Übung 03: Machine Learning

Programmiertechniken in der Computerlinguistik II, FS 16

Abgabetermin: 21.04.2016

Hinweise zur Abgabe

- Bitte gib jedes Python-Programm in einer eigenen Datei ab, die die Dateiendung `.py` hat und *ausführbaren* Python-Code enthält.
- Geize nicht mit Kommentaren direkt im Programm-Code, wo Erläuterungen angebracht sind. Umfangreiche Erklärungen werden hingegen besser in einer separaten README-Datei mitgeliefert (vorzugsweise Plain-Text oder PDF).
- Um das Hochladen der Abgabe auf OLAT zu erleichtern, kannst du die Dateien mit `zip` oder `tar` (oder einem anderen verbreiteten Format) archivieren/komprimieren.
- Bitte gib nicht dein vollständiges Korpus ab, sondern nur ein Subset welches deine gewählte Dateistruktur ausreichend repräsentiert.

1 Ein Korpus erstellen

In dieser Übung werden wir eine Sammlung von Emails[1], welche im Zuge einer Untersuchung veröffentlicht wurden, als Grundlage für verschiedene Zwecke benutzen.

Um diese Emails einfach zu verarbeiten soll daher ein Korpus erstellt werden, welches dann unter Python mit NLTK einfach verwendbar sein sollte.

Lade die Daten aus OLAT (`Materials/Additional_Material/Enron`) herunter und überlege dir wie du die Korpora gestalten würdest:

- Sollen getrennte Korpora für Spam und Nicht-Spam erstellt werden?
- Welche Metadaten sollten erfasst werden?
- Wie sollten die Daten in den Programmen gespeichert werden?
- Wie sollten die Daten auf der Festplatte gespeichert werden?
- Wie könnte Spam markiert werden?

Implementiere deine Überlegungen als Korpus, zum Beispiel mit Hilfe des `PlaintextCorpusReader` aus der NLTK-Korpus-Kollektion.

Beachte, dass die folgenden Aufgaben auf deinen erstellten Korpora aufbauen, von daher ist es ratsam, sich zu überlegen welche Merkmale nützlich sein könnten.

Abzugeben ist ein Skript, welches ein oder mehrere Korpora erstellt und sich in den folgenden Aufgaben einfach als Modul aufrufen lässt.

[1] `http://www.aueb.gr/users/ion/data/enron-spam/`

2 n-Gramm-Modelle

Schreibe eine Sammlung von Funktionen, welche das Korpus aus Aufgabe 1 in n-Gramme umwandelt und auf verschiedene Eigenschaften testet:

a) Ein Test, ob ein bestimmtes n-Gramm im Korpus vorkommt.

b) Eine Funktion die eine Liste mit allen n-Grammen ausgibt, welche mit dem gewählten (n-1)-Gramm beginnen.

c) Eine Funktion zur Berechnung der unbedingten Wahrscheinlichkeit eines n-Gramms.

d) Eine Funktion zur Berechnung der bedingten Wahrscheinlichkeit eines n-Gramms.

e) Eine Funktion welche testet, ob ein n-Gramm eine Kollokation darstellt; die Kriterien sind selber zu wählen und im Programmcode zu begründen.

Da NLTK schon viele Werkzeuge zum Umgang mit n-Grammen enthält, sollen die Funktionen deiner Sammlung auf keine externe Module zugreifen, ausser natürlich auf deine Implementation von Aufgabe 1.

Abzugeben ist ein Skript, welches die Korpora aus Aufgabe 1 einliest, die Texte in frei wählbare[2] n-Gramme umwandelt und die entsprechenden Abfragefunktionen zur Verfügung stellt. Das Skript sollte sich in den folgenden Aufgaben ohne Probleme als Modul aufrufen lassen.

3 Klassifikation: Spam, Spam, Spam!

Nun ist es an der Zeit, die Daten einzusetzen um Emails auf Spam (unerwünschte) und Ham (erwünschte Emails) zu klassifizieren.

Verfahre analog zu Sektion 1.3 von Kapitel 6 im NLTK-Buch, wobei hier die Struktur deines Korpus (oder deiner Korpora) den Ablauf stark beeinflusst.

a) Verwende am Anfang noch keine Merkmale (*Features*), sondern das untenstehende Code-Schnipsel. Die resultierende Präzision des Klassifikators stellt die Baseline dar, welche dir bei der Weiterentwicklung deiner Merkmale hilft. Auf welcher Basis entscheidet der Klassifikator, ob ein Dokument Spam oder Ham ist?

b) Erstelle nun eigene Merkmale wie in Sektion 1.3 beschrieben steht. Experimentiere mit verschiedenen Merkmalen sowie deren Kombinationen, vergleiche die Präzision deines Klassifikators mit der Baseline und notiere deine Feststellungen direkt im Programmcode beim jeweiligen Merkmal.

Listing 1: Leeres Merkmal zur Erstellung der Baseline

```
# empty dictionary to comply with the requirements of the classifier
def document_features(document, words):
    return {}
```

Hinweise

- Die Klassifikation kann eine Weile in Anspruch nehmen, daher ist es empfehlenswert, ein kleines Subset zu verwenden und das Programm erst am Schluss auf die Gesamtmenge der Daten auszuweiten.

[2] der Fokus liegt auf Uni-, Bi- und Trigrammen

- Falls `[Errno 24] Too many open files` auftritt, liegt dies am ausführenden System und kann relativ einfach behoben[3] werden.
- Die Aufgaben basieren aufeinander, daher kann es von Vorteil sein, die bisherigen Programme für den weiteren Gebrauch wieder anzupassen.

Reflexion/Feedback

a) Fasse deine Erkenntnisse und Lernfortschritte in ein paar Sätzen zusammen.

b) Wie viel Zeit hast du in diese Übungen investiert?

[3] `http://swampdragon.net/blog/too-many-open-files/`