Klassifizierung von Filmkritiken

Katsiaryna Mlinaric, Alain Chavannes, Thomas Steiner

18. Januar 2017

Inhalt

Problemstellung

Wie lassen sich Texte mittels eines Algorithmus klassifizieren?

- 2'000 englischsprachige Filmkritiken aus der IMDB
- positiv / negativ

Baseline

- empirisch
- nicht spezifisch trainiert

Naive Bayes

- statistisch
- arbeitet mit Wahrscheinlichkeiten
- spezifisch trainiert

VADER Sentiment Analysis

- Lexikon und Regel-basiertes Sentiment Analysis Tool
- Abgestimmt auf Social Media Texte
- Lexikon mit 7'500 Wörtern
- Ratings von -4.0 bis $+4.0$

Weka Library

- Open Source Suite
- Data-Mining und maschinelles Lernen
- Unterstützt eine grosse Bandbreite an Verfahren
- Weka Explorer vereinfacht das Austesten von Parametern

Baseline Methode: Counting Sentiments

VADER Sentiment Lexicon

- Ratings
- zusätzlich: Polarität -1 für Rating < 0, $+1$ für Rating > 0

Methode: für jedes Dokument Akkumulation

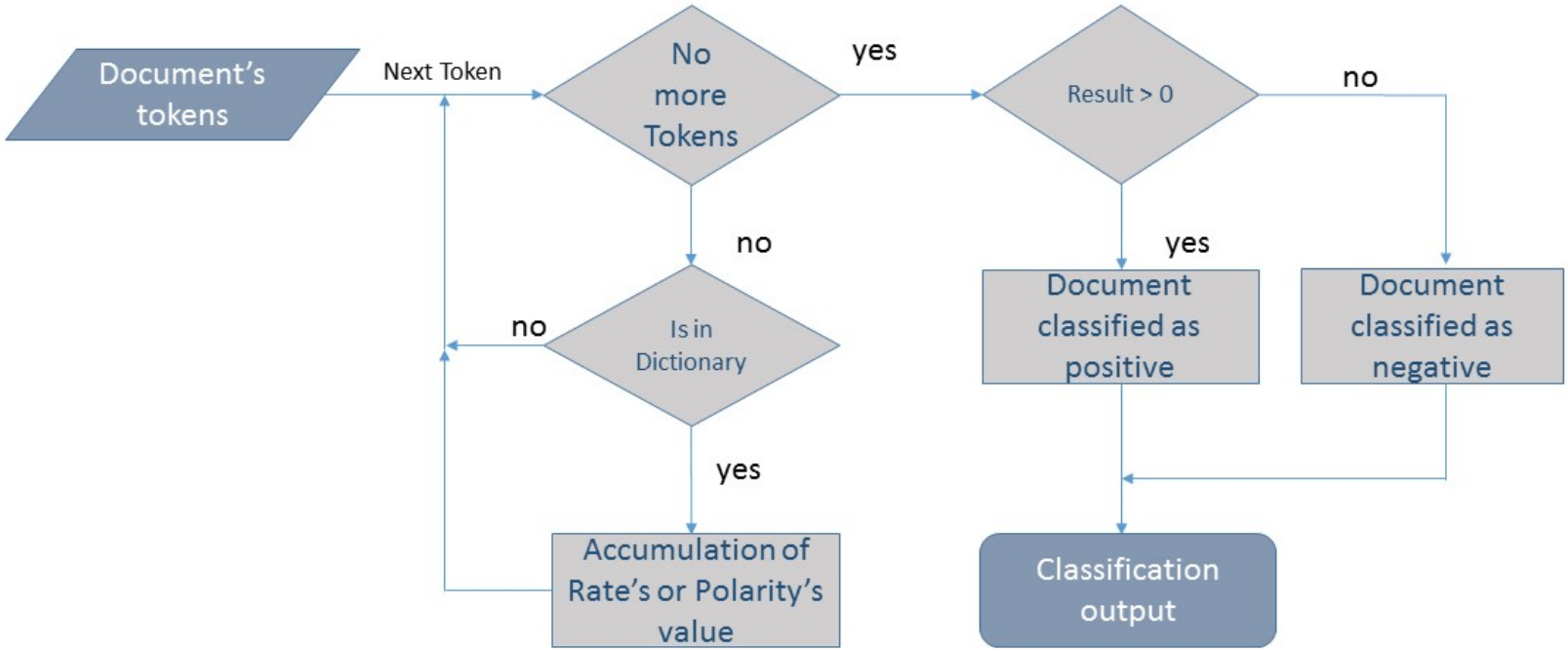
- der Ratings aller Wörter
- der Polaritäten aller Wörter

Resultat: aufsummierte Werte der Ratings bzw. Polaritäten
Klassifikation:

positiv Wenn Summe der Ratings (oder Polaritäten) > 0

negativ Wenn Summe der Ratings (oder Polaritäten) < 0

Algorithmus-Schema



Algorithmus-Schema von Dokumentenklassifikation

Baseline Klasse

Implementation

- Implementiert das Interface *Classifier*
- Konstruktor nimmt als Argumente die Art der Klassifikation und das Dictionary:

```
public Baseline(String choice, Dictionary dict){
    this.choice=choice;
    this.dict=dict;
}
```

- Beinhaltet folgende 4 Methoden:

Baseline Klasse

- *tokenizeDoc(Document doc)*
  splittet das Dokument in einzelne Token (mit Länge > 1) auf; gibt Liste der Tokens zurück

```
public ArrayList<String> tokenizeDoc(Document doc){
  tokensList.clear();
  String docText=doc.getText();
  String [] words= docText.split("\\s");
  tokensList.addAll(Arrays.asList(words));

  List<String> minwordsList=tokensList.stream().filter(
    w->w.length()>1).collect(Collectors.toList());
  ArrayList<String> retwords=new ArrayList<String>(
    minwordsList);

  return retwords;
}
```

Baseline

- *analyzeDocRate(String docName, ArrayList<String> tokenList)*
  berechnet das Rating des Dokuments; gibt den akkumulierten Wert des Ratings zurück
- *analyzeDocPolarity(String docName, ArrayList<String> tokenList)*
  berechnet die Polarität des Dokuments; gibt den akkumulierten Wert der Polarität zurück
- *classify(Document doc)*
  gibt die Klassifizierung des Dokuments zurück (*pos* oder *neg*)

Naive Bayes

- Nimmt Unabhängigkeit der einzelnen Features an
- Aufgrund dieser Annahme lässt sich das Bayes'sche Theorem anwenden

Klassifikation von Dokumenten unter Einsatz des Bayes'schen Theorems

- $P(c|d_i) = \frac{P(c) \cdot P(d_i|c)}{P(d_i)}$
- $P(c) = \frac{1}{\#Klassen}$
- $P(d_i|c) = P(f_1|c) \cdot P(f_2|c) \cdot ... \cdot P(f_j|c)$
- $P(d_i) = \frac{1}{\#Dokumente}$

Datenaufbereitung

- Zerlegen der Texte mittels Tokenizer
- Ermitteln der aussagekräftigsten Wörter für beide Kategorien (verkürzt die Laufzeit)

Baseline

Accuracy des Baseline-Algorithmus

- mit Polaritäten: 58.6 %
- mit Ratings: 61.8 %

Einige der aussagekräftigsten Features

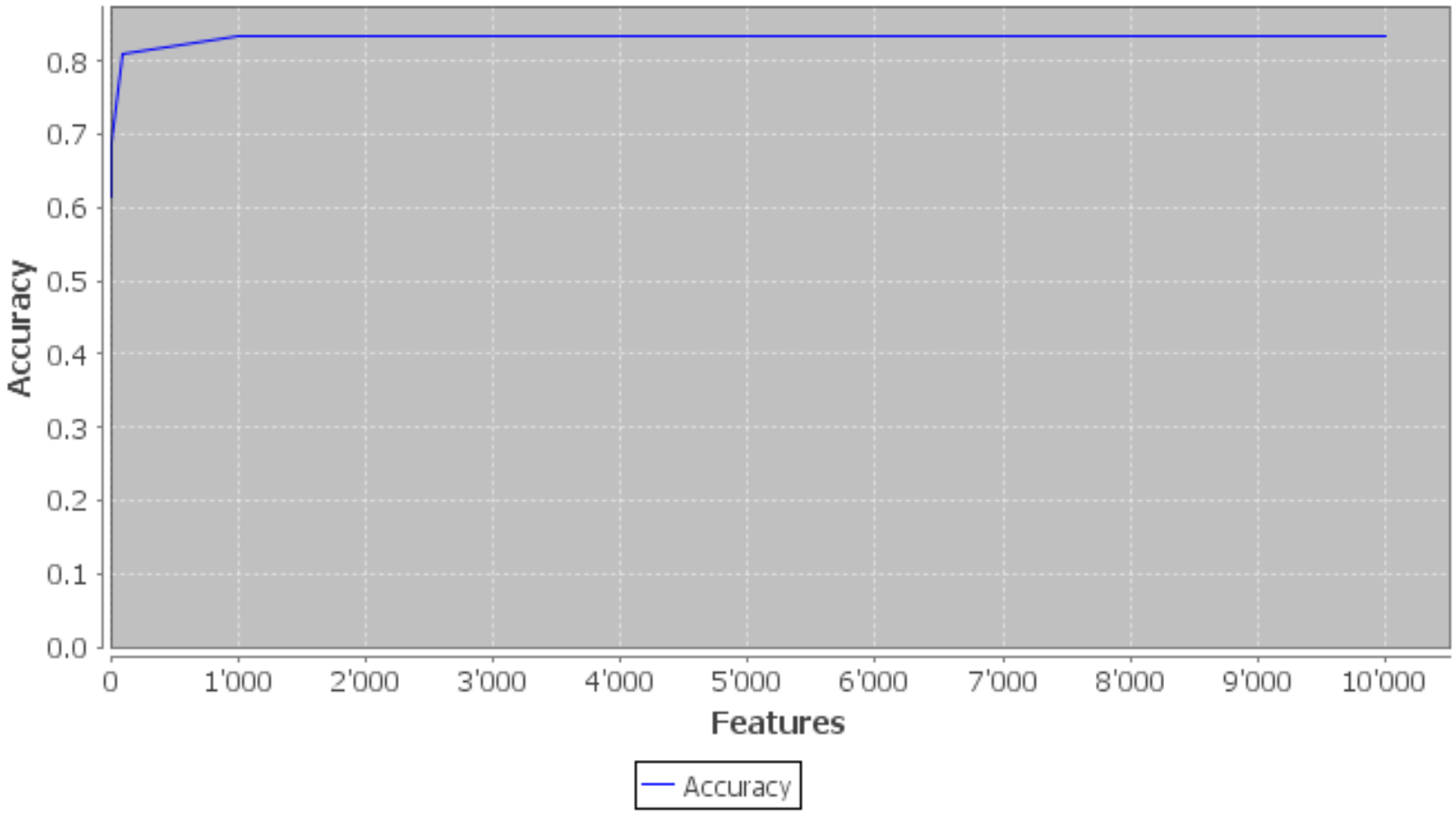
Positiv

- outstanding
- excellent
- perfect

Negativ

- bad
- worst
- stupid

Optimale Featureanzahl



Naive Bayes

- "Naiver" Ansatz funktioniert erstaunlich gut: Ohne Einsatz von speziellen Filtern werden rund 80% aller Dokumente korrekt kategorisiert
- Stemming hat in unserem Fall keine positive Wirkung gezeigt
- Wahl des passenden(deren) Tokenizers hat das Ergebnis leicht verbessert
  - Whitespace: 80.5%
  - 3-gram: 83.45%