Project

Fabian Otto – Matrikelnummer: 2792549
Foundations of Language Technology

4. Februar 2018

1 Datenstruktur und Problemstellung

Die Aufgabe ist es eine Sentimentanalyse für den IMDB Corpus auszuführen. Hierfür stehen jeweils 12.500 Reviews für das negative und positive Sentiment zur Verfügung. Zusätzlich werden auch noch weitere Datensätze angeboten, die ohne Labels sind. In diesem Projekt wurde diese Datenquelle jedoch nicht herangezogen. Die Reviews weisen ca. ein Länge von 500 bis 2.500 Zeichen auf und bieten damit eine ausreichende Länge, um Features zu extrahieren und die Stimmung des Verfassers zu identifizieren. Ein beispielhafter Datensatz sieht folgendermaßen aus:

> *Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school life, such as "Teachers". My 35 years in the teaching profession lead me to believe that Bromwell High's satire is much closer to reality than is "Teachers". The scramble to survive financially, the insightful students who can see right through their pathetic teachers' pomp, the pettiness of the whole situation, all remind me of the schools I knew and their students. When I saw the episode in which a student repeatedly tried to burn down the school, I immediately recalled ..... at ..... High. A classic line: INSPECTOR: I'm here to sack one of your teachers. STUDENT: Welcome to Bromwell High. I expect that many adults of my age think that Bromwell High is far fetched. What a pity that it isn't!*

Für die untenstehenden Ansätze wird immer das Vorgehen während der Dev Phase beschrieben. Gleiches gilt auch für die gezeigten Confusion Matrizen, die Evaluationskennzahlen befinden sich in der Readme.md. Zudem sind alle Train:Dev splits auf 80:20 gesetzt worden. Die Ergebnisse für die Testdaten finden sich in Codalab.

2 Naïve Bayes Ansatz

Mein erster Versuch basiert auf dem Ansatz der vorherigen Hausübung. Er extrahiert manuell Features, um im Anschluss einen Naïve Bayes einzusetzen, der das Sentiment bestimmt.

Bezüglich der Features werden dabei vor allem die Wörter selbst als auch Bigramme herangezogen. Dies trägt dazu bei, wenn Wörter verstärkt in einer der Klassen ausgeprägt sind, dass diese Klasse wahrscheinlicher als Vorhersage wird. Weiterhin werden Uni-/Bigramm Scores ermittelt. Diese basieren auf der Anzahl der häufigsten Uni-/Bigramme im Trainingscorpus. Dies stellt letztendlich eine Gewichtung der obigen beiden Features mit der Frequenz im Trainingscorpus dar. Allerdings trägt die Verwendung dieses Features nicht so stark, wie erwartet, zu den Ergebnissen bei. Vor allem da Wörter (auch nach Stopwort Bereinigung) in beiden Klassen ähnlich häufig sind (z.B.„gut“). Auch die Suche nach Schimpfwörter brachte keine wesentliche verbesserung für diesen Corpus. Allerdings konnten mit einer Accuracy von 0.891 durchaus gut Ergebnisse erzielt werden.

3 Tfldf Ansatz

Eine zweite Herangehensweise an das Problem basiert auf der Ermittlung des TfIdf Vectors für die jeweiligen Reviews. TfIdf verfolgt die Idee, dass für jedes Wort w die Frequenz innerhalb des Dokuments d bzw., in dem vorliegenden Problem, der Review. Zudem wird durch die Berechnung der Dokumentenfrequenz, d.h. wie viele Dokumente d im Corpus w beinhalten, verhindert, dass insgesamt sehr frequente Wörter (the, a, etc.) zu stark gewichtet werden. Die konkrete Ausprägung, die im Rahmen von Sklearn hier eingesetzt wird, setzt sich wie folgt zusammen:

$$w_{t,d} = (1 + \log(tf_{t,d})) \cdot \log\left(\frac{1+N}{1+df_t} + 1\right) \tag{1}$$

Zudem wird eine Kosinus-Normalisierung vorgenommen. Von Sklearn wird hierfür ein Pipeline-Ansatz zur Verfügung gestellt, der sowohl einen CountVector erstellt als auch im Anschluss die TfIdf Scores berechnet. Am Ende wird ein Classifier bereitgestellt, der anhand der TfIdf Vectoren die Klassen bestimmt. Hierbei sind zum Vergleich Multinominal Naïve Bayes (MNB) und Stochastic Gradient Descent (SGD) verwendet worden. Letzteres ist laut der Webseite von Sklearn bereits oft erfolgreich in NLP Problemen eingesetzt worden.[1] Dabei ergeben sich folgende Ergebnisse:

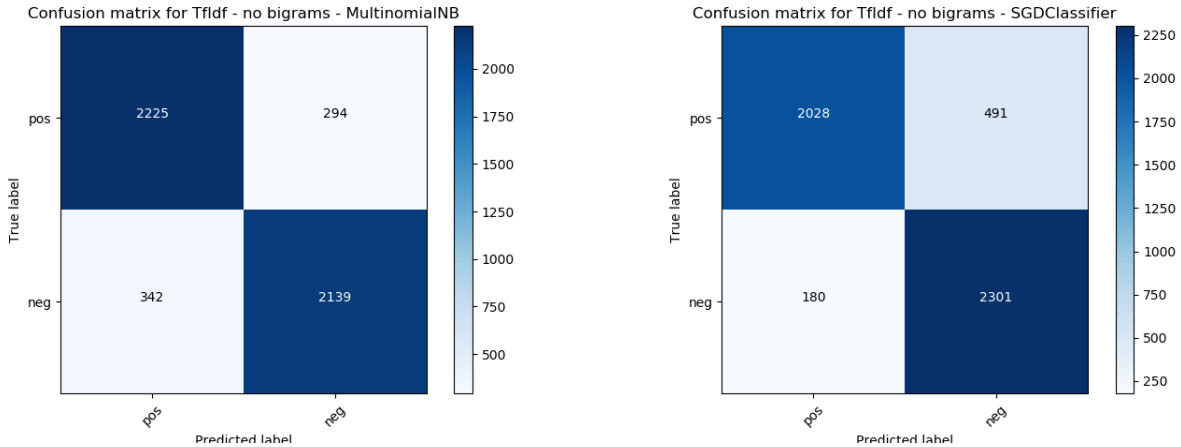

Abbildung 1: Confusion Matrix für Multinomial Naïve Bayes und Stochastic Gradient Descent.

[1] `http://scikit-learn.org/stable/modules/sgd.html`

Zusätzlich besteht die Möglichkeit n-gramm-Ranges einzusetzen. Dabei ergibt sich folgenden Ergebnis für $n \in \{1, \ldots, 3\}$

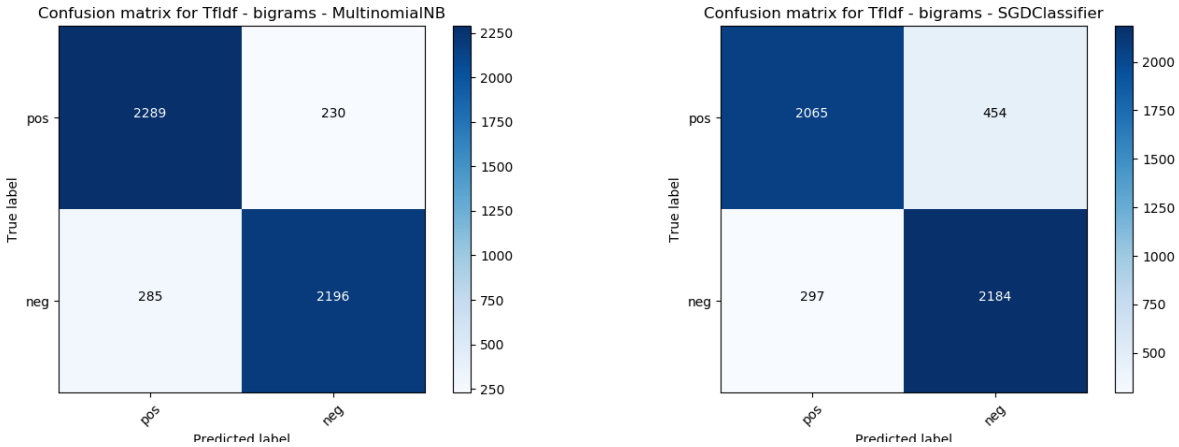

Abbildung 2: Confusion Matrix für Multinomial Naïve Bayes und Stochastic Gradient Descent mit n-grammen.

Ohne n-Gramme zeigt MNB eine bessere Bestimmung von positiven Reviews auf, wohingegen SGD für negative Reviews besser zu funktionieren scheint. Beim Einsatz von n-Grammen zeigen die Ergebnisse ein leichte Verbesserung mit MNB im Vergleich zu SGD . Wobei insgesamt bei MNB mit n-Grammen die besten Resultate zu finden sind mit einem F-Score von knapp unter 90%.

4 Paragraph Word Vectors

Der dritte Ansatz basiert auf Paragraph Word Vectoren, die mit einem Deep Learning Ansatz erstellt wurden. Hierzu wurde die „doc2vec“ Bibliothek von gensim eingesetzt.[2] Nach der Erstellung der Vectorrepräsentationen des Textes, werden diese als Trainingsdaten für eine Logistik Regression [3] bzw. eine SVM genutzt, um die Klassen der Testdaten vorherzusagen. Hierbei sei erwähnt, dass für die Erstellung der Vectoren auch die Testdaten beim Training des Neuronalen Netzes eingesetzt wurden. Dies wurde so realisiert, da es sich hierbei lediglich um die Ermittlung der Features und nicht um die Vorhersage selbst handelt[4]. Bei der Anzahl der Epochs für das Training des Deep Nets zeigten sich keine signifikanten Unterschiede in der Genauigkeit der Logistik Regression oder der SVM. Das derzeitige Modell wurde mit 200 und 500 Epochs trainiert, dabei konnte allerdings keine Verbesserung erreicht werden.

[2] `https://radimrehurek.com/gensim/index.html`

[3] Diese wurde auch im Beispiel von gensim eingesetzt `https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-IMDB.ipynb` In diesem Projekt wurde jedoch die Sklearn Implementierung verwendet `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`.

[4] Darauf hatte ich auch Prof. Dr. Christian Meyer angesprochen.

Dabei ergaben sich folgender Ergebnisse:

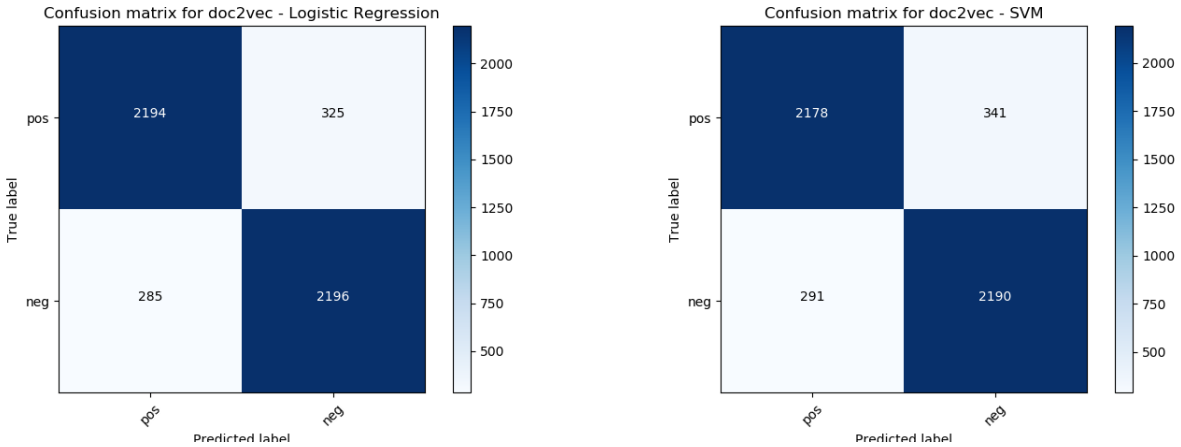

Abbildung 3: Confusion Matrix für Doc2Vec mit Logistik Regression und SVM.

5 Fazit

Sowohl die Verwendung von doc2vec als auch von TfIdf bringt ähnlich gute Resultate für die Developmentdaten. Eine Veränderung der Parameter im Rahmen einer GridSearch[5], insbesondere für SVM und Logistik Regression hat hierbei keinen signifikaten Verbesserungen bewirkt. Insgesamt zeigt sich, wie erwartet, die Verwendung von n-Grammen als nützlich. Das Doc2Vec Modell i.V.m. Logistik Regression bzw. SVM zeigte, schlechtere Ergebnisse als erwartet und konnte von einer „simplen“ TfIdf Kalkulaton geschlagen werden. Bei der finalen Evaluation auf den Test Daten sind alle Ansätze ähnlich gut und im Bereich von 87% Accuracy.

6 Laufzeit

Dies is abhängig was alles ausgeführt wird. Insgesamt habe ich nie alles ausgeführt, sollte aber ca. 3h+ benötigen. Nur der Ansatz basieren auf der Hausübung (NaiveBayesApproach) sollte ca. 45 min. brauchen. TfIdf hingegen nur ca. 1-2 min, und doc2Vec benötigt nur ca. 1h, wenn das Modell neu erstellt werden muss.

7 Mentions

Das Paper hat einige gute Ansätze, aber ich hatte leider nicht genügend Zeit mehr auszuprobieren: `https://pdfs.semanticscholar.org/c521/80a8fe1acc99b4bf3cf3e11d3c8a38e2c7ff.pdf`
Und nette Confusion Matrizen finden sich hier: `http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model`

[5] `http://scikit-learn.org/stable/modules/grid_search.html`