise von K-Nearest Neighbor Algorithmen zu optimieren ist die Verwendung von *Voronoizellen* [7]. Dadurch dass nur begrenzte Trainingsobjekte vorhanden sind, wird bei der Darstellung klar, das Bereiche um die Datenpunkte nicht einfach freier Raum sind, sondern zur Klasse des des Datenpunktes zählen. Ein solcher Raum wird *Voronoizelle* genannt. Dieser ergibt sich durch den Vergleich zu anderen Vektoren. Durch die Verwendung des euklidischen Abstands, ist eine Voronoizelle der Raum in dem ein möglicher Punkt immer am nächsten dieser einen Zelle im Vergleich zu anderen liegt. Eine Voronoizelle ist immer ein Polygon. Ein Beispiel für ein Voronoi-Diagramm ist in Grafik 6 dargestellt.
Diese Raumaufteilung erleichtert das Klassifizieren von Testobjekten ungemein.

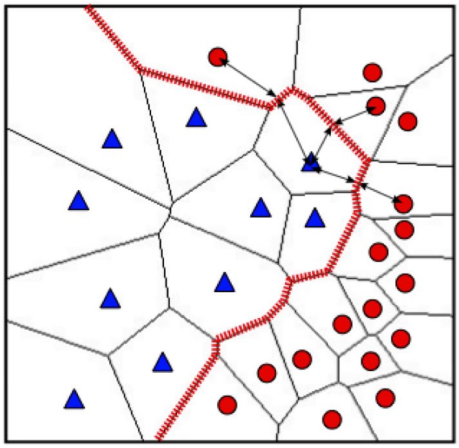

Figure 6: Beispiel einer Voronoi-tessalation

Das Testobjekt wird zu der Klasse gezählt, welche der Klasse der Voronoizelle entspricht. Die Berechnung der Voronoizellen wird nur einmal während des Trainings ausgeführt, dadurch fällt das Finden der nächsten Nachbarn für jedes Testobjekt weg. Bei einer größeren oder engeren Ansammlung der Trainingsdaten, werden die Zellen automatisch kleiner, folglich kann genauer zwischen den einzelnen Trainingsobjekten unterschieden werden. Im Verglich zu SVM oder Entscheidungsbäumen, wird in 2.4.1 erläutert, ist eine viel flexiblere Un-

terteilung der Datensätze möglich. Andere Ansätze wie SVM verwenden Hyperebenen oder vertikale Unterteilungen. Der KNN Algorithmus kann sogar Parabeln oder Hyperbeln darstellen.
Leider bringt Flexibilität auch ihre Nachteile mit sich. Der Algorithmus *overfittet* sehr leicht und hat große Probleme mit Outliern. Angenommen ein Trainingsobjekt vorab falsch gelabled. Bei anderen Classifiern würde dies im Laufe des Trainings immer irrelevanter und wenig Einfluss auf das Endergebnis haben. Für den K-Nearest Neighbor Ansatz können die Auswirkungen von Outliern katastrophal sein, da jedes Testobjekt, welches in der falsch ausgeschriebenen Voronoizelle liegt, auch falsch gelabled wird.
Um dem entgegenzuwirken und den Algorithmus robuster zu gestalten, vergleicht man nicht nur ein Sample mit dem Testsample sondern mehrere gleichzeitig. Um ein Testsample zu einer Klasse zuzuordnen wird ein $k$ bestimmt, welches festlegt wie viele Samples mit dem Testsample verglichen werden soll. Beispielsweise wird k=3 festgelegt, dann wird nach den drei samplen gesucht, die am nächsten an unserem Testsample liegen. Bei einer Zweiklassenaufteilung wird das Testsample zu der Klasse zugeordnet, welche die zwei von drei Samples stellt. Im Idealfall wird K immer ungerade gewählt und nicht als ein Vielfaches der vorhandenen Klassen. Das erspart falsche Klasseneinteilung die zustande kommen würde, wenn nur das nächste Sample verglichen werden würde.

### 2.3.4 Naive Bayes

Ein Bayes Klassifizierer basiert auf dem Satz von Bayes und Klassifziert anhand Wahrscheinlichkeiten.

**Definition 2** *Es besteht ein Verhältnis zwischen der bedingten Wahrscheinlichkeit zweier Ereignisse $P(A|B)$ und der umgekehrten Form $P(B|A)$.*

Unter dem Begriff der Bayes Classifier ist eine Ansammlung von verwandten Algorithmen gemeint, die alle ähnliche Vorgehensweisen beim Klassifizieren aufweisen. Auch dieser Classifier arbeitet mit einem diskreten Featureraum. Ein Bayes Classifier versucht ein *A-posteriori Wahrscheinlichkeitsmodell* zu erstellen. A-posteriori beschreibt einen Wissensstand über einen unbekannten Umweltzustand. Diesen Wissensstand zu erhalten gestaltet sich schwierig, wenn alle mit einspielenden Faktoren oder Features *abhängig* von einander sind. Deshalb nimmt der Bayes Ansatz an, dass alle Features *unabhängig* von einander sind. Ein realitätsferner Ansatz, der aber in der Regel zu Herausragenden Ergebnissen führt. Vorallem in hoch dimensionalen Featureräumen, findet der Bayes-Ansatz Verwendung. Man wählt $x_j$ Komponente $j$ von $x$ daraus ergibt sich folgende Funktion:

$$p(x|y) = \prod_i p(x_i|y) \tag{6}$$

Um eine Entscheidung treffen zu können müssen nur die nachfolgenden Werte bekannt sein, um auf $x$ schließen zu können, somit muss nicht $p(x)$ gelöst werden,

denn es reicht wenn $y$ so gewählt wird, dass $\prod_i p(x_i|y)$ den höchsten Wert ergibt. Für $x_i$ können verschiedene Arten von Werten gewählt werden, beispielsweise führt eine binäre Variable zu einer *Bernoulliverteilung*. Es gibt diverse Varianten des Naive Bayes Algorithmus abhängig von der Variablenwahl. Um den Vorgang besser zu verstehen betrachtet man folgendes [4]Beispiel.

| Fruit | Long | Sweet | Yellow | Total |
|---|---|---|---|---|
| Banana | 400 | 350 | 450 | 500 |
| Orange | 0 | 150 | 300 | 300 |
| Other | 100 | 150 | 50 | 200 |
| **Total** | **500** | **650** | **800** | **1000** |

Figure 7: Beispieltabelle mit Features: Long, Sweet, Yellow und Klassen: Banana, Orange, Other

Für den Algorithmus ist wichtig wie die Früchte auf die Gesamtmenge verteilt sind, also 50% Bananen, 30% Orangen und 20% Andere. Außerdem spielen Werte wie viele Bananen haben das Feature *Long* oder wie viele Orangen sind *Sweet* mit in die Berechnung ein.
Nun muss der Algorithmus anhand der vorher bekannten Parameter erkennen ob eine unbekannte Frucht, deren Features bekannt sind, eine Banane, Orange oder eine andere Frucht ist. Angenommen die unbekannte Frucht besitzt die Features **Long,Sweet,Yellow**. Der Classifier errechnet nun die Wahrscheinlichkeit pro Klasse wie Wahrscheinlich eine Frucht in diese Klasse gehört, daraus ergeben sich folgende Terme:

$$P(Banana|Long, Sweet, Yellow) = 0.8 * 0.7 * 0.9 * 0.5 = 0.252 \quad (7)$$

$$P(Orange|Long, Sweet, Yellow) = 0 * 0.5 * 1 * 0.3 = 0 \quad (8)$$

$$P(Other|Long, Sweet, Yellow) = 0.5 * 0.75 * 0.25 * 0.2 = 0.01875 \quad (9)$$

Es ist sehr wahrscheinlich ,dass die unbekannte Frucht der Klasse Banane zuzuordnen ist, da die Formel der Klasse Banane höchste Wahrscheinlichkeit ergibt.
Der Naive Bayes-Ansatz benötigt keine großen Datensätze um solide Ergebnisse zu liefern. außerdem spielen störende Features eine geringere Rolle bei der Wahrscheinlichkeitsberechnung, da angenommen wird, dass keine Abhängigkeit zwischen den einzelnen Features existiert. Dieser Vorteil kann aber auch ein Nachteil sein, da zwischen Features im realen Umfeld oft ein kausaler Zusammenhang herrscht.

## 2.4 Alternativen

Es gibt eine Vielzahl von Klassifizierungsalgorithmen, die ebenso gute Ergebnisse erzielen könnten. Einige beliebte und bewehrte Algorithmen werden hier

[4]http://blog.aylien.com/naive-bayes-for-dummies-a-simple-explanation/

vorgestellt.

### 2.4.1 Descision Tree

Jeder Classifier stellt seine eigenen Klassifizierungsregeln auf und versucht so ein Datenset in seine Klassen einzuteilen. Eine Variante bedient sich mehrerer einfacher Tests, wobei jeder nachfolgende Test die Ergebnisse, des vorherigen Tests weiterverwendet. Lässt man ein Testobjekt durch diese Testsequenz laufen entsteht eine Baumstruktur in der jeder Knoten einen Test, und jede Kante ein mögliches Ergebnis des Tests darstellt. Nachdem alle Testdurchläufe beendet sind, erreicht man die Blätter dieser Baumstruktur, in der das Testobjekt mit einem Label versehen wird. Das Klassifizieren durch diese Baumstruktur nennt man *Descision Tree*. Beim Erstellen eines Descision Tree spielt der Zufall eine große Rolle, da das Datenset nach jedem Test halbiert wird, entsteht bei jedem neuen lernen des Classifiers ein anderer Baum. Jedes Testobjekt wird in den Blättern gelabled, also ist ein Descision Tree prädestiniert für Multiclass Klassifizierungsprobleme.

### 2.4.2 E-Mail Classification with Co-Training

Ein Alternativer Ansatz ist die Verwendung von Co-Training[5] Algorithmen. Co-Training verwendet, ebenfalls eine kleine Menge labeled-Data. Beim Beispiel der Klassifizierung werden zwei verschiedene *classifier* erstellt, welche sich jeweils um ein anderes Klassifizierungsproblem kümmert. Beide Elemente werden mit den wenig vorbereiteten Daten trainiert. Danach arbeiten beide nur mit unlabled-data und unterstützen sich gegenseitig.
Classifier1 bekommt einen unbearbeiteten Datensatz zur Bearbeitung, welcher mit einer hohen Wahrscheinlichkeit zu einer bestimmten Klasse zu zählen ist. Classifier1 labeled das Ergebnis, welches danach auch für Classifier2 gelabeled ist. Für Classifier2 wäre die Entscheidung welche Klasse der Datensatz hat nicht so leicht gefallen, da es eine andere initiale Problemstellung erhalten hat. Somit arbeiten sich beide Classifier gegenseitig zu und unterstützen sich. Untersuchungen zu Folge kann der Einsatz von Co-Training die Präzision nach gleichen Trainingsiterationen um den Faktor 2 verbessern.
Weitere Erkenntnisse wurde durch das Anwenden der Methode mit Support Vector Machines und Naive Bayes erlangt. Zum einen ist die Accuracy und Precision, wird in 4.4.5 erklärt, stark von der Balance des Datensatzes abhängig. Außerdem ist die Effektivität der Methode sehr vom verwendeten Algorithmus abhängig. SVM erzielte deutlich bessere Ergebnisse als Naive Bayes, welches sogar negative Trainingsergebnisse bei schlecht balancierten Datensätzen lieferte.

## 2.5 Text Pattern Recognition

Ein ungewöhnlicher Ansatz bedient sich der Text Pattern Recognition[13]. Ein großteil der Textklassifizierungsansätze bedient sich der relevanten Schlagwörtern

in einem Text. Dabei wird für gewöhnlich die Auftrittshäufigkeit im Dokument über alle vorhandenen Dokumente zu Hilfe genommen, beschrieben in 4.4.3.

# 3 Features

## 3.1 Feature Engineering

Ein großer Schwerpunkt der Arbeit liegt auf der Anwendung von *Feature Engineering*.
Grundsätzlich ist der Begriff des Feature Engineering ein informeller Ausdruck, der im Bereich des Machine Learnings verwendet wird, um das Erstellen von Feature Sets zu beschreiben. Ein Feature ist eine Information, die eventuell bei Vorhersagen nützlich sein könnte. Angewandt bei diesem Beispiel wäre ein Feature die Wahl des Eingangsdatums oder der Absender. Beide Informationen können bei der Vorhersage helfen. Wenn ein bestimmter Absender, bei 80% der E-Mails, nur Quotinganfragen stellt, wäre die Wahl des Features sehr hilfreich für einen Classifier. Der Mangel an einfachen Heuristiken, um sinnvolle von nicht sinnvollen Features zu unterscheiden, macht das Feature Engineering zu einem komplexen und zeitaufwändigen Unterfangen.

## 3.2 Featurewahl

Da eine E-Mail ein Textdokument ist, gehen die ersten Schritte bei der Featurewahl in Richtung NLP, dem oben erwähnten *Natural Language Processing*. Zu den grundlegenden Features, die den semantischen Inhalt eines Textes widerspiegeln, fügt man Features, die von Metadaten der einzelnen E-Mailfiles Gebrauch machen hinzu.

### 3.2.1 NLP basierte Features

Um eine E-Mail einer Klasse zuzuordnen, ist es notwendig zu verstehen welchen Inhalt eine Nachricht vermittelt. Eine Maschine ist nicht wie ein Mensch der Schrift mächtig, noch kann sie das geschriebene lesen und verstehen. Um den Inhalt trotzdem zu verstehen gibt es die Möglichkeit eine besondere Gewichtung von bestimmten Wörtern abzuleiten. Da hier nach Angebotsanfrage und keiner Angebotsanfrage sortiert werden soll, sind Begriffe wie Preis, Angebot oder Kosten mögliche Schlagworte, die einem Classifier helfen können ein Objekt einer Klasse zuzuordnen. Um einen Text in eine numerische Repräsentation zu konvertieren, damit dieser maschinenlesbar wird, kann auf das so genannte *tfidf-Maß* zurückgegriffen werden.
Der Begriff setzt sich aus *term frequency*, Vorkommenshäufigkeit und der *inverse document frequency*, inversen Dokumenthäufigkeit zusammen.
Term frequency zählt, wie der Name schon sagt, die Auftrittshäufigkeit bestimmter Terme. Term ist ein Information Retrieval spezifischer Fachbegriff und beschreibt ein Wort in einem Text.
Die Inverse Dokumentenhäufigkeit zeigt auf, ob ein Term häufig oder selten über alle vorgegebenen Dokumente auftritt. Aus diesem Wert lässt sich ableiten wie viel Informationsgehalt in einem Term enthalten ist.

$t$ = Term der betrachtet wird

Formel für *term frequency*:

$$[ht]tf = \frac{TermCountInDocument}{TotalTermsInDocuments} \tag{10}$$

Formel für *inverse document frequency*:

$$idf = log\frac{N}{\sum_{D:t\epsilon D} 1} \tag{11}$$

Formal für *Tf-idf*:

$$tf.idf(t, D) = tf(t, D) * idf(t) \tag{12}$$

Beide Werte in Kombination ergeben den tf-idf Wert. Ein Wert ist dann hoch, wenn ein hohe Term frequency vorhanden ist und der ausgewählte Term in wenigen Dokumenten auftritt. Daraus kann ein hoher Stellenwert und Informationsgehalt des Terms abgeleitet werden.

Um Texte zu vergleichen, wird für alle Terme, die in jedem Dokument vorkommen ein so genanntes Vokabular erstellt. Anhand diesem wird für jedes Dokument der Tf-idf-Wert berechnet und abgespeichert. Jedes Dokument verwendet das gleiche Vokabular, aber nicht jedes Wort im Vokabular tritt in jedem Dokument auf, deshalb entstehen sehr lange Werte Tabellen. Im Falle der E-Mailklassifizierung kann der Tf-idf-Wert aus dem Inhalt der Mail und dem Betreff erstellt werden.

### 3.2.2 Sekundäre Features

Abgesehen vom offensichtlich wichtigem Inhalt, liefert eine E-Mail diverse andere Informationen, die beim Sortieren hilfreich sein können. Diese Art von Informationen nennt man Metadaten; Daten, die erst auf den zweiten Blick sichtbar werden. Ein Beispiel wäre die Anzahl an Anhängen einer E-Mail. In diesem Fall sind manche Metadaten mit Vorsicht zu verwenden, da nicht jede verfügbare Information gleich ein Hinweis in die richtige Richtung ist.
Im folgenden werden alle Metadaten, die zu Features konvertiert werden aufgeführt:

**Attachment** Dieses Feature zeigt, ob eine E-Mail eine angehängte Datei besitzt oder nicht. Die Überlegung hinter dieser Wahl war der manuelle Vergleich durch einen Menschen. Es war auffällig, dass Quotemails tendenziell keine Anhänge besitzen und vor allem Nicht-Quotemails, die Anfragen zu Bauteilen oder Bewerbungen waren, angehängte Dateien besaßen.

**Anzahl der Fragezeichen** Quotemails können auch als typische Anfragen gesehen werden, deshalb die Überlegung die Fragezeichen des Inhalts zu zählen. Eventuell gibt es eine Korrelation zwischen dem Auftreten der Fragezeichen und den Quotemails. Beim extrahieren der Fragezeichen traten Probleme auf, da zum einen in den meisten E-Mails URLs auftreten, welche oft als Trennzeichen oder Ankersymbole Fragezeichen verwenden. Diese müssen vorher entfernt werden. Zum anderen sind manche E-Mailobjekte Produkte langer Dialoge, was bedeutet, dass ein Mailobjekt aus vielen kleineren E-Mails bestehen kann. Je länger so ein Dialog fortgeschritten ist, desto mehr Fragezeichen treten im Objekt auf, nur dass die Auftrittsmenge nun kaum Rückschlüsse auf den Inhalt oder die Klasse der E-Mail gibt.

**Prüfen nach Q-Nummer** Ein naheliegender Ansatz, um bei der Sortierung schnell und einfach bessere Ergebnisse zu erlangen, ist es zu prüfen, ob eine E-Mail eine Bauteil-ID oder Nummer enthält. Die meisten Quotemails enthalten Bauteilbezeichnungen, nur unterscheiden sich die Bauteil-IDs so grundsätzlich voneinander, dass es schwer ist einen gemeinsamen Nenner zu finden. Eine Stufe über der Bauteil-ID steht die sogenannte Q-Nummer, welche unter sich mehrere Bauteile vereint. Leider wird die Q-Nummer selten in Quotemails verwendet, da sie oft nicht bekannt ist oder einfach zu ungenau beschreibt, um welches Bauteil sich die Anfrage dreht. Trotzdem sollten Q-Nummern exklusiv in Quotemails auftreten.

**Sendezeit** Als letztes sekundäres Feature wird die Zeit verwendet, zu der eine E-Mail abgeschickt worden ist. Generell ist es für einen Menschen fast unmöglich durch die Sendezeit einer E-Mail den Inhalt zu erkennen, für eine Maschine ist es fast genauso schwer, nur könnte in Kombination mit anderen Features eine Tendenz erkennbar sein. Angenommen vormittags würden nur Quotemails eintreffen und nachmittags keine, dadurch könnte die Maschine ein Muster erkennen und dementsprechend leichter sortieren.

Um sinnvolle Ergebnisse aus den gewählten Features zu erhalten, muss nun empirisch getestet werden, welche Kombination sinnvoll ist. Es gibt diverse Ansätze, die mit nur einem Feature eine Klassifizierung vorgenommen haben, manche sogar recht erfolgreich[14]. Beispielsweise reicht es oft aus nur den Betreff zu verwenden, um auf den Inhalt Rückschlüsse ziehen zu können. Angewandt auf diesen Usecase soll sich nun zeigen, wie sinnvoll die Featurewahl ausfällt.

# 4 Implementierung

Der Implementierungsabschnitt beinhaltet Details die den Usecase betreffen, der Architektur und den daraus folgenden Resultaten.

## 4.1 Usecase

Um die Problemstellung zusammenfassen zu können, muss der vorhandene Usecase genau definiert sein. Die Vision von Osram OS besteht in der Verwendung von teilweise automatisierten Prozessen, gestützt durch Künstliche Intelligenz. Genau ein solcher Prozess kann das Berarbeiten von E-Mails in der Sales-Abteilung des Konzerns definieren. Die Inbox der Mitarbeiter ist mit einer großen Anzahl von Standardmails gefüllt, die zu beantworten und verarbeiten einen langen Zeitraum in Anspruch nehmen würde. Eine spezielle Art dieser E-Mails sind so genannte *Quote* E-Mails, welche mit einer Angebotsanfrage gleichzusetzen sind. Eine standard Quoteanfrage beinhaltet die Anfrage der Kosten, der Produkt-ID, einer Mengenangabe des gewünschten Bauteils und deren Lieferzeit. Der nächste Schritt beinhaltet das Sortieren der Mails in Quoteanfragen und Nicht-Quoteanfragen. Nach dem Sortieren sollte das erwähnte Bauteil aus dem Fließtext extrahiert werden, im SAP-ERP der Preis und die Lieferzeit bestimmt werden und zuletzt sollen alle zurückgegebenen Parameter in einer Antwortmail kombiniert werden. Der wissenschaftliche Mehrwert liegt vor allem im Sortieren und dem Evaluieren der Features.
Um den Vorgang des Sortierens genauer zu verstehen, muss das zu sortierende Objekt im Detail betrachtet werden. Dafür war es notwendig zuständige Mitarbeiter der Abteilung zu befragen, welche Parameter der E-Mail für sie mehr oder weniger ausschlaggebend sind. Grundsätzlich lassen sich Quoteanfragen auf die einfache Anfrage reduzieren. Jede technische Zusatzfrage muss von einem zusätzlichen Mitarbeiter beantwortet werden, was nur schwer durch eine automatisch generierte Mail lösbar ist. Technische Fragen treten häufig in Kombination mit Quoteanfragen auf, da vom Kunden manchmal ein Usecase beschrieben wird und nach einer Produktempfehlung zum jeweiligen Usecase gebeten wird. Ebenso als nicht-Quote deklariert, sind typische Anfragen zur Firma, Bewerbungen oder generell technischer Support. Da die Sprache dieser E-Mails im Geschäftsumfeld sehr homogener Natur ist, ist eine Intelligente Featurewahl unverzichtbar.

## 4.2 Wahl des Ansatzes

Nachdem der Usecase dargelegt wurde, muss festgelegt werden welchen Ansatz man verfolgt. In Punkt 2.1.1 sind Trainingsarten beschrieben und wann welche Art sinnvoll einzusetzen ist oder welche Modelle des Machine Learnings anzuwenden sind. Dieser Usecase entspricht einer binären Klassifizierung, da man zwischen zwei Klassen unterscheidet, welche durch Quote und nicht Quote definiert sind. Ein Regressionsansatz fällt dadurch weg, da keine Ergebnisse oder Entwicklungen anhand des Datensatzes voraus gesagt werden müssen.

Die Entscheidung zwischen einem *supervised-* oder *unsupervised-* Ansatz ist auch recht naheliegend. Man hat kein Clusteringproblem, noch muss man einen Datensatz genauer verstehen, sondern will ihn Klassifizieren, dadurch fällt die Wahl auf *supervised learning*. Um diese Art des Trainings durchführen zu können, muss nun das gesamte Trainingsdatenset per Hand sortiert und gelabled werden. Nachdem dieser einmalige manuelle Vorgang abgeschlossen worden ist, geht man zur Wahl des Classifiers und der Implementierung über.

## 4.3 Salesforce Einstein oder untrainierte Classifier

Zu Beginn der Evaluierung des Usecases startete Osram OS die Einbettung eines neuen Customer Relationship Management Systems von Salesforce. Ein Teil dieses Systems war eine Künstliche Intelligenz namens Einstein[5], welche Businessprozesse optimieren soll, wenn sie dementsprechend trainiert wird. Als möglicher alternativer Classifier, sollte Einstein in Betracht gezogen werden. Wie im Usecase schon erklärt liegt ein klassisches binäres Klassifizierungsproblem vor, das abhängig vom Featureraum gelöst werden soll. Da Einstein selbst nur mit Daten aus Salesforce arbeiten kann, waren die Möglichkeiten stark begrenzt. Darüberhinaus ist Einstein wie eine Blackbox zu betrachten, bei der nur In- und Output sichtbar sind, folglich ist eine wissenschaftliche Aufstellung schwerer zu bewerkstelligen.
Ein Ansatz mit untrainierten, leicht verständlicheren und Open Source Classifiern war daher naheliegend.

## 4.4 Architektur

Die angestrebte Struktur beinhaltet das Preprocessing der unstrukturierten E-Mails, kreieren des Featureraumes, dem Trainieren der Classifier und dem anschließenden Testen.

### 4.4.1 Tools und Libaries

Als Entwicklungsumgebung wurde *IPython Notebook*, mittlerweile *Jupyter Notebook*, mit *Anaconda* gewählt. Für das Preprocessing und die Classifier hat sich *Sklearn* [6] angeboten. Als einzelne Library wurde zum verarbeiten der *.msg* Dateien eine Open-Source Library verwendet.

[5] https://www.salesforce.com/de/products/einstein/overview/
[6] http://scikit-learn.org/stable/index.html

### 4.4.2 Datensatz

Als Quelle der vorliegenden Datensätze wurde das Archiv der Salesabteilung genutzt. Über mehrere Jahre wurden alle eingehenden Mails aller Regionen im Archiv abgespeichert.
Das Archiv beinhaltet auch E-Mails, die in anderen Sprachen verfasst worden sind. Dieser Usecase betrachtet nur E-Mails, die in deutscher Sprache erstellt worden sind. Zum Erstellen der Datensätze wurden zuerst 300 E-Mails und im zweiten Schritt zusätzliche 200 E-Mails per Hand gelabled.
Im Endeffekt resultiert dies in zwei unterschiedlich große Datensätze, die jeweils eine gleich große Anzahl an Quotemails und Nicht-Quotemails beinhalten. Dies macht beide Datensätze vergleichbarer.

### 4.4.3 Preprocessing

Als Format der eintreffenden E-Mails wird *.msg* verwendet, nachdem Outlook standardmäßig auf den Firmenrechnern installiert ist. Da die verwendete Open-Source Library den Zugriff auf den *body* und die Metadaten der E-Mail ermöglicht, ist umständliches Extrahieren dieser Daten hinfällig.

**tf-idf** Die erwähnten Hauptfeatures Inhalt- und Betreff-tfidf werden in zwei Schritten erstellt. Zu Beginn wird eine Liste der einzelnen *body* Inhalte aller E-Mails erstellt und mit einem *CountVectorizer* weiterverarbeitet. Vorab werden alle Links aus den Dokumenten gelöscht.

```
count_vect = CountVectorizer(stop_words = stopwords)
count = count_vect.fit_transform(documents)
```

Der Parameter `count` beinhaltet eine zweidimensionale Matrix, wie oft ein Wort in jedem Dokument auftritt, die oben erwähnte *term-frequency* 3.2.1. Während des Erstellens der Matrix werden alle *stopwords*, die als Liste zuvor definiert worden sind, entfernt.

```
tf_transformer = TfidfTransformer(use_idf=False).fit(count)
content_tfidf = tf_transformer.transform(content_count)
```

Der oben erstellte CountVector wird dem Objekt `TfidfTransformer` übergeben. Dieser erstellt durch die Methode `transform()` eine zwei dimensionale Matrix für alle Trainingsdokumente mit den jeweiligen *tf-idf* Werten pro Wort. Beide Schritte werden bei Betreff und Inhalt der Trainingsmails durchgeführt.

**Sekundäre Features** Um den Featureraum der sekundären Features zu erstellen, werden beim Erstellen der *body lists* zusätzliche Listen für Sendezeit oder Anhang erstellt. Beide Eigenschaften werden mit `mail.date` für die Sendezeit und einer selbsterstellten Hilfsmethode `flag_attachment(mail)`, die ein *boolean flag* setzt, wenn ein Anhang vorhanden ist, ausgelesen.
Das Parsen einer Produktnummer oder Q-Nummer wird ebenso wie das Zählen

der Fragezeichen durch Hilfsmethoden gelöst. `count_question(content` zählt wie viele Fragezeichen im Text auftreten. Damit Fragezeichen, die in Links auftreten das Ergebnis nicht verfälschen, sind diese oben schon entfernt werden. Falls im Fließtext eine Q-Nummer auftritt, setzt `check_Qnumber()` für diese Mail ein *boolean flag*. Das Ergebnis sind einzelne Matrizen, die beliebig kombiniert werden können, je nachdem mit welcher Featurekombination ein Classifier trainiert werden soll.

### 4.4.4 Classifier

Bevor die gewählten Classifier erstellt und trainiert werden können, müssen die gewünschten Features in einen gemeinsamen Featureraum transformiert werden. Um das zu bewerkstelligen nutzt man die Library [7] `scipy.sparse` und ihre Klasse `hstack`. Der entstandene Featureraum wird in einem `numpy array` gespeichert.

```
train_content_subject_time_set =
np.hstack((content_tfidf, subject_tfidf, train_send_time))
```

Hier im Beispiel werden Inhalt, Betreff und Sendezeit zu einem gemeinsamen Featureraum zusammengefügt. Dieses Konstrukt wird den Classifiern als Trainingsdatensatz zur Verfügung gestellt.
Die erwähnte Library *sklearn* ermöglicht es Classifier leicht zu erstellen.

```
knn= KNeighborsClassifier(n_neighbors=7)
knn.fit(training_dataset, labels)
```

Als Beispiel ist hier der K-Nearest-Neighbor[8] Classifier ausgewählt worden. Zuerst wird ein untrainierter Classifier erstellt, dessen Übergabeparameter *Hyperparameter* genannt werden. Diese können zur Optimierung angepasst werden, wie in diesem Fall `n_neighbors = 7` . Der Classifier hat hier die Vorgabe beim Bestimmen der Klasse eines Objekts die sieben nächsten Nachbarn zu vergleichen (2.3.3). Die Hyperparameter sind von Classifier zu Classifier unterschiedlich und haben mehr oder minder große Auswirkungen auf ihre Ergebnisse. Das Trainieren der Classifier wird auch hier von fast ausschließlich von *Sklearn* übernommen. Jeder Classifier wird durch eine `fit(training_dataset, labels )` Funktion trainiert, dabei wird der vorher erstellte Featureraum, hier training_dataset und jedes *Label*, hier labels, der einzelnen Mails übergeben.

### 4.4.5 Qualität und Testen

Nachdem das Training eines Classifiers abgeschlossen ist, muss getestet werden, um die Performance der Algorithmen in Kombination der ausgewählten Features zu dokumentieren. Der Prozess ist in zwei Stufen aufgeteilt, die für jeden Classifier wiederholt werden müssen.

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.hstack.html
[8] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

**Schritt 1: Wahl der Features und Training** Dieser Schritt ist zum großen Teil schon von den vorherigen Punkten abgedeckt. Das Training und der Kombinationsprozess sind abgedeckt, nur welche Featurekombination ist sinnvoll? Inhalt und Betreff spielen eine größere Rolle, insbesondere, wenn der Featureraum an sich betrachtet wird. Die Spaltenanzahl von Inhalt und Betreff liegen beim Inhalt im hohen vierstelligen Bereich und im Betreff im dreistelligen. Wohingegen kleinere Features oft nur eine oder zwei Spalten aufweisen. Dabei lassen sich zwei Hypothesen ableiten.

**Hypothese 1:** *Alle Ergebnisse der Classifier ohne Verwendung von Inhalt schneiden deutlich schlechter ab, als Ergebnisse mit Inhalt und Betreff.*

**Hypothese 2:** *Da Inhalt und Betreff durch ihre Größe ein mögliches Ungleichgewicht bei der Klassifizierung auslösen, haben sekundäre Features geringe bis keine Auswirkungen auf die Ergebnisse der Classifier.*

Um beide Hypothesen zu bestätigen, sind die Kombinationen wie folgt gewählt

**Schritt 2: Ergebnis Analyse** Ein Classifier ist nun mit einer bestimmten Featurekombination trainiert worden. Um die Kombinationen zu vergleichen wird auf mehrere Werte zurückgegriffen. In diesem Usecase wurde das Evaluieren auf fünf verschiedene Werte begrenzt:

- Cross-validation score
- Precision
- Recall
- Accuracy
- Specificity

**Cross-validation Score** Einen Classifer häufig mit den selben Trainingsdaten zu testen, läuft auf ein Modell hin, das *overfittet*. Dem kann man entgegenwirken, indem das Trainingsdatenset aufgespalten wird. Mit einem Teil wird der Classifier trainiert und die restlichen Daten werden zum Testen verwendet. Dieser Vorgang verkleinert die vorhanden Trainingsdatenmenge und erschwert somit den Lernprozess.
Ob die Aufspaltung der Trainingsdaten sinnvoll ist, hängt vom Zufall ab. Um dies zu vermeiden nutzt man den *k-fold approach*, bei dem alle Trainingsdaten in $k$ Samples aufgeteilt werden. Für das Trainieren des Classifiers werden $k-1$ verwendet, der Rest ist für das Testen eingeplant. Der Prozess wird so oft wiederholt, bis jedes Sample getestet worden ist. Als *Cross-validation Score* wird der Durchschnittswert aller Samples zurückgegeben. Ein hoher Score zeugt von

einem gut trainierten Classifier. Die Aussagekraft des Cross-validation Score hält sich im Vergleich zu den folgenden Parametern in Grenzen, hilft aber dem Entwickler bei der Implementierung schnell zu sehen, ob Änderungen am Datenset, den Features oder den Classifiern vorzunehmen sind.

**Precision** Ein *Precision-Wert* zeigt an wie präzise ein Classifier gearbeitet hat. Generell ist Precision ein Parameter der aus dem Information Retrieval stammt. Im oben erwähnten Beispiel soll ein Algorithmus erkennen ob ein Objekt ein Apfel ist oder nicht. Das Ergebnis des Classifiers wird in vier Klassen unterteilt:

- True Positives: Ein Objekt, welches ein Apfel ist und auch als solcher klassifiziert wurde.
- False Positives: Ein Objekt, welcher kein Apfel ist aber als solcher klassifiziert wurde.
- True Negatives: Ein Objekt, welcher kein Apfel ist und auch als solcher klassifiziert wurde.
- False Negatives: Ein Objekt, welcher ein Apfel ist aber nicht als solcher klassifiziert wurde.

Das Ergebnis lässt sich durch eine *Confusion Matrix* darstellen. Um den *Precision Score* zu erhalten benötigt man folgende Formel:

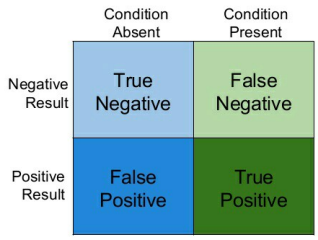


Figure 8: Beispiel einer Confusion Matrix

$$\frac{TruePositives}{TruePositives + FalsePositives} \tag{13}$$

Precision liefert einen Anhaltspunkt wie viele positiv deklarierte Objekte korrekt sortiert worden sind.

**Recall** Ähnlich wie der Precision Wert ist *Recall* eine Möglichkeit die Leistung eines Classifiers anhand der eben erklärten vier Parametern, zu messen. Im Gegensatz zur Precision zeigt Recall an, wie viele positive Objekte als solche

erkannt worden sind. Anhand des Beispiels kann durch den Recall Wert bestimmt werden, wie viele Äpfel von allen Äpfel als solche erkannt worden sind. Folgende Formel gibt an welchen Recall-Wert ein Algorithmus hat:

$$\frac{TruePositives}{TruePositives + FalseNegatives} \tag{14}$$

Folglich lässt ein hoher Recall-Wert auf eine gute Erkennungsrate schließen.

**Accuracy** *Accuracy* muss für ein besseres Verständnis klar von Precision getrennt werden. Anhand der Formel ist der Unterschied leicht zu erkennen:

$$\frac{TruePositives + TrueNegatives}{GesamtanzahlAllerObjekte} \tag{15}$$

Accuracy gibt wieder wie viele Objekte richtig zugeordnet worden sind, nicht aber wie präzise. Im Vergleich könnte ein Classifier einen schlecht balancierten Datensatz als Trainingsdatensatz zur Verfügung haben. Das resultiertin der Regel in einen schlecht trainierten Algorithmus. Angenommen dieser Algorithmus wird bei der "Apfelproblemstellung" eingesetzt und der Datensatz besteht aus 95 nicht Äpfeln und 5 Äpfeln. Der resultierende Accuracy-Wert liegt bei 95%, wobei kein Apfel als solcher erkannt worden ist. Accuracy allein ist als Performance Benchmark nicht ausreichend. In Kombination mit anderen Paramtern liefert Accuracy trotzdem wertvolle Informationen

**Specificity** Als Letzter Anhaltspunkt um die verwendeten Classifier zu evaluieren, wird ein *Specificity Score* errechnet. Dieser zeigt an wie viele negativ eingestufte Objekte korrekt als solche eingestuft wurden. Angewandt an das Beispiel, zeigt dieser Wert an wie viele Nichtäpfel richtig sortiert wurden. Folglich die Formel:

$$\frac{TrueNegatives}{TrueNegatives + FalsePositives} \tag{16}$$

Für Klassifizierungsfälle bei denen eine korrekte negative Zuordnung wichtig ist, sollte viel Wert auf diesen Parameter gelegt werden. Falls das maschinelle Sortieren der E-Mails nur unterstützden agieren soll, ist es von großen Wert zu wissen wie präzise alle *abgelehnten* oder *weggeworfenen* irrelevante E-Mails waren.

## 4.5 Ergebnisse

Im folgenden werden alle empirisch nachgewiesenen Resultate aufgezeigt. Die Auswertung ist zweigeteilt im Aspekt der Datensatzgröße. Zuerst wurde mit 300 E-Mails gearbeitet und eine Auswertung erstellt, danach mit 500.

### 4.5.1 Versuchsaufbau

**Schritt 1: Extrahieren der Emails** Zu Beginn werden alle Mailobjekte eingelesen und in einer *list* gespeichert. Dabei wird bei jedem Objekt nicht nur der Inhalt berücksichtigt, sondern Metadaten wie Sendezeit, Anhänge oder Betreff werden in separaten Listen abgespeichert. Im aus den Rohformaten der Listen Features zu generieren, wird für jedes Feature die passende Liste bereitgestellt um eine Feature Matrix zu erstellen.

**Schritt 2: Erstellen der Featurematrizen** Nachdem jedes Feature als Matrix repräsentiert ist, können diese beliebig kombiniert werden je nachdem welche Featurekombination ausgewertet werden soll. Dabei wird die Funktion *hstack()* aus der Library*numpy* verwendet.

**Schritt 3: Wahl und Training des Classifiers** Bevor ausgewählt wird welcher Classifier trainiert werden soll, wird der vorhandene Datensatz in ein Testsample und ein Trainingssample aufgespalten. Die Teilung ist die Voraussetzung um nach dem Trainieren festzustellen wie gut das Training verlaufen ist. Werte wie Precision(4.4.5) oder Recall können nur dadurch ermittelt werden.

**Schritt 4: Auswertung der Ergebnisse** Die Auswertung beinhaltet das Berechnen der Werte Precision, Accuracy, Specificity und Recall,die in4.4.5 beschrieben wurden. Das Ziel der Auswertung ist es die Featurekombination zu finden, die mit einem bestimmten Classifier die besten Werte hervorbringt. Als weiterer grafischer Anhaltspunkt für eine Auswertung von binären Klassifizierungsproblemen kann eine *ROC-Curve* herangezogen werden. Zwei Faktoren spielen bei der Entstehung der Kurve eine Rolle.
Zum einen werden verschiedene Grenzwerte der Klassifizierung verwendet. Das bedeutet es fallen mögliche Verschiebungen der Klassen ins Gewicht, ab wann ein Objekt zu welcher Klasse gehört. Der Vorteil dieser Methode liegt darin, dass nicht nur eine Fehlerrate für einen einzigen Schwellenwert berechnet wird, sondern für mehrere.
Zum anderen werden als Parameter die Anzahl der True-Positives und False-Positives verwendet. Mit ihnen wird in Abhängigkeit des gewählten Schwellenwertes berechnet welcher Anteil rechts oder links der Schwelle liegt. Die daraus resultierende Kurve ist die *ROC-Curve*.Diese wird außerdem verwendet um die Performance mehrerer Classifier mit einander zu vergleichen. Ein gut arbeitender Classifier resultiert in einer Kurve die eine möglichst große Fläche des Diagramms unter sich einnimmt. Diese Fläche wird *Area under the curve(AUC)*

genannt. In Figure 9 ist ein Beispiel einer ROC Kurve und ihrer dazu gehörenden AUC aufgeführt.

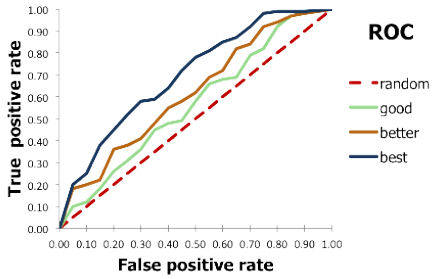


Figure 9: Ein Beispeil einer ROC-Curve. Den höchsten *AUC-Wert* liefert die dunkel blaue Kurve, somit ist im dieser Classifier im Vergleich der beste.

Um alle folgenden Auswertungen vergleichen zu können, wurden alle Classifier mit den selben Featurekombinationen trainiert. Dieser Usecase wird mit sechs verschiedenen Features durchgeführt, was zu einer hohen Anzahl an von Kombinationsmöglichkeiten führt. Im Zuge der Durchführbarkeit und dem vorab Entscheiden welche Kombinationen Sinn ergeben, wurde die Anzahl der Druchläufe auf acht verschiedene Kombinationen reduziert:

- Inhalt, Betreff, Anhang, Sendezeit Fragezeichen, Q-Nummer (Combi1)
- Inhalt, Betreff, Anhang, Sendezeit, Q-Nummer (Combi2)
- Inhalt, Betreff, Anhang, Q-Nummer (Combi4)
- Inhalt, Betreff, Sendezeit, Q-Nummer (Combi4)
- Inhalt, Betreff (Combi5)
- Inhalt (Combi6)
- Betreff (Combi7)
- Sendezeit (Combi8)

Die eingeklammerten Tags wurden zur Vereinfachung der Tabellendarstellung gewählt und fungieren als Legende.

Bei der Wahl der Kombinationen spielten mehrere Faktoren eine Rolle. Die Kombination aus allen vorhandenen Features soll zeigen, dass nicht jedes Feature zuträglich für das Ergebnis ist. Das Weglassen einzelner Features soll den Einfluss dieses einzelnen Features aufzeigen. Beim Verwenden eines einzelnen Sekundären Features dürfte ein Classifier keine guten Resultate liefern. Zuletzt werden beide Haupt-Features einzeln getestet, da diese selbst viel Informationsgehalt zur Verfügung stellen [6, pp. 1-2].

Alle verwendeten Classifier wurden mit Standard-Hyperparametern (4.4.4 trainiert. Das bedeutet kein Classifier wurde nach ersten Trainings optimiert, sodass keine beeinflussten Ergebnisse entstehen können. Der Grund dafür liegt in der Forschungsfrage, da der Fokus auf den Features und ihren Kombinationen liegt. Diese sollen den Ausschlag bei allen Resultaten geben, keine optimierten Classifier.
Zur Beschreibung der Ergebnisse werden zum einen Durchschnittswerte von Precision und Recall aufgeführt, zum anderen Besonderheiten der Ergebnisse im Vergleich zu anderen Classifiern.

### 4.5.2 Auswertung der Support Vector Machine

Der erste trainierte Classifier war eine Support Vector Machine. Im Folgenden werden die Ergebnisse aller Kombinationen mit beiden Datensätzen dargestellt.

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.77 | 0.94 | 0.84 | 0.74 |
| Combi2 | 0.77 | 0.76 | 0.78 | 0.79 |
| Combi3 | 0.69 | 0.94 | 0.76 | 0.60 |
| Combi4 | 0.79 | 0.875 | 0.83 | 0.79 |
| Combi5 | 0.77 | 0.91 | 0.83 | 0.75 |
| Combi6 | 0.78 | 0.89 | 0.83 | 0.77 |
| Combi7 | 0.70 | 0.79 | 0 .74 | 0.69 |
| Combi8 | 0.39 | 0.55 | 0.38 | 0.21 |

Table 1: Ergebnisse mit Datensatzgröße 300

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.78 | 0.825 | 0.80 | 0.77 |
| Combi2 | 0.80 | 0.85 | 0.81 | 0.78 |
| Combi3 | 0.78 | 0.75 | 0.77 | 0.79 |
| Combi4 | 0.79 | 0.825 | 0.80 | 0.78 |
| Combi5 | 0.79 | 0.83 | 0.80 | 0.78 |
| Combi6 | 0.83 | 0.81 | 0.82 | 0.83 |
| Combi7 | 0.78 | 0.76 | 0 .77 | 0.78 |
| Combi8 | 0.5 | 0.69 | 0.49 | 0.30 |

Table 2: Ergebnisse der Support Vector Machine mit Datensatzgröße 500

Nimmt man den Durchschnitt der einzelnen Spalten lässt sich folgendes erkennen:

- Die durchschnittliche Precision steigt bei einem größeren Datensatz von 0.65 auf 0.75
- Der durschnittliche Recall sinkt bei einem größeren Datensatz von 0.83 auf 0.79
- Trotz alleiniger Verwendung resultieren *Betreff* und *Inhalt* in hohe Precision und Recall Werte
- Das Weglassen oder Hinzufügen sekundärer Features hat nur geringen Einfluss auf die Werte.

### 4.5.3 Auswertung des Gaussian Naive Bayes

Die Folgenden Ergebnisse liefert ein Gaussian Naive Bayes Classifier2.3.4.

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.74 | 0.82 | 0.78 | 0.74 |
| Combi2 | 0.74 | 0.82 | 0.78 | 0.74 |
| Combi3 | 0.74 | 0.82 | 0.78 | 0.74 |
| Combi4 | 0.74 | 0.83 | 0.78 | 0.73 |
| Combi5 | 0.73 | 0.83 | 0.77 | 0.71 |
| Combi6 | 0.75 | 0.83 | 0.79 | 0.75 |
| Combi7 | 0.63 | 0.90 | 0 .70 | 0.51 |
| Combi8 | 0.65 | 0.375 | 0.61 | 0.82 |

Table 3: Ergebnisse des Gaussian Naive Bayes mit Datensatzgröße 300

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.76 | 0.71 | 0.74 | 0.78 |
| Combi2 | 0.76 | 0.71 | 0.74 | 0.78 |
| Combi3 | 0.76 | 0.71 | 0.74 | 0.78 |
| Combi4 | 0.76 | 0.71 | 0.74 | 0.78 |
| Combi5 | 0.76 | 0.71 | 0.74 | 0.77 |
| Combi6 | 0.75 | 0.79 | 0.75 | 0.73 |
| Combi7 | 0.65 | 0.79 | 0.68 | 0.57 |
| Combi8 | 0.49 | 0.65 | 0.48 | 0.31 |

Table 4: Ergebnisse des Gaussian Naive Bayes mit Datensatzgröße 500

Besonderheiten:

- Die durchschnittliche Precision sinkt beim größeren Datenset von 0.715 auf 0.711. Ebenso sinkt der durchschnittliche Recall sinkt von 0.778 auf 0.722.

- Ähnlich wie die Ergebnisse der Support Vector Machine wirken sich Datensatzgröße jeweils positiv auf Precision und negativ auf Recall aus.
- Unabhängig welche sinnvolle Kombination von Features gewählt worden ist, sind die Werte sehr ähnlich

#### 4.5.4 Auswertung des Bernoulli Naive Bayes

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.8 | 0.85 | 0.83 | 0.81 |
| Combi2 | 0.81 | 0.84 | 0.83 | 0.82 |
| Combi3 | 0.81 | 0.85 | 0.83 | 0.82 |
| Combi4 | 0.80 | 0.85 | 0.83 | 0.81 |
| Combi5 | 0.81 | 0.85 | 0.83 | 0.82 |
| Combi6 | 0.82 | 0.84 | 0.84 | 0.83 |
| Combi7 | 0.60 | 0.97 | 0.68 | 0.41 |
| Combi8 | 0.48 | 1.0 | 0.48 | 0.0 |

Table 5: Ergebnisse des Bernoulli Naive Bayes mit Datensatzgröße 300

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.77 | 0.85 | 0.79 | 0.74 |
| Combi2 | 0.76 | 0.825 | 0.79 | 0.74 |
| Combi3 | 0.77 | 0.84 | 0.79 | 0.74 |
| Combi4 | 0.76 | 0.825 | 0.78 | 0.74 |
| Combi5 | 0.76 | 0.81 | 0.78 | 0.74 |
| Combi6 | 0.77 | 0.80 | 0.78 | 0.76 |
| Combi7 | 0.65 | 0.89 | 0.71 | 0.52 |
| Combi8 | 0.49 | 0.64 | 0.48 | 0.33 |

Table 6: Ergebnisse des Bernoulli Naive Bayes mit Datensatzgröße 500

Besonderheiten:

- Precision sinkt von 0.74 auf 0.716. Recall sinkt von 0.88 auf 0.81
- Stabile Werteverteilung wie beim Gaussian Ansatz vorhanden.
- Kleinerer Datensatz liefert bessere Ergebnisse.
- Combi 8 liefert 100% Recall Wert, der entweder auf fehlerhaftem Ausführen des Algorithmus oder der Natur der Featurekombination.

#### 4.5.5 Auswertund des K-Nearest Neighbor Algorithmus

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.53 | 0.69 | 0.56 | 0.45 |
| Combi2 | 0.53 | 0.77 | 0.62 | 0.47 |
| Combi3 | 0.57 | 0.77 | 0.79 | 0.74 |
| Combi4 | 0.60 | 0.91 | 0.78 | 0.74 |
| Combi5 | 0.62 | 0.95 | 0.78 | 0.74 |
| Combi6 | 0.67 | 0.77 | 0.78 | 0.76 |
| Combi7 | 0.58 | 0.94 | 0.71 | 0.52 |
| Combi8 | 0.63 | 0.26 | 0.48 | 0.33 |

Table 7: Ergebnisse des K-Nearest Neighbor Algorithmus mit Datensatzgröße 300

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.62 | 0.65 | 0.63 | 0.60 |
| Combi2 | 0.61 | 0.68 | 0.62 | 0.56 |
| Combi3 | 0.61 | 0.68 | 0.62 | 0.60 |
| Combi4 | 0.67 | 0.825 | 0.71 | 0.62 |
| Combi5 | 0.70 | 0.85 | 0.74 | 0.62 |
| Combi6 | 0.76 | 0.625 | 0.71 | 0.80 |
| Combi7 | 0.62 | 0.89 | 0.67 | 0.46 |
| Combi8 | 0.50 | 0.75 | 0.50 | 0.25 |

Table 8: Ergebnisse des K-Nearest Neighbor Algorithmus mit Datensatzgröße 500

**Beobachtung:**

- Precision steigt von 0.59 auf 0.63 wobei Recall von 0.75 auf 0.74 sinkt.
- Sehr fluktuierende Precision- und Recall-Werte
- Punktuell sehr gute Recall-Werte bei kleinerem Datenset.

#### 4.5.6 Auswertung des Neuronalen Netzes

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.76 | 0.91 | 0.83 | 0.74 |
| Combi2 | 0.80 | 0.91 | 0.85 | 0.79 |
| Combi3 | 0.79 | 0.93 | 0.85 | 0.78 |
| Combi4 | 0.80 | 0.94 | 0.86 | 0.78 |
| Combi5 | 0.80 | 0.94 | 0.85 | 0.77 |
| Combi6 | 0.77 | 0.875 | 0.82 | 0.76 |
| Combi7 | 0.67 | 0.83 | 0.72 | 0.62 |
| Combi8 | 0.64 | 0.33 | 0.59 | 0.83 |

Table 9: Ergebnisse des Neuronalen Netzes mit Datensatzgröße 300

| Datensatzgröße | Precision | Recall | Accuracy | Specificity |
|---|---|---|---|---|
| Combi1 | 0.79 | 0.86 | 0.82 | 0.77 |
| Combi2 | 0.78 | 0.875 | 0.81 | 0.75 |
| Combi3 | 0.78 | 0.875 | 0.81 | 0.75 |
| Combi4 | 0.80 | 0.91 | 0.83 | 0.77 |
| Combi5 | 0.77 | 0.875 | 0.81 | 0.74 |
| Combi6 | 0.80 | 0.86 | 0.82 | 0.78 |
| Combi7 | 0.73 | 0.775 | 0.75 | 0.72 |
| Combi8 | 0.39 | 0.09 | 0.47 | 0.85 |

Table 10: Ergebnisse des Neuronalen Netzes mit Datensatzgröße 300

**Beobachtung:**

- Die durchschnittliche Precision sinkt von 0.75 auf 0.73. Der durchschnittliche Recallwert sinkt von 0.83 auf 0.76.
- Schlechtere Recall-Werte bei größerem Datenset, dafür kaum Veränderung bei Precision
- Extrem niedrige Recall Werte bei Combi8 drücken den Durchschnittswert signifikant, wobei bei Weglassen Spitzenwerte entstehen würden.

## 4.6 Conclusion

Zuerst muss festgelegt werden welche Werte höher priorisiert werden und welche niedriger. Letztendlich sind Precision und Recall die aussagekräfigeren Parameter, vor allem da der Usecase grob gesagt das Sortieren von E-Mails umfasst. Ein hoher Recall-Wert bedeutet, dass eine sehr hohe Zahl aller vorhandenen Quote-Mails auch als solche erkannt worden sind. Wichtig wird das, falls die Voreinteilung komplett maschinell übernommen wird, da so wenig relevante Mails wie möglich unter den Tisch fallen dürfen. In der Folge heißt das aber auch, dass viele False positives vorhanden sein könnten.
Im Gegenzug führt eine hohe Präzision zu sicheren Vorhersagen, bedeutet aber nicht dass viele relevante Mails als solche betitelt wurden. Die Einbettung und Zielsetzung eines solchen Systems ist bis jetzt noch in Planung, deshalb werden beide Parameter gleich hoch priorisiert.

### 4.6.1 Allgemeine Erkenntnisse

Zuerst wird geprüft ob beide aufgestellten Hypothesen zutreffen oder widerlegt worden sind:

**Hypothese I** Die aufgestellte Hypothese in1 behauptet, dass das Feature Inhalt einen so hohen Stellenwert inne hat, sodass das Vermeiden in diesem Usecase zu schlechten Ergebnissen führen würde. Dies wurde bestätigt, da jedes Weglassen zu deutlich schlechteren Werten bei Precision führt. Jedoch führt das Weglassen des Features bei manchen Classifiern zu sehr hohen Recall-Werten, somit ist die Hypothese nicht erfüllt, da zu Beginn entschieden worden ist beide Parameter gleich zu gewichten.

**Hypothese II** In Hypothese II wird behauptet, dass Inhalt und Betreff mit ihrer Featureraumgröße andere Features in ihrerer Wertigkeit abschwächt. Als ausschlaggebend sollen Wertabweichungen von 5-10% gesehen werden. Die Unterschiede zwischen den einzelnen Featurekombinationen mit Inhalt- und Betreffzusatz belaufen sich bei allen Classifiern, außer K-Nearest Neighbor Algorithm, auf maximal 3-4% und gelten somit als nicht ausschlaggebend. Beide Naive Bayes Ansätze weisen sogar gar keine Unterschiede zwischen den mehreren Kombinationen auf, was aber auch daran liegt dass der Bayes Ansatz alle Features unabhängig von einander betrachtet und Inhalt und Betreff Featureräume im hohen Tausenderbereich aufweisen.Im Vergleich dazu sind kleinere Features nicht ausschlaggebend. K-Nearest Neighbor wird außen vor gelassen, da seine Ergebnisse unstetig und als eher mangelhaft einzuordnen sind. Somit trifft diese Hypothese zu.

**Datensatzgröße** Desweiteren können diverse Eigenschaften im Verhalten der Featurekombinationen festgestellt werden. Im Grunde kann nicht behauptet werden, je größer das Datenset desto besser die Precision-Werte. NAch den

Egebnissen zu urteilen hängt dies vom Algorithmus ab. Es ist zu erkennen, dass oft ein größeres Datenset zu schlechteren Precision Werten führt. Dies kann an mehreren Faktoren liegen, aber durch die homogene Sprache der einzelnen E-Mails könnten Quote und Nicht-Quote Mails bei größerer Datensetgröße sehr ähnlich gesehen werden.
Oftmals leidet aber der Recall-Wert unter der Datensatzvergrößerung, was ebenso an der größer werdenden Ähnlichkeit der E-Mails liegen könnte. Außerem ist zu beobachten, dass *Inhalt* als einzelnes Feature vorallem dann bessere Ergebnisse liefert, wenn der Datensatz größer wird. Dieser Effekt ist auf den tfidf-Wert zurückzuführen, da die Wertigkeit einzelner Keywords steigt. Mehr Text mit öfter vorkommenden Keywords, bedeutet höhere Werte an wichtigen Punkten im Featureraum.

**Accuracy** Die höchsten Accuracy-Werte treten bei *Combi5-6* auf, die Inhalt und Betreff einzeln beinhalten. Folglich wurden bei dieser Kombination die meisten Mails in ihre korrekten Klassen eingeteilt. Daraus kann geschlussfolgert werden, dass sekundäre Features beim Entscheidungsprozess stören können, aber trotzdem für höhere Precision-Werte notwendig sind.

**Specificity** Im Gegensatz zum Recall wird hier auf das Korrekte negative Klassifizieren wert gelegt. Da es relevant ist ob eine E-Mail Korrekt zugeordnet wird oder ob so viele Quote-Mails als solche erkannt werden, rückt dieser Wert eher in den Hintergrund. Es treten aber Fälle auf bei denen ein hoher Recall vorhanden ist, aber eine sehr niedrige Specificity. Es wurden zwar viele Quotemails als solche erkannt, aber sehr viele False-Poisitves sind aufgetreten. Das Extrembeispiel tritt während des Bernoulli Naive Bayes Ansatz auf. Hier tritt ein perfekter Recall auf, aber ebenso eine nicht vorhandene Specificity, was bedeutet der Classifier ordnet alles einer Klasse zu. Der Wert an sich steht hier nicht stark im Fokus, hilft aber die Evaluierung abzurunden.

**Schlechtere Precision durch E-Maillänge** Im Paragraph Datensatzgröße wurde erklärt, warum bei steigender Datensatzgröße der Precision Wert leiden kann. Abgesehen von der Sprachähnlichkeit tritt ein noch größerer Faktor auf, der die Werte senkt. Die E-Maillänge und der potentiell enthaltene Dialog zwischen Sender und Absender wirken sich schlecht auf den tfidf-Wert des Features *Inhalt* aus.
Betrachtet man beispielsweise eine Quoteanfrage, welche einen Dialogbaum beinhaltet, erkennt man, dass nur eine von mehreren Teilnachrichten relevante Informationen zum Sortieren und späteren Weiterverarbeiten aufweist. Als Mensch erkennt man dieses Mailsnippet und entscheidet, während des manuellen Labelns, dass diese E-Mail als Quote einzustufen ist. Während des Trainings entsteht ein Featurevekto,r der weiter entfernt oder den meisten Vektoren anderer Quotemails unähnlich ist. Trotzdem beachtet ein Algorithmus diesen Datenpunkt als Quote, sodass das empfindliche Gefüge der Vektoren stark beeinflusst werden kann. Classifier wie Naive Bayes werden dadurch weniger beein-

flusst, K-nearest Neighbor hingegen leiden unter solch "halben" *outliern* (2.2.3). Als Lösungsmöglichkeit wurde in Betracht gezogen nur die Ursprungsmail zu extrahieren und so den Dialogbaum zu vermeiden. Dies war nicht möglich, da viele Mails weitergeleitet wurden und somit die Kernnachricht mitten im Dialog verankert sein könnte. Da jeder Classifier mit den gleichen Problemen zu kämpfen hat, ist die Vergleichbarkeit trotzdem vorhanden.
Dennoch wurde durch die Aufstellung bewiesen, dass bei alleiniger Verwendung des Features Inhalt (Combi6), alle Precisionwerte gestiegen sind. Somit wirkt sich die Mailanzahl nicht bei jeder Featurekombination negativ aus.

### 4.6.2 Vergleich der Classifer und sinnvolle Anwendung

Letztendlich stellt sich die Frage wie sich alle Classifier im Vergleich geschlagen haben. Wie in 4.5.1 bereits erklärt kann dies sehr gut durch ROC Curves beschrieben werden. Hier zum Vergleich zuerst die ROC-curve des Neuronalen Netzes:

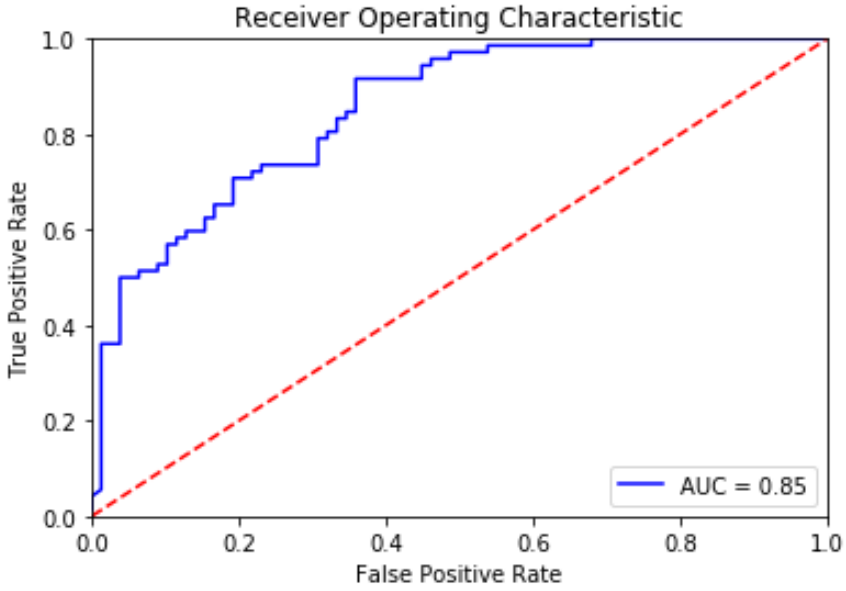


Figure 10: Abbildung der ROC-Curve des trainierten Neuronalen Netzes

Folglich die ROC-Curve des K-nearest Neighbor Classifiers:

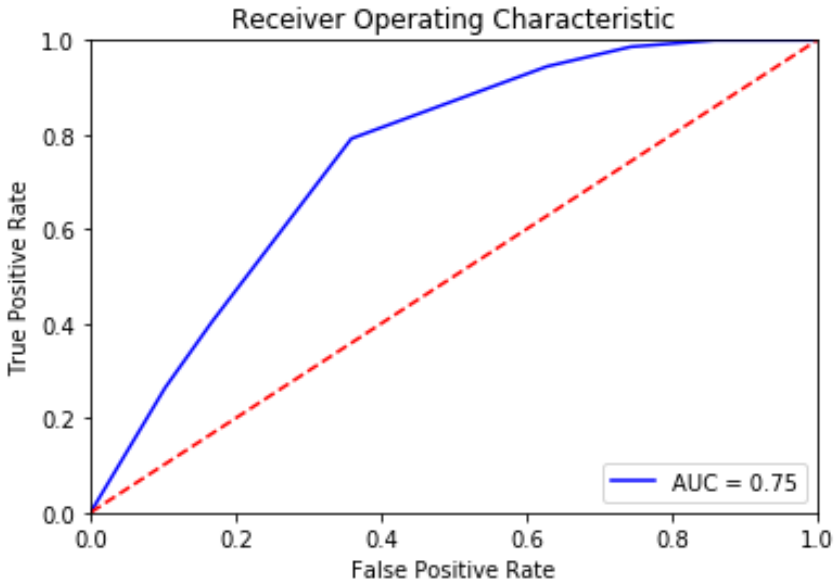


Figure 11: Abbildung der ROC-Curve des trainierten K-Nearest neighbor Classifiers

**Neuronales Netz** Zum einen liefert das Neuronale Netz die höchsten Precisionwerte und hohe Recallwerte. Zum anderen, wenn beide Kurven verglichen werden, ist gut zu erkennen, dass die Kurve des Neuronalen Netzes eine wesentlich größere AUC aufweist als die des KNN. Dies gibt zu erkennen, dass das Neuronale Netz wesentlich zuverlässiger klassifiziert. Die Werte in 4.5.6 unterstreichen dies, da die Precision wenig unter dem Datensatzwachstum leidet. Außerdem ist zu beobachten, dass fast alle Werte nah beieinander liegen. Dies zeugt zum einen von hoher Stabilität, aber auch vom geringen Einfluss der wechselnden Featurekombinationen.

**SVM** Als Spitzenreiter des Usecases hat sich der SVM Algorithmus bewährt, da mit wachsender Größe des Datensatzes ebenso die Precision steigt. Zwar leidet der Recall unter dem Wachstum, aber der Verlust ist wesentlich kleiner als der Gewinn. Die Verwendung eines SVM bei einem einfachen binären Klassifizierungsproblem ist bei diesem Usecase sehr zu empfehlen.

**Bernoulli und Gaussian Naive Bayes** Zwar schneidet der Bernoulli-Ansatz im Vergleich besser ab als der Gaussian-Ansatz, aber beide nehmen die gleiche Entwicklung, somit können beide bei der Auswertung zusammengefasst werden. Beide Naive Bayes-Ansätze leiden unter dem Wachstum des Datensatzes.

Außerdem haben Featureanpassungen kaum bis gar keine Auswirkungen auf deren Ergebnisse. Somit ist von der Verwendung in diesem Usecase abzuraten.

**K-Nearest Neighbor** Die in Abbildung 11 gezeigte ROC-Curve beschreibt die Performance des Algorithmus optimal. Eine im Vergleich sehr niedrige Steigung und Teils Precisionwerte die gegen Zufallsentscheidungen konvergieren, erübrigen jede Überlegung der Verwendung in diesem Usecase.

Zuletzt muss differenziert werden für welche Aufgaben ein Classifier gebraucht wird. Hält sich der Usecase, wie in dieser Arbeit in klaren Grenzen, kann auf diese Aufstellung zurückgegriffen werden. Da dies aber ein Feldversuch für die Anwendung von Machine Learning Ansätzen in einem modernen Unternehmen ist und Usecases oft modular und flexibel sind, sollte eher auf flexiblere Algorithmen zurückgegriffen werden. Betrachtet man alle Classifier im Punkt der Flexibilität und der Lernmöglichkeiten, ist das Neuronale Netz der Support Vector Machine vorzuziehen.

# 5 Ausblick

Nach Abschluss der Evaluierung richtet sich der Blick in Richtung Zukunft. Machine Learning im Allgemeinen etabliert sich heutzutage sehr oft in großen Unternehmen und wird zukünftig noch an Stellenwert gewinnen. Zuerst betrachten wir Osram OS und den behandelten Usecase, der repräsentativ für viele mögliche Ansätze im Unternehmen steht. Das Ziel des automatisierten Bearbeitens und Antwortens ist im Verlauf der Arbeit bereits dargelegt worden. Andere komplexere Anwendungen für Machine Learning bei Osram die nicht nur Klassifizierungsproblematiken abdecken sind zum einen Pricing und diverse Fertigungsprozesse.
Pricing selbst wird stand heute per Hand durch erfahrene Mitarbeiter betrieben. Das Pricing beinhaltet das Sammeln diverser Parameter, um einen korrekten, konkurrenzfähigen und länderabhängigen Produktpreis wiederzugeben. Dieser variiert von Kunde zu Kunde, wie teuer die Konkurenz ist oder wie viel auf Lager ist, etc. Die große Menge an nicht festgelegten Parametern, plus diverse neue Parametern die eine Maschine verarbeiten könnte, soll in Kombination einen sinnvollen und nachvollziehbaren Preis, schnell und verlässlich liefern können. Die Erfolgsaussichten eines solchen Projekts hängen stark von Datenqualität und Training ab.
Ein weiterer interessanter Anwendungsbereich lässt sich in der Fertigung finden. Diverse Regressionsmodelle können Fehlerquellen oder Ausfälle vorhersagen. Komplizierte Rezepterstellungsprozesse, die viele Testläufe als Grundlage benötigen, sollen unterstützt durch einfache Neuronale Netze beschleunigt werden. Osram OS beschreitet, wie viele andere große Unternehmen einen neuen Weg Businessprozesse und Fertigungsabläufe zu automatisieren und zu beschleunigen, indem auf neue Techologien gesetzt wird.
So groß Ambitionen im Bereich Machine Learning sind, so schnell können diese zu Nichte gemacht werden, wenn der "Rohstoff" für Machine Learning Algorithmen fehlt oder in mangelhafter Qualität vorhanden ist. Die Rede ist von Datenqualität und -quantität. Im Zuge der Arbeit konnte teilweise gezeigt werden wie viel Einfluss bestimmte Eigenschaften der Datenqualität auf das Ergebnis haben können. Projiziert man diese Einflüsse des kleinen Feldversuchs auf größere Projekte steht und fällt der Erfolg mit den Daten.

# References


[1] Martin Ester, Hans-Peter Kriegel, and Stefan Wirth. Feature based classification of protein docking sites: An algorithm for large databases and experimental results. In *German Conference on Bioinformatics*, 1996.

[2] Eibe Frank and Remco R. Bouckaert. Naive bayes for text classification with unbalanced classes. In *PKDD*, 2006.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] John J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 81 10:3088–92, 1984.

[5] Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. In *CASCON*, 2001.

[6] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*, pages 217–226. Springer, 2004.

[7] Mohammad R. Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, 2004.

[8] S. B. Kotsiantis, Dimitris Kanellopoulos, and P. E. Pintelas. Data preprocessing for supervised leaning. 2006.

[9] Charles X Ling and Victor S Sheng. Class imbalance problem. In *Encyclopedia of machine learning*, pages 171–171. Springer, 2011.

[10] Shai Shalev-Shwartz and Shai Ben-David. Understanding machine learning from theory to algorithms. 2015.

[11] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[12] Le Wu and Min-Ling Zhang. Multi-label classification with unlabeled data: An inductive approach. In *ACML*, 2013.

[13] Qin Wu, Eddie Fuller, and Cun-Quan Zhang. Text document classification and pattern recognition. In *Social Network Analysis and Mining, 2009. ASONAM'09. International Conference on Advances in*, pages 405–410. IEEE, 2009.

[14] Fernando Yepes-Calderon, Fabian Pedregosa, Bertrand Thirion, Yalin Wang, and Natasha Lepore. Automatic pathology classification using a single feature machine learning support-vector machines. In *Medical Imaging 2014: Computer-Aided Diagnosis*, volume 9035, page 903524. International Society for Optics and Photonics, 2014.