Maschinelle Sprachverarbeitung

Übung

Aufgabe 3: SPAM-Klassifikation

Mario Sänger

Aufgabe

- Rund 50% des weltweiten Email-Verkehrs ist Spam*
  - Spam- und Phishing-Mails stellen eines der größten Sicherheitsrisiken für Unternehmen dar
- Implementiert einen Klassifizierer zur Erkennung von Spam-Emails
- Trainingsdatensatz mit realen Spam-/Ham-Emails
  - Daten enthalten Email-Text + Header-Informationen (Absender, Empfänger, Content-Type, ...)
  - https://hu.berlin/ue_maschsp1718_ue3_email_korpus

* https://newsroom.web.de/2016/02/08/spam-aufkommen-in-deutschland/

Email-Korpus

- Je ein Verzeichnis mit Spam- und Ham-Emails
  - Je Email ist in einer separaten Datei gespeichert
  - Verteilung: 2158 Spam- und 2522 Ham-Emails

```
From exmh-users-admin@redhat.com  Thu Aug 22 14:44:07 2002
Return-Path: exmh-users-admin@example.com
....
Sender: exmh-users-admin@example.com
Reply-To: exmh-users@example.com
...
Date: Thu, 22 Aug 2002 23:36:32 +1000
Subject: Re: Insert signature

Hi! Is there a command to insert the signature using a
combination of keys and not> to have sent the mail to insert it
....
```

Gestaltungsmöglichkeiten

- Auswahl der Features
  - Nur Text? Nur Betreff? Email-Inhalt?
  - Absender- und Empfänger (-domäne)?
  - MIME-Type? Empfangszeitpunkt?
- Repräsentation der Features
  - Texte: Binär? TF*IDF? Word Embeddings?
- Feature-Selection:
  - Manuelle Auswahl, Information Gain, Keine Reduktion?

Gestaltungsmöglichkeiten

- Auswahl der Klassifikationsmethode
  - Support Vector Machine (SVM), Naive Bayes, K-Nearest-Neighbor, Random Forests, Künstliche Neuronale Netze, ...
- Verwendung beliebiger Bibliotheken
  - Java: Stanford Core NLP, OpenNLP, Lingpipe, Weka, LibSVM, Mallet, Deep-Learning-4J, ...
  - Python: NLTK, Scikit-Learn, Gensim, Tensorflow, ...
- Einsatz spezieller Spam-Software (z.B. SpamAssassin) ist nicht erlaubt!

Aufgabendetails

- Entwickelt ein Programm in Java, Scala oder Python zur Spam-Erkennung
  - Unterstützung eines Trainings- und eines Klassifikationsmodus

- 1. Trainingsmodus
  - Lesen der Trainingsbeispiele und Lernen eines Modells
  - Speichern des Modells in „*model_name*“ im Ausführungsverzeichnis

```
java –jar uebung3-groupX.jar train model_name \
      ham_directory spam_directory
```

```
python uebung3-groupX.py train model_name \
      ham_directory spam_directory
```

Aufgabendetails

- 2. Klassifikationsmodus
  - Lesen des Modells aus „*model_name*" aus dem aktuellen Verzeichnis
  - Lesen und Klassifikation aller Dateien in „*email_directory*"
  - Ergebnisausgabe: „DATEINAME\tSPAM/NOSPAM" in „*result_file*"

```
java –jar uebung3-groupX.jar classify model_name \
        email_directory result_file

python uebung3-groupX.py classify model_name \
        email_directory result_file
```

Aufgabendetails

- Java: Abgabe eines ausführbaren Jar-Archivs
  - Archiv muss alle Abhängigkeiten / Bibliotheken enthalten

Kompilieren:

```
javac –cp ./lib/* -d ./out/ ./src/de/huberlin/wbi/*.java
```

Bibliotheken entpacken:

```
cp ./lib/*.jar ./out/

cd out

tar xf *.jar && rm *.jar
```

Ausführbares Jar-Archiv erstellen:

```
jar cvfm uebung3.jar ./MANIFEST.FM -C out .
```

Aufgabendetails

- Weitere Details und Erläuterungen:

https://dzone.com/articles/java-8-how-to-create-executable-fatjar-without-ide

- Erstellung wird auch von Build-Tools unterstützt
  - Maven Assembly Plugin:
    https://maven.apache.org/plugins/maven-assembly-plugin/usage.html
  - Gradle Shadow Plugin:
    https://github.com/johnrengelman/shadow

Aufgabendetails

- Python: Angabe der verwendeten Version und Auslistung aller Dependencies
  - Bereitstellung einer *requirements.txt* mit allen Abhängigkeiten
  - https://pip.pypa.io/en/stable/reference/pip_install/#requirements-file-format

requirements.txt

```
sklearn

gensim

tensorflow == <version>
```

Evaluation

- 10-fach Kreuzvalidierung (in Entwicklungsphase)
  - Beachten der Verteilung (Ham-vs-Spam) bei der Bildung der disjunkten Teilmengen!
  - Ergebnisse müssen besser als „A-Priori-Klassifikation" sein

- Finale Evaluation erfolgt auf einem separatem Test-datensatz
  - Gleiche Spam-/Ham-Verteilung wie Trainingsdatensatz
  - Zusätzliche Evaluation mit anderen Verteilungen (nicht Wettbewerbsrelevant!)

Abgabe

- Abgabe eines ZIP-Archivs uebung3-groupX.zip
  - Ausführbares Programm und dessen Quellcode
  - Python: Auflistung der Dependencies (*requirements.txt*)
  - Ergebnisse der 10-fach Kreuzvalidierung (inkl. Durchschnitt, Standardabweichung und Varianz)

- Testet Euer Programm vor der Abgabe!

- Abgabe bis spätestens 07.01.2018, 23:59 Uhr über: https://hu.berlin/ue_maschsp1718_ue3

Wettbewerb

- Die Lösung mit der höchsten Genauigkeit gewinnt
  - Genauigkeit = Anteil korrekt klassifizierter Emails
- Geschwindigkeit ist diesmal irrelevant!
- Die drei besten Teams erhalten 5/3/1 Punkte