Künstliche Intelligenz - PVL Dokumentation

Nils Generlich (s85449)

15.06.2024

1 Einführung

Im Folgenden wird das aus der PVL-Aufgabe entstandene Programm dokumentiert und dessen Benutzung erläutert. Das Programm setzt einen Bernoulli Naive Bayes Klassifikator mit Laplace-Glättung um, welcher mittels Training (durch eingelesene Trainingsdaten) Text klassifiziert und diese der Klasse A oder der Klasse B zuordnet. Die Kompilierung und das Starten, sowie das Hinzufügen von Trainingsdaten ist folgend erläutert. Außerdem wird die Klasse NaiveBayesClassifier mit ihren Methoden und Klassenvariablen erläutert. Als Letztes sind die Ergebnisse, welche das Programm bringt, bewertet/geprüft worden. Das als PVL entstandene Programm ist in der Programmiersprache Java geschrieben.

2 Benutzung des Programms

Der Naive Bayes Klassifikator ist als Klasse in der Java-Datei "NaiveBayesClassifier.java" im src-Verzeichnis gespeichert. Er lässt sich mittels zwei übergebenen txt-Dateien trainieren und klassifiziert mit Berücksichtigung der Laplace-Glättung Dokumente (je eine Zeile) einer anderen txt-Datei zu einer der Klassen A oder B. Im Folgenden ist die Kompilierung und Ausführung des Programmes, von der Kommandozeile (aus dem Basisverzeichnis KI-PVL_s85449) aus, beschrieben.

2.1 Kompilieren

Das Java-Programm kann nach Installation des Java Development Kit (JDK) über folgenden Weg kompiliert werden:

- Ausführen von Befehl `javac -d bin src/*.java` (ausgeführt im Verzeichnis "KI-PVL_s85449")

2.2 Ausführen

- Wechsel in das Binary-Verzeichnis: `cd bin`
- Ausführen des Programms mit:
  `java NaiveBayesClassifier <traindataA> <traindataB> <data_to_be_classified> [includeWordsNotInDataToBeClassified]`
  - **traindataA**: relative Pfadangabe (ausgehend vom Verzeichnis KI-PVL_s85449) zu einem txt-Dokument, welches alle Trainingsdokumente (je eine Zeile) für Klasse A beinhaltet (z.B.: data/train/A.txt)
  - **traindataB**: relative Pfadangabe (ausgehend vom Verzeichnis KI-PVL_s85449) zu einem txt-Dokument, welches alle Trainingsdokumente (je eine Zeile) für Klasse B beinhaltet (z.B.: data/train/B.txt)
  - **data_to_be_classified**: relative Pfadangabe (ausgehend vom Verzeichnis KI-PVL_s85449) zu einem txt-Dokument, welches die zu klassifizierenden Dokumente enthält (z.B.: data/test.txt)
  - **includeWordsNotInDataToBeClassified**: *(optional)* Boolean-Wert, welcher bestimmt, ob alle Wörter des Trainingsvokabulars, auch wenn sie nicht in dem zu klassifizierendem Dokument vorkommen, in der zu berechnenden Wahrscheinlichkeit (als Gegenwahrscheinlichkeiten) berücksichtigt werden sollen. (**true** = Berücksichtigung von Wörtern, die nicht im zu klassifizierenden Dokument vorkommen, **false** = nur Berücksichtigung von Wörtern, welche im klassifizierenden Dokument vorkommen). Falls kein 4. Kommandozeilenparameter eingegeben wird, gilt hierfür standardmäßig false.

  Wichtig!: Die Pfadangabe der Dateien muss relativ zum Basisverzeichnis, in welchem sich die Unterverzeichnisse src, bin, und data befinden, (KI-PVL_s85449) geschehen
- Aufruf des Programms mit gegebenen Beispieldatensätzen ist damit möglich mit Berücksichtigung aller Wörter des Vokabulars bei der Klassifikation:
  `java NaiveBayesClassifier data/train/A.txt data/train/B.txt data/test.txt true`

  ...und lediglich nur mit der Berücksichtigung der Wörter, die im zu klassifizierenden Dokument vorkommen:
  `java NaiveBayesClassifier data/train/A.txt data/train/B.txt data/test.txt`

2.3 Aufbau der Dateien mit Trainings- oder zu klassifizierenden Daten

2.3.1 Trainingsdaten

- Die beiden Dateien für Klasse A und B sollten eine txt-Datei sein. Ansonsten sind auch weitere Textdateiformate möglich, solange die Datei im Textformat vorliegt und durch Zeilen getrennte Zeichenfolgen beinhaltet.
- Jede Datei steht für eine Klasse und enthält pro Zeile ein Trainingsdatum/-dokument

- Zeilen werden durch einen Zeilenumbruch getrennt
- Die Wörter sind sowohl in Groß- und Kleinschreibung erfassbar. Erfasste Wörter sind nicht case sensitive, wodurch es egal ist ob sie groß- oder kleingeschrieben werden. (z.B. gilt "Hallo" als das gleiche Wort wie "hallo")
- Es werden nur Wörter erfasst, welche aus Buchstaben, Ziffern oder Untersrichen bestehen. Gleichsam werden alle weiteren Zeichen, wie bspw. Satzzeichen oder Leerzeichen als Trennzeichen der Wörter verwendet.

2.3.2 Klassifizierende Daten

- Die Datei sollte eine txt-Datei sein. Ansonsten sind auch weitere Textdateiformate möglich, solange die Datei im Textformat vorliegt und durch Zeilen getrennte Zeichenfolgen beinhaltet.
- Diese Datei beinhaltet beliebige Zeilen an zu klassifizierenden Dokumenten
- Dabei wird jede Zeile als einzelnes Dokument klassifiziert
- Zeilen/Dokumente werden durch einen Zeilenumbruch getrennt
- Die Wörter sind sowohl in Groß- und Kleinschreibung erfassbar. Erfasste Wörter sind nicht case sensitive, wodurch es egal ist ob sie groß- oder kleingeschrieben werden. (z.B. gilt "Hallo" als das gleiche Wort wie "hallo")
- Es werden nur Wörter erfasst, welche aus Buchstaben, Ziffern oder Untersrichen bestehen. Gleichsam werden alle weiteren Zeichen, wie bspw. Satzzeichen oder Leerzeichen als Trennzeichen der Wörter verwendet.

2.3.3 Beispieldaten

Im Ordner *data* befinden sich Beispieldateien mit Beispieldaten aus Aufgabe 25 des KI-Praktikums.

- Trainingsdaten: *data/train/A.txt* oder *data/train/B.txt*
- Dokumente, die klassifiziert werden sollen: *data/test.txt*

3 Verzeichnisstruktur von Basisverzeichnis "KI-PVL_s85449"

- `docs`: Hier befindet sich diese Dokumentation der PVL, in welcher ich auf die Benutzung des Programms und die Klasse inkl. ihrer Methoden (Architektur) eingehe.
- `src`: Hier befindet sich das gesamte Programm in Form einer Java-Datei, welche als Bernoulli Naive Bayes Klassifikator mit Laplace-Glättung fungiert. Mit übergebenenen Trainingsdaten können weitere Dokumente durch dieses Programm klassifiziert werden.
- `bin`: Enthält die kompilierte Klassendatei (.class) der Java-Quellcodedatei (.java), die sich im src-Ordner befindet. Diese ist mittels `java Klassendateiname [Kommandozeilenparameter]` ausführbar.
- `data`: Enthält bereits Beispielsdaten für Trainingsdaten der Klassen A und B (im Unterverzeichnis train) und eine Datei mit 2 zu klassifizierenden Dokumenten/Zeilen. Diese Beispieldaten entstammen der Aufgabe 25 aus dem KI-Prakiktum.

4 Architektur

Die Klasse NaiveBayesClassifier implementiert einen Naive Bayes Klassifikator für Textklassifikation, der Dokumente basierend auf Trainingsdaten in eine von zwei Klassen kategorisiert.
Zur Funktionalität des Programms zählt:

- Trainieren des Klassifikators mit Dokumenten aus zwei Klassen (Klasse A und Klasse B).
- Klassifizieren einzelner Dokumente oder mehrerer Dokumente in einer Datei.
- Option, Wörter, die nicht in den Testdaten vorkommen, während der Klassifikation einzubeziehen.

4.1 Klassenvariablen

- **Set<String> vocabulary:** Stellt die Menge aller Wörter dar, die in den Trainingsdaten von Klasse A oder B vorkommen. Dabei kommt jedes Wort nur einmal vor, was die Schnittstelle Set garantiert.
- **Map<String, Integer> wordCountA/wordCountB:** Stellt ein Mapping mit einem Wort als Schlüssel und der dazugehörigen Integer-Anzahl, wie oft das Wort in Klasse A/ B vorkommt, dar.
- **int totalLinesA/totalLinesB:** Zählt die Anzahl der Dokumente in Klasse A/ B, welche wichtig für die Berechnung der Wahrscheinlichkeiten zur Klassifizierung ist.

Als Implementierung der Interfaces Set und Map werden jeweils Hash-Tabellen genutzt, aufgrund ihrer schnellen Laufzeit für elementare Zugriffsoperationen.

4.2 Methoden

4.2.1 Konstruktor:

```
public NaiveBayesClassifier()
```

Initialisiert den Klassifikator mit leerem Vokabular und Wortzählungen.

4.2.2 train:

```
public void train(String filePathA, String filePathB) throws IOException
```

Trainiert den Naive Bayes Klassifikator mit den bereitgestellten Trainingsdateien für Klasse A und Klasse B.

Parameter:

- filePathA: Der relative Dateipfad, vom bin-Verzeichnis aus, zu den Trainingsdaten für Klasse A.
- filePathB: Der relative Dateipfad, vom bin-Verzeichnis aus, zu den Trainingsdaten für Klasse B.

4.2.3 trainClass:

```
private boolean trainClass(String filePath, Map<String, Integer> wordCount,
boolean classA)
```

Methode zum Trainieren des Klassifikators für eine spezifische Klasse.

Parameter:

- filePath: Der relative Dateipfad, vom bin-Verzeichnis aus, zu den Trainingsdaten für die Klasse.
- wordCount: Die Map, um die Wortzählungen für die Klasse zu speichern.
- classA: Wahrheitswert, der angibt ob es sich um Klasse A (true) oder Klasse B (false) handelt

Rückgabewert:

- boolean: true, wenn das Training erfolgreich ist, ansonsten false.

4.2.4 classify_all:

```
public void classify_all(String filePath, boolean includeWordsNotInTestData)
throws IOException
```

Klassifiziert alle Dokumente in der angegebenen Datei und gibt das Klassifizierungsergebnis für jedes Dokument aus.

Parameter:

- filePath: Der Dateipfad zu den Testdaten, die ein oder mehrere Dokumente enthalten.
- includeWordsNotInTestData: Eine boolean-Flag, die angibt, ob Wörter, die nicht in den zu klassifizierenden Dokumenten vorkommen, während der Klassifikation einbezogen werden sollen.

4.2.5 classify:

```
public String classify(String line, boolean includeWordsNotInTestData, int i)
throws IOException
```

Klassifiziert ein einzelnes Dokument basierend auf den trainierten Daten.

Parameter:

- line: Der Text des zu klassifizierenden Dokuments.
- includeWordsNotInTestData: Eine boolean-Flag, die angibt, ob Wörter, die nicht in dem zu klassifizierenden Dokument vorkommen, während der Klassifikation einbezogen werden sollen.
- i : Nummer des durch die Funktion gerade zu klassifizierenden Dokuments in der angegebenen Datei (hat v.A. Relevantnis für die Ausgabe)

Rückgabewert:

- String: Das Klassifizierungsergebnis inkl. Wahrscheinlichkeitswerte.

4.2.6 main:

```
public static void main(String[] args) throws IOException
```

Hauptmethode, um den Klassifikator von der Kommandozeile auszuführen. Sie prüft zusätzlich die Angabe von Kommandozeilenparametern und ob die gegebenen Dateipfade tatsächlich existieren.

Parameter: args: Kommandozeilenargumente. Erwartete Argumente sind:

- args[0]: Der relative Dateipfad (vom Basisverzeichnis) zu den Trainingsdaten für Klasse A.
- args[1]: Der relative Dateipfad (vom Basisverzeichnis) zu den Trainingsdaten für Klasse B.
- args[2]: Der relative Dateipfad (vom Basisverzeichnis) zu der Textdatei, die (ein oder mehrere) Dokumente/Zeilen enthält, die klassifiziert werden sollen.
- args[3] (optional): Ein boolean-Flag (true oder false), die angibt, ob Wörter, die nicht in den zu klassifizierenden Dokument vorkommen, während der Klassifikation einbezogen werden sollen.

5 Ergebnisse

5.1 Bernoulli Naiver Bayes Klassifikator

Ein Bernoulli Naiver Bayes Klassifikator arbeitet mit binären Merkmalen, d.h. er betrachtet nur das Vorhandensein oder Nichtvorhandensein von Wörtern in einem Dokument. Mein Programm implementiert diesen, indem es für jedes Dokument/jeder Zeile der Textdatei mehrfach vorkommende Wörter nur jeweils einmal zählt.

5.2 Laplace-Glättung

Die Laplace-Glättung wird verwendet, um zu verhindern, dass Wahrscheinlichkeiten auf Null gesetzt werden, wenn ein Wort in den Trainingsdaten fehlt. Dies wird durch Hinzufügen von 1 zur Zähleranzahl (Anzahl der Vorkommen eines Wortes + 1) und Hinzufügen von 2 zum Nenner (Gesamtzahl der Dokumente in der Klasse + 2) erreicht. Im Code erfolgt das durch folgende Zeilen:

Listing 1: Java: Laplace-Glättung

```
probA *= (countA + 1.0) / (totalLinesA + 2.0);
probB *= (countB + 1.0) / (totalLinesB + 2.0);
```

5.3 Anwendung der Klassifizierung von Beispiel von Praktikumsaufgabe 25

Mit den 6 gegebenen Trainingsdokumenten, wovon je 3 einer Klasse (+ oder -) angehören, sollen zwei weitere Dokumente klassifiziert werden. Die Berechnung dazu taten wir im Praktikum händisch. Mit dem Vergleich der Werte, welche per Hand berechnet wurden, und der Werte, welche vom entwickelten Programm generiert wurden, konnte die Ausgabe des Programms auf Richtigkeit getestet werden.

5.3.1 Händisches Klassifizieren

Hierbei konzentrierte ich mich bei dem händischen Berechnen auf das Errechnen der Wahrscheinlichkeit, ohne die Wörter im Vokabular, welche nicht in den zu klassifizierenden Dokumenten vorkommen, einzubeziehen.

Dokument 7:

$\hat{p}(\text{plain boring dialogues uninteresting plot}|+) = \hat{p}(+) \cdot \hat{p}(boring|+) \cdot \hat{p}(plot|+) \cdot \hat{p}(plain|+) \cdot \hat{p}(uninteresting|+) = \frac{1}{2} \cdot \frac{1}{5} \cdot \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{1}{5} = 0.0024$

$\hat{p}(\text{plain boring dialogues uninteresting plot}|-) = \hat{p}(-) \cdot \hat{p}(boring|-) \cdot \hat{p}(plot|-) \cdot \hat{p}(plain|-) \cdot \hat{p}(uninteresting|-) = \frac{1}{2} \cdot \frac{2}{5} \cdot \frac{4}{5} \cdot \frac{2}{5} \cdot \frac{2}{5} = 0.0256$

$\Rightarrow$ Dokument 7 wird zur Klasse "-" klassifiziert, da $0.0256 > 0.0024$

Dokument 8:

$\hat{p}(\text{great scenes, interesting story}|+) = \hat{p}(+) \cdot \hat{p}(great|+) \cdot \hat{p}(interesting|+) = \frac{1}{2} \cdot \frac{3}{5} \cdot \frac{2}{5} = 0.12$

$\hat{p}(\text{great scenes, interesting story}|-) = \hat{p}(-) \cdot \hat{p}(great|-) \cdot \hat{p}(interesting|-) = \frac{1}{2} \cdot \frac{1}{5} \cdot \frac{1}{5} = 0.02$

$\Rightarrow$ Dokument 8 wird zur Klasse "+" klassifiziert, da $0.12 > 0.02$

5.3.2 Klassifizieren mittels Bernoulli Naive Bayes Klassifizierer-Programm

Indem ich die auf Seite 3 beschriebene Benutzung des Programms befolgte und die Beispieldaten (welche der Aufgabe 25 des Praktikums entsprechen, siehe Seite 4) nutzte, konnte ich mit dem Aufruf "`java NaiveBayesClassifier data/train/A.txt data/train/B.txt data/test.txt`" beide Dokumente klassifizieren lassen. Dabei berücksichtigte ich, wie bereits beim händischen Errechnen angemerkt, nur die Wörter, die im zu klassifizierenden Dokument vorkommen und verzichtete so auf einen 4. Kommandozeilenparameter. Das Programm lieferte folgende Ausgabe:

```
Documents to be classified from file: ../data/test.txt
-------------------------------------------------
Classification of document 1:
prob(doc1|A) = 0.002400000000000002
prob(doc1|B) = 0.025600000000000008
The document 1 was classified as class B with the probability: prob(doc1|B) = 0.025600000000000008
-------------------------------------------------
Classification of document 2:
prob(doc2|A) = 0.12
prob(doc2|B) = 0.020000000000000004
The document 2 was classified as class A with the probability: prob(doc2|A) = 0.12
```

Diese Werte entsprechen gerundet den händisch errechneten Werten.