**Sentiment-Analyse**

Sentiment-Analyse (auf Deutsch oft auch als Stimmungsanalyse bezeichnet) ist ein Bereich des maschinellen Lernens und der natürlichen Sprachverarbeitung (Natural Language Processing, NLP), der sich mit der Erkennung und Klassifizierung von Emotionen und Meinungen in Texten beschäftigt. Ziel der Sentiment-Analyse ist es, festzustellen, ob ein Text positiv, negativ oder neutral ist, basierend auf der Stimmung oder dem Ton, den der Text ausdrückt.

1. **Anwendungen der Sentiment-Analyse :**

* **Kundenzufriedenheit**: Unternehmen nutzen sie, um Meinungen und Feedback von Kunden zu Produkten oder Dienstleistungen zu analysieren.
* **Social Media Monitoring**: Um Trends zu erkennen oder herauszufinden, wie Menschen über bestimmte Themen denken.
* **Marktforschung**: Um zu sehen, wie ein neues Produkt oder eine Marketingkampagne wahrgenommen wird.
* **Politik und Meinungsforschung**: Um zu verstehen, wie die Öffentlichkeit auf politische Entscheidungen oder Ereignisse reagiert.

1. **Herausforderungen der Sentiment-Analyse:**

* **Ironie und Sarkasmus**: Die Sentiment-Analyse hat oft Schwierigkeiten, ironische oder sarkastische Aussagen korrekt zu interpretieren.
* **Mehrdeutigkeit**: Manchmal sind Wörter je nach Kontext positiv oder negativ, was die Analyse erschwert.
* **Kontextabhängigkeit**: Der gleiche Satz kann in unterschiedlichen Kontexten eine andere Stimmung haben.

1. **Spezielle Herausforderungen bei Twitter**

* **Kürze der Tweets**: Twitter beschränkt Tweets auf 280 Zeichen, was bedeutet, dass oft wichtige kontextuelle Informationen fehlen.
* **Vielfalt der Ausdrucksweisen**: Die Vielzahl an Schreibweisen und Abkürzungen auf Twitter erfordert eine detaillierte Vorverarbeitung, um relevante Informationen zu extrahieren.
* **Echtzeit-Daten**: Twitter-Daten werden oft in Echtzeit analysiert, was zusätzliche Herausforderungen an die Effizienz der Vorverarbeitung stellt.

**Vorverarbeitung**

**1. Entfernen von unnötigen Zeichen**

URLs: Twitter-Nachrichten enthalten oft Links zu externen Seiten, die für die Analyse nicht relevant sind. Diese sollten entfernt werden.

Benutzernamen (@mentions): Erwähnungen anderer Benutzer bringen keine Informationen für die Sentiment-Analyse und sollten ebenfalls entfernt werden.

Hashtags (#hashtags): Hashtags können in bestimmten Fällen wichtige Informationen enthalten, aber sie müssen eventuell bereinigt oder getrennt betrachtet werden (z.B. "#HateSpeech" könnte als "Hate Speech" umgewandelt werden).

Sonderzeichen und Emojis: Sonderzeichen und Emojis können das Modell verwirren und müssen je nach Relevanz entfernt oder in Text umgewandelt werden.

**2. Tokenisierung**

Bei der Tokenisierung wird der Text in einzelne Wörter (Tokens) zerlegt. Twitter-Texte enthalten oft Slang, Abkürzungen oder unkonventionelle Schreibweisen, die korrekt gehandhabt werden müssen. Dies ist der erste Schritt, um den Text in eine verarbeitbare Form zu bringen (<https://www.kaggle.com/code/alamintokdershoukhin/tokenization-in-nlp>)

**3. Stop-Words entfernen**

Stop-Words sind häufig verwendete Wörter (z. B. „und“, „der“, „ist“), die wenig Bedeutung tragen und bei der Sentiment-Analyse oft entfernt werden, um den Fokus auf die inhaltlich relevanten Wörter zu legen. Dies reduziert die Komplexität der Daten.

**4. Lemmatisierung/Stemming**

Lemmatisierung: Dies bedeutet, Wörter auf ihre Grundform zurückzuführen, um ähnliche Wörter gleich zu behandeln (z.B. „läuft“, „laufen“ → „laufen“).

Stemming: Hier werden Wörter auf ihre Wurzel gekürzt, um Variationen zu vereinheitlichen (z.B. „running“, „runs“ → „run“). Für Hate Speech kann die Lemmatisierung oft sinnvoller sein, da die genaue Bedeutung der Wörter wichtig ist.

**5. Umgang mit Abkürzungen und Slang**

Twitter-Tweets enthalten oft Abkürzungen, Slang und umgangssprachliche Ausdrücke (z.B. „LOL“, „wtf“). Diese müssen entweder entfernt oder übersetzt werden, um sie in der Analyse berücksichtigen zu können. Es gibt auch spezielle Wörterbücher für Internet-Slang, die verwendet werden können.

**6. Umwandlung von Emojis und Emoticons**

Emojis und Emoticons können Stimmungen stark beeinflussen. Sie können in Text umgewandelt werden, um ihre Bedeutung zu berücksichtigen (z. B. „:)“ wird zu „glücklich“). Dies ist besonders wichtig, da Emojis häufig als Verstärkung oder Abschwächung von Stimmungen genutzt werden.

**7. Normalisierung von Groß- und Kleinschreibung**

Twitter-Nutzer neigen dazu, durch Großschreibung Emotionen auszudrücken (z. B. „ICH HASSE DICH!“). Hier kann es sinnvoll sein, die Großschreibung zu normalisieren (alle Buchstaben klein), es sei denn, die Großschreibung hat besondere Bedeutung für die Stimmungsbestimmung.

**8. Verarbeitung von Abkürzungen und negativen Konstruktionen**

Negationen: Negative Aussagen wie „nicht gut“ müssen erkannt werden, da sie die Stimmung komplett umdrehen können.

Abkürzungen: Twitter ist für kurze Nachrichten bekannt, daher müssen Abkürzungen wie „u“ für „you“ oder „ur“ für „your“ in ihre vollständigen Wörter umgewandelt werden.

**9. Entfernung von Duplikaten und irrelevanten Daten**

Viele Tweets sind Duplikate oder Retweets, die die Analyse verzerren können. Solche Duplikate müssen entfernt werden. Ebenso sollten irrelevante Tweets, die keine emotionale Information oder Hate Speech enthalten, herausgefiltert werden.

**10. Feature-Engineering**

Für Hate Speech können bestimmte Schlüsselwörter, die typischerweise in hasserfüllten Äußerungen verwendet werden, hervorgehoben werden. Es könnte auch sinnvoll sein, zusätzliche Features wie die Länge des Tweets, die Häufigkeit bestimmter beleidigender Wörter oder die Verwendung von Schimpfwörtern zu extrahieren, um die Erkennung zu verbessern.

**11. Vektorisierung**

Schließlich muss der Text in numerische Form gebracht werden, um ihn maschinellen Lernmodellen zugänglich zu machen. Dies kann durch:

Bag-of-Words (BoW): Zählt, wie oft jedes Wort im Text vorkommt.

TF-IDF: Bewertet die Relevanz jedes Wortes basierend darauf, wie oft es in allen Tweets vorkommt.

Word-Embeddings: Techniken wie Word2Vec oder GloVe, die semantische Beziehungen zwischen Wörtern in Vektoren darstellen.

**12. Oversampling**

Da Datensätze zur Hate-Speech-Erkennung oft mehr nicht-hasserfüllte Tweets als Hate Speech enthalten, sind Techniken wie Oversampling oder Undersampling wichtig, um die Balance wiederherzustellen und die Qualität des Datensatzes zu erhöhen.

**Modellierung**

**1. Klassische Machine-Learning-Ansätze**

Diese Methoden sind oft einfacher zu implementieren und erfordern weniger Rechenleistung als Deep-Learning-Modelle, liefern aber oft auch schnell gute Ergebnisse bei kleineren Datensätzen.

**1.1 Naive Bayes**

**Prinzip**: Naive Bayes basiert auf dem Satz von Bayes und verwendet Wahrscheinlichkeiten, um die Klasse eines Textes vorherzusagen. Es wird oft für Textklassifizierungsaufgaben wie Spam-Erkennung oder Sentiment-Analyse verwendet.

**Vorteile**:

Einfach und schnell zu trainieren.

Funktioniert gut bei kleinen und mittelgroßen Datensätzen.

**Nachteile**:

Geht davon aus, dass die Features (z. B. Wörter) unabhängig voneinander sind, was in der Praxis oft nicht der Fall ist.

Weniger leistungsfähig bei komplexen Datensätzen, wo der Kontext von Wörtern wichtig ist.

**1.2 Support Vector Machine (SVM)**

**Prinzip**: SVM sucht die optimale Trennlinie (Hyperplane) zwischen den verschiedenen Klassen im Feature-Raum, um die Daten zu klassifizieren.

**Vorteile**:

Sehr gute Leistung bei Textklassifizierungsaufgaben, insbesondere bei binären Problemen (z. B. Hate Speech vs. Non-Hate Speech).

Funktioniert gut bei hochdimensionalen Daten wie Texten.

**Nachteile**:

Hoher Rechenaufwand bei großen Datensätzen.

Komplex in der Optimierung der Hyperparameter wie dem Regularisierungsparameter C und dem Kernel.

**1.3 Logistische Regression**

**Prinzip**: Ein klassisches Modell, das eine Wahrscheinlichkeitsfunktion verwendet, um Vorhersagen zu treffen. Es ordnet einem Text die Wahrscheinlichkeit zu, in eine bestimmte Klasse zu fallen.

**Vorteile**:

Einfach zu verstehen und zu implementieren.

Liefert oft gute Ergebnisse bei binären Klassifizierungsproblemen.

**Nachteile**:

Bei komplexen Datenstrukturen, die nicht linear trennbar sind, kann die logistische Regression unterlegen sein.

**1.4 Random Forest**

**Prinzip**: Ein Ensemble-Algorithmus, der mehrere Entscheidungsbäume kombiniert. Jeder Baum wird auf einem zufälligen Teil des Datensatzes trainiert, und das Modell trifft Vorhersagen auf Basis der Mehrheitsentscheidung der Bäume.

**Vorteile**:

Robust gegenüber Überanpassung (Overfitting) bei ausreichend vielen Bäumen.

Liefert oft gute Ergebnisse bei unausgewogenen Datensätzen.

**Nachteile**:

Kann schwierig zu interpretieren sein.

Längere Trainingszeit und hoher Ressourcenverbrauch, insbesondere bei großen Datensätzen.

**2. Deep-Learning-Ansätze**

Deep Learning ist besonders nützlich bei größeren, komplexeren Datensätzen, da neuronale Netzwerke in der Lage sind, tiefere Muster in den Daten zu erkennen.

**2.1 Recurrent Neural Networks (RNNs)**

**Prinzip**: RNNs sind neuronale Netzwerke, die speziell für sequenzielle Daten wie Text entwickelt wurden. Sie berücksichtigen den zeitlichen oder sequenziellen Zusammenhang zwischen Wörtern in einem Tweet.

**Vorteile**:

Kann Text in einer sequentiellen Reihenfolge verarbeiten und Muster im Zusammenhang zwischen Wörtern erkennen.

Spezialisierte Varianten von RNN wie wie **LSTM** (Long Short-Term Memory) oder **GRU** (Gated Recurrent Units) sind besonders gut darin, längere Abhängigkeiten zwischen Wörtern zu verstehen. Beheben die schwächen von klassischen RNN

**Nachteile**:

Lange Trainingszeiten und hoher Ressourcenverbrauch.

Kann bei sehr großen Textmengen ineffizient werden.

**2.2 Convolutional Neural Networks (CNNs) für Text**

**Prinzip**: Obwohl **CNNs** ursprünglich für Bilder entwickelt wurden, können sie auch für Textklassifikationsaufgaben verwendet werden. Dabei wird der Text als eine „Sequenz von Merkmalen“ betrachtet, und CNNs lernen, wichtige Textmerkmale wie Wortgruppen zu extrahieren.

**Vorteile**:

Schnell zu trainieren im Vergleich zu RNNs.

Funktioniert gut bei kurzen Texten oder Absätzen, wie sie auf Twitter vorkommen.

**Nachteile**:

Kann den sequentiellen Charakter von Texten nicht so gut erfassen wie RNNs.

**2.3 Transformer-Modelle (z.B. BERT)**

**Prinzip**: Transformer-Modelle, insbesondere BERT (Bidirectional Encoder Representations from Transformers), haben das Feld der natürlichen Sprachverarbeitung revolutioniert. Sie analysieren den Text in beide Richtungen (bidirektional) und erfassen so den Kontext jedes Wortes im gesamten Satz.

**Vorteile**:

Sehr leistungsfähig bei der Verarbeitung von Texten, da sie den Kontext eines Wortes innerhalb des gesamten Satzes verstehen.

Pretrained-Modelle (vortrainierte Modelle) wie BERT können auf spezifische Aufgaben feinabgestimmt werden (Fine-Tuning), was die Notwendigkeit eines großen Trainingsdatensatzes reduziert. Spezialisiterte BERT-Varianten sind **RoBERTa** oder **DistilBERT**

**Nachteile**:

Hoher Rechenaufwand, insbesondere bei großen Textmengen.

Lange Trainings- und Inferenzzeiten.

**3. Ensemble-Methoden**

Eine weitere Möglichkeit, die Leistung deines Machine-Learning-Modells zu verbessern, ist die Verwendung von **Ensemble-Methoden**, die mehrere Modelle kombinieren, um die Vorhersagegenauigkeit zu erhöhen.

**3.1 Bagging (z.B. Random Forest)**

**Prinzip**: Mehrere Modelle werden auf unterschiedlichen Teilmengen des Datensatzes trainiert, und ihre Vorhersagen werden kombiniert (meist durch Mehrheitsabstimmung).

**Vorteile**: Kann die Generalisierungsfähigkeit verbessern und Überanpassung reduzieren.

**Nachteile**: Höherer Ressourcenverbrauch durch die Notwendigkeit, mehrere Modelle zu trainieren.

**3.2 Boosting (z.B. Gradient Boosting, XGBoost)**

**Prinzip**: Bei Boosting-Methoden wird jedes nachfolgende Modell auf den Fehlern des vorherigen Modells aufgebaut. Die Idee ist, schwache Modelle zu verbessern, indem Fehler korrigiert werden.

**Vorteile**: Sehr leistungsstark, da es Fehler kontinuierlich minimiert.

**Nachteile**: Kann sehr rechenintensiv sein und ist anfällig für Überanpassung, wenn es nicht richtig eingestellt wird.