Gruppenname: GodPutASmileUponYourFace

Kontaktperson: Tim Schmittmann (Tim.Schmittmann@gmx.de)

Mitglieder: Tim Schmittmann, Sebastian Riechert, Suraka Al Baradan

Problemart: Classification, Sentiment Analysis

Problembeschreibung:

Als Aufgabenstellung haben wir Sentiment Analysis auf Tweets mit Emojis als Label gewählt. In der ersten Aufgabe beschränkten wir uns hierbei auf die Binary class und Multi-Class Varianten des Problems und werden die Multi-Label Variante ggfs. in der zweiten Aufgabe abhandeln.
Unser Vorgehen in der ersten Aufgabe lässt sich dabei grob in drei Schritte unterteilen. Begonnen haben wir mit der Erstellung des Datasets und der Bereinigung des selbigen. Als nächstes haben wir diverse Vorverarbeitungsschritte wie Parsing, Labelsetzen, Kodierung der Tweets (z.B. Bag of Words), Stemming, etc. verwendet, um, die Daten für die Verwendung von “klassischen” Classifiern wir logistische Regression und Random Forest vorzubereiten. Im letzten Schritt haben wir die einzelnen Classifier mit verschiedenen Verarbeitungsschritten miteinander verglichen und unsere Ergebnisse mittels Modeltuning optimiert.

Dataset:

Tweets, die über Twitter-API ausgelesen wurden. Es wurden nur deutsche Tweets die Emojis enthalten abgerufen. Die Emojis wurden aus den Tweets-Korpus entfernt und als Labels für die Klassifikation gesetzt. Hierbei dienen die Emojis als Repräsentationen von Emotionen. Dieser Ansatz wurde gewählt, da es über diesen Parsing-Ansatz einfach war, ein großes Trainingsset zu erstellen und die Emojis ein recht breites Spektrum möglicher Emotionen abdecken. Grundsätzlich handelt es sich hierbei also um eine Multilabel-Klassifikation.

Bis dato wurden etwa 2,3 Mio. deutsche Tweets mit Emojis gesammelt, welche durch diverse Vorverarbeitungsschritte auf ca. 1,3 Mio individuelle Tweets reduziert wurden.
Alle für die Ausführung des Codes relevanten Datensätze lassen sich auch über das Google Drive oder GitHub herunterladen.

Methoden

Datensammlung:

Zum Abfragen der Tweets wurde das Skript tweet_extractor.py erstellt. Darin bedienten wir uns der Python Lib twitter-python zur Kommunikation mit der Standard Search API von Twitter. Mit dieser API ist es möglich Tweets bis zu 7 Tage in die Vergangenheit anzufragen. Da es mit der API nicht möglich ist uneingeschränkt *alle* Tweets in deutscher Sprache zu erhalten, haben wir zuerst einen Query mit den 40 häufigsten deutschen Wörtern zusammengebaut und mit diesem etwa 600 000 Tweets gesammelt (german_tweets_text_id_only_q2_597549_samples.csv).
Zusätzlich haben wir noch den API-Parameter "lang=de" zur Anfrage hinzugefügt und die Python Lib langdetect benutzt, um nach deutscher Sprache zu filtern. Ohne diese Filterungen hätten wir viele unnütze Tweets gesammelt, welche größtenteils nur aus Satzzeichen, Emojis, unlesbaren Symbolen oder falschen Kodierungen bestehen.
Im Anschluss haben wir das "emoji_statistics.py" Skript erstellt und mithilfe der "emoji" Python-Bibliothek alle 1489 Emojis aus den gesammelten Tweets extrahiert und in der Datei "emoji_cnt_used_for_twitter_api_requests.csv" gespeichert. Da die Länge der Twitter-API Querys begrenzt ist, wurden die Emojis in der "emoji_cnt_used_for_twitter_api_requests.csv" so verteilt, dass wir nun immer 45 Emojis zu einer Twitter-Anfrage zusammenfassen konnten. Außerdem hatten die einzelnen Emojis sehr unterschiedliche Häufigkeiten und durch die Verteilung innerhalb der "emoji_cnt_used_for_twitter_api_requests.csv" ließen sich so ähnlich große Dateien für die einzelnen Twitter-Anfragen erzeugen. Es entstanden also 34 Anfragen und Dateien (1489 Emojis/45 Emojis/Anfrage), die wir seitdem regelmäßig zur weiteren Tweet-Sammlung verwenden. Dazu wird zuerst die höchste bisher gesammelten tweet-id für jede Datei geparst und anschließend rückwirkend alle Tweets bis zum Erreichen der entsprechenden tweet_id gesammelt.
Neben den Tweets, Tweet-ID und Datum werden auch weitere Daten wie user_id, user_description, user_followers_count, user_friends_count, etc. gesammelt aber bis dato nicht verwendet. Eine komplette Auflistung findet sich in der "tweet_extractor.py"

Vorverarbeitung:

Folgende Arbeitsschritte werden dann nach jedem Update mit diesen 34 CSV Daten in der "csv_preparation.py" durchgeführt:

1. Zusammenführen aller einzelnen Tweet Dateien zu einer großen Datei nach dem Muster "all_emoji_tweets_DATUM.csv". Z.B. all_emoji_tweets_03_12_18.csv.
2. Ersetzen von Linebreaks (\r, \n) durch Leerzeichen, um CSV konsistenter lesen zu können. (_1_fixed_line_endings.csv)
3. Naive Duplikateentfernung von identischen tweet_full_texts, tweet_ids und doppelten Header Zeilen (_2_deduplicated.csv)
4. Sortieren der CSV nach tweet_id, implizit damit auch nach Datum (_3_sorted.csv)
5. Entfernen von sehr ähnlichen Tweets. Dazu wird ein SciKit TfidfVectorizer mit Zerlegung der Tweets in word-2gramme genutzt. Mittels Matrixmultiplikation lässt

sich dann die Kosinussimilarität der einzelnen Tweets untereinander bestimmen. Es werden anschließend zu jedem Tweet alle Tweets mit einer Kosinussimilarität von > 0.5 entfernt. Dieser Parameter sowie die Zerlegung in 2Gramme wurden empirisch bestimmt, lassen sich aber auch gut logisch nachvollziehen. (_4_removed_similar.csv)

6. Es wird die Emoji Lib benutzt, um von jedem Tweet den Tweet_Full_Texts zu parsen und sämtliche gefundenen Emojis per Komma getrennt in eine neue Spalte der CSV zu extrahieren. Dabei werden sämtliche Emojis aus den Tweet_Full_Texts entfernt aber in die neue Spalte jeder Emoji maximal 1x pro Tweet aufgenommen. Später könnte es hier noch relevant werden die Position von Emojis zu betrachten. (_5_labels_extracted.csv)
7. In “emoji_statistics.py” wurde aus den extrahierten Labels eine Statistik aufgestellt in wie vielen Tweets jeder Emoji insgesamt vorkommt und nach Häufigkeit sortiert (_emoji_cnt_.csv). Diese Statistik wurde genutzt, um von Hand bestimmte Emojis mit gleicher Bedeutung zu gruppieren (Herz-Emojis, Emojis verschiedener Hautfarben,...). Es wurden hierbei jedoch nicht alle Emojis gruppiert, sondern nur etwa die häufigsten 300 (emoji_mappings.csv)

7b Durch Formatierungsprobleme und andere Fehler sind einige Emojis aus der mappings.csv abhanden gekommen. Außerdem kann bei neuen Tweets auch ab und zu ein neues Emoji auftauchen. Diese werden hinten an die emoji_mappings.csv angefügt (emoji_mappings_2.csv)

8 Die emoji_mappings_2.csv wird nun angewandt, um die bereits extrahierten Label aus der _5_labels_extracted.csv zu gruppieren und entsprechend die Anzahl verschiedener Labels zu verringern (_6_labels_mapped.csv)

9 Einige sehr oft vorkommende Emojis haben unserer Meinung nach sehr wenig Aussagekraft (Formatierungspfeile, Copyright, ...). Wir haben diese in der emojis_to_exclude.csv zusammengefasst und nutzen diese, um die entsprechenden Labels aus _6_labels_mapped.csv zu entfernen (_7_labels_excluded.csv). Allerdings werden auch später in den Notebooks einige Mappings von Hand am geladenen Dataframe vorgenommen. Diese Schritte werden wir zur besseren Kontrolle ggfs. noch vereinen.

10 Zu statistischen und Kontrollzwecken werden zuletzt für die CSV aus 10, 12 und 13 erneut Statistiken über die Häufigkeit der einzelnen Emojis gewonnen (_emoji_cnt_5_, _emoji_cnt_6_, _emoji_cnt_7_)

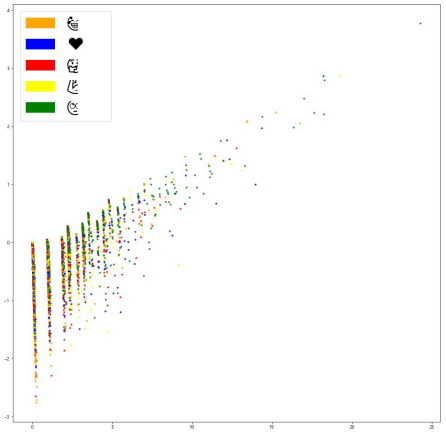
Methoden Predictive Modelling:

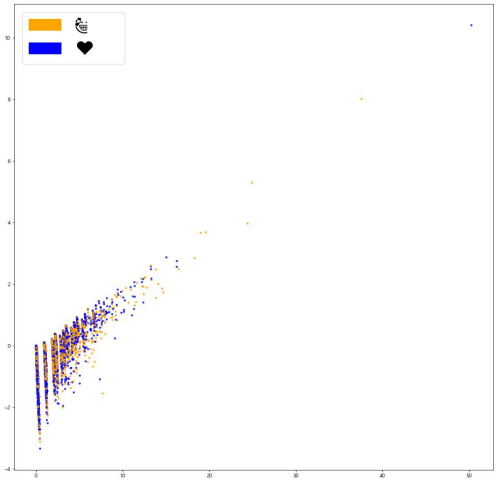
Weitere Vorverarbeitung:

Um auf den Daten überwachte Lernverfahren zur Klassifizierung trainieren zu können, muss der natürlichsprachige Text in Feature-Vektoren kodiert werden. Bevor dies geschehen kann, müssen die Daten weitere Vorverarbeitungsschritte durchgehen:

- Bereinigung von Sonderzeichen, und Formatierungsartefakten durch regular expressions und
- Tokenisierung der Tweets für Entfernung von Stopwords und Stemming

Außerdem werden alle Jahreszahlen auf einen speziellen Jahreszahl-Token gemappt, und alle restlichen Zahlen auf einen speziellen Zahl-Token gemappt.
Anschließend werden alle Tweets auf Wortebene mittels einem CountVectorizer zu Feature-Vektoren transformiert. In dieser Kodierung bleibt für jeden Tweet die Information enthalten, welche Wörter er enthält und wie oft diese Wörter jeweils in diesem Tweet vorkommen. Grammatikalische Zusammenhänge (Reihenfolge der Wörter und Satzstruktur) und semantische (zb. Ähnlichkeit von “Ich” und “mir” oder “schön” und “toll”) werden in dieser Darstellungsform nicht abgebildet. Diese “Bag of Words”-Kodierung wird für Modelle, die direkt für den Umgang mit diskreten Features gedacht sind (zb. multinomialer Naive Bayes Classifier) genutzt. Für alle anderen Classifier wurde vorerst die “Term-Frequency-Inverse-Document-Frequency”-Kodierung genutzt, welche eine normalisierte Form von “Bag of Words” darstellt.
Für beide Kodierungen wurden mittels einer PCA (genauer genommen mit einer singular-value-decomposition, da die Daten im Bag-of-Words/TF-IDF als sparse-matrix gespeichert werden und eine PCA zu langsam wäre) die Daten im zweidimensionalen Raum geplottet, um visuell die Klassen auf Separierbarkeit zu überprüfen. Diese Visualisierungen liegen im beigefügten Verzeichnis. Die folgende Abbildung zeigt die erste beiden Hauptkomponenten der PCA jeweils für ein Dataset mit 2 Klassen und für ein Dataset mit 5 Klassen. Die Klassen sehen leider in diesen beiden Kodierungen sonderlich separierbar aus.





Als nächsten wurden zwei Funktionen geschrieben, wovon die erste das Dataset so umformt, dass darauf ein 1vs1-Classifier für zwei frei wählbare Labels (Emojis) trainiert werden kann. Hierfür werden alle Einträge aus dem Dataset gedroppt, deren Label keins der beiden ausgewählten Label ist. Des Weiteren werden durch die Funktion die Klassenanteile ausgeglichen, indem von der häufigeren Klasse solange zufällige Einträge gedroppt werden, bis von beiden Klassen gleich viele Einträge vorhanden sind. Die zweite Funktion formt das Dataset so um, dass darauf eine Multi-Class-Classifications für n frei wählbare Labels (Emojis) trainiert werden kann. Die weiteren Schritte entsprechen denen der ersten Funktion.

Trainierte Classifier:

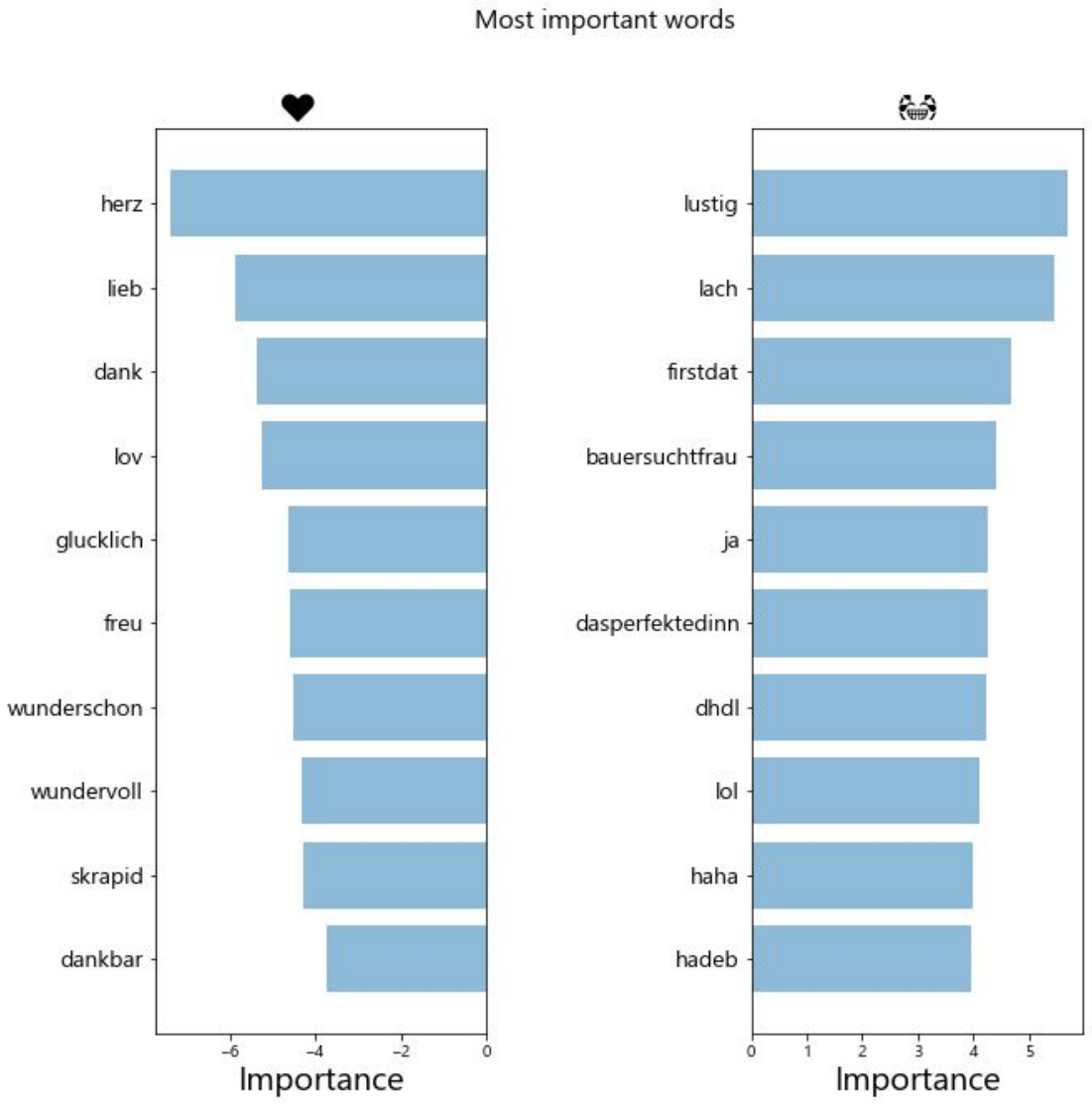
Es wurden für jeweils die 1vs1- und die multi-class-classification die folgenden drei Classifier trainiert:

- multinomial naive bayes classifier
- sgdclassifier
- random forest classifier

Für alle drei wurden Umfangreiche Hyperparametersuchen durchgeführt. Die Parametergrids sind in den beiliegenden Notebooks einsehbar. Die folgende Tabelle zeigt die Accuracies (auf Datasets mit ausgeglichenen Klassenanteilen) aller Classifier für beide Aufgaben nach der Hyperparametersuche:

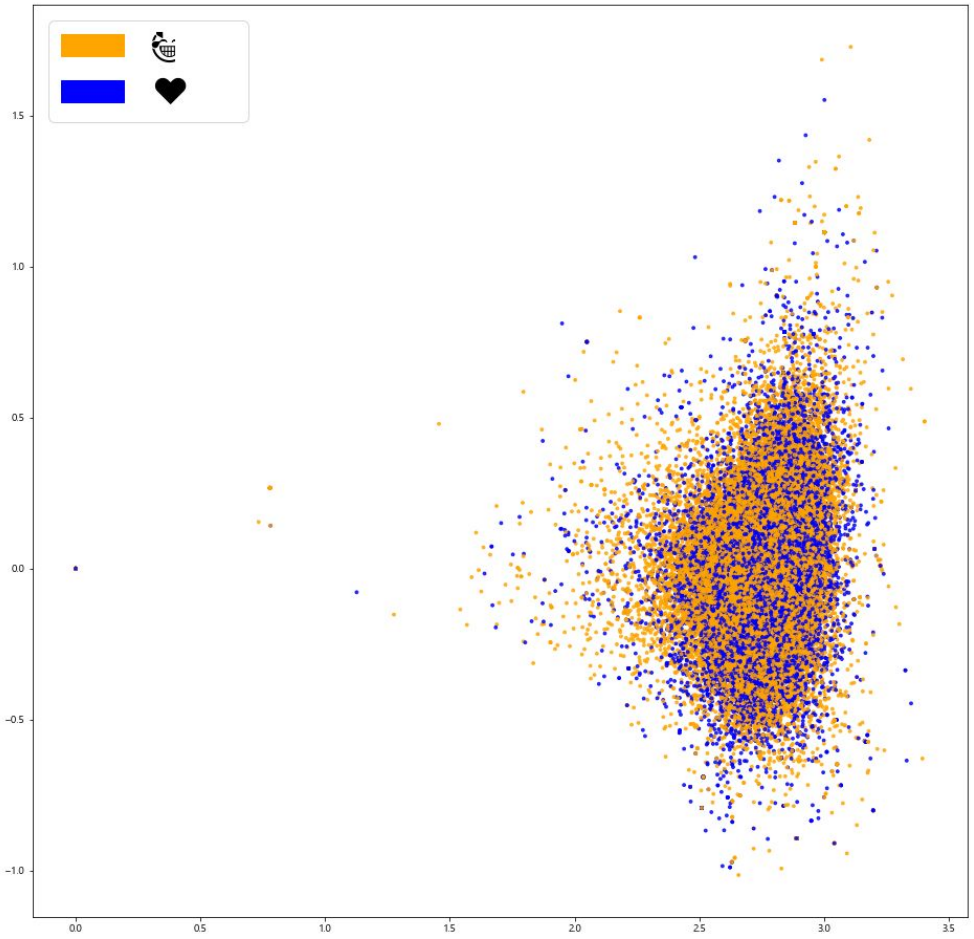
	multinomial naive bayes classifier	sgdclassifier	random forest classifier
1vs1	0.7744	0.7792	0.7501
multiclass (5 classes)	0.4736	0.4777	0.4284

Aus der Tabelle lässt sich erkennen, dass alle Classifier ziemlich ähnliche Peformance erreichen. Dies ließ uns vermuten, dass der limitierende Faktor hier nicht unbedingt die Classifier an sich sind, sondern eher die von uns gewählte Kodierung der Daten. In den Feature-Importance-Plots (Koeffizienten bei SGD-Classifier und multinomialNB-Classifier, mean-gini-decrease bei RandomForestClassifier), kann man die Wörter ablesen, deren Präsenz die Classifier als indikativ für die Labels (Emojis) gelernt haben. Die folgende Abbildung zeigt beispielsweise die Feature-Importance des SGD-Classifiers.



Wie oben bereits erwähnt, gehen in dieser Kodierung die Wortreihenfolge (=Problem1) und inhaltliche Zusammenhänge zwischen den Wörtern (=Problem2)[1] verloren.
Nach einiger Recherche entschieden wir uns dafür, für die Codierung unserer Tweets ein Word2Vec-Modell auszuprobieren. Die Fachlektüre verspricht, dass in den durch Word2Vec erstellten Embeddings ähnliche Wörter im Vektorraum nahe zueinander gruppiert werden. Damit soll Problem2 adressiert werden.
Die folgende Abbildung zeigt eine PCA für das Word2Vec-Embedding:

[1] Problem1 wäre mit Classifiern anzugehen, die besser mit der sequentiellen Struktur von natürlicher Sprache umgehen können, aber diese Ansätze werden erst später, in der zweiten Aufgabenstellung (zweiter Teil des Machine Learning 2 Projektes) umgesetzt.



Die Klassen sehen auch hier nicht wesentlich besser trennbar als zuvor aus. Folgende Tabelle zeigt die Performanz der Modelle, wenn die Word2Vec-Kodierung genutzt wird.

	multinomial naive bayes classifier	sgdclassifier	random forest classifier
1vs1	NULL	0.7334	0.6562
multiclass (5 classes)	NULL	0.4599	0.2938

*Anmerkung: der multinomial naive bayes classifier funktioniert nicht mit dieser Kodierung, er funktioniert nur bei Inputfeatures mit positiven, diskreten Feature-Ausprägungen.

Die Performanz hat sich nicht verbessert. Nächste Schritte sind das weitere Tuning des Word2Vec-Embeddings bzw. das untersuchen von anderen sinnvollen Embeddings.