Data Mining

Versuch 4 Dokument Klassifikation / Spam Filter

von
Armin Schwarz (as219)
Florian Tatzel (ft020)
Marc Walter (mw136)

Das git Repository mit den Quellcodes findet sich unter https://github.com/floriant/DataMining
Alle Dateien zu diesem Versuch befinden sich im Unterordner Versuch_4.

2 Durchführung

2.1 Implementierung

Der Naive Bayes Classifier ist in der Datei **Versuch_4/Program/docclass.py** implementiert.

2.2 Test

Eine Implementierung eines einfachen Spam-Classifiers nach dem Skript aus der Vorlesung Einführung Künstliche Intelligenz findet sich in der Datei **Versuch_4/Program/docclass.py**.

2.3 Klassifikation von RSS Newsfeeds

Nach dem Training sollen alle Nachrichten aus der Liste test vom Naive Bayes Classifier automatisch klassifiziert werden.
Insgesamt wurden 130 Trainings-Samples für die Kategorie tech und 103 für die Kategorie nontech gelernt. Anschließend wurden 64 Test-Samples kategorisiert.

Beurteilen und diskutieren Sie die Rate korrekter Zuweisungen:
Der Klassifizieren hat folgende Einteilungen vorgenommen:
59 Test-Samples wurden als tech kategorisiert. Dies entspricht etwa 92,2% der gesamten Test-Menge. Davon waren 40 wirklich tech-feeds, was einer Rate von 67,8% entspricht, und 19 nontech-feeds.
Nur 5 Test-Samples wurden als nontech kategorisiert, was ca 7,8% entspricht. Von diesen war eines eigentlich ein tech-feed und vier, also 80%, wurden richtig einsortiert.

Insgesamt gesehen wurden also:
40 von 41 tech-feeds richtig kategorisiert (~97,6%)
4 von 23 nontech-feeds richtig kategorisiert (17,4%)

Hieran lässt sich erkennen, dass die relativ ähnlichen Tech-Feeds eigentlich ganz gut eingeordnet werden konnten, die teilweise sehr unterschiedlichen nontech-feeds jedoch nicht. Wenn jetzt die nontech-feeds genauer aufgeteilt werden würden (zum Beispiel in Politik, Sport, Finanzen, Unterhaltung), müsste ein genaueres Ergebnis erreicht werden können.

Der Quellcode für die Klassifizierung von RSS Newsfeeds wurde in der Datei **Versuch_4/Program/parseTechFeed.py** gespeichert.

Zusätzlich wurde das Speichern und Laden des gelernten Klassifizierers implementiert. Gespeichert wird der Klassifizierer in der Datei **Versuch_4/Programm/classifier.serialized**, ein Beispiel für das Einlesen des Klassifizierers und das Einteilen von Testdaten in der Datei **Versuch_4/Program/parseTechFeedWithSerializedClassifier.py** gespeichert.

3 Fragen zum Versuch

1. Was wird mit Evidenz bezeichnet und warum muss diese für die Klassifikation nicht berücksichtigt werden?

Die Evidenz beschreibt eine Wahrscheinlichkeit des Auftritts einer bestimmten Klasse. Der Naive Bayes Classifier geht jedoch davon aus, dass alle Variablen unabhängig voneinander sind und somit einander nicht bedingen. Der Wert für die Evidenz wäre folglich immer sehr ähnlich und kann darum ignoriert werden.

2. Wann würden Sie in der Formel für die gewichtete Wahrscheinlichkeit den Wert von initprob kleiner, wann größer als 0.5 wählen? (Falls Sie die Möglichkeit haben diesen Wert für jedes Feature und jede Kategorie individuell zu konfigurieren)

Initprob könnte daran angepasst werden, wie oft ein bestimmtes Wort in dem vorliegenden Text vor kommt. Falls das Wort nur einmal vorkommt würde initprob klein gewählt werden, und falls jedes zehnte Wort in einem langen Text genau dieses ist, würde initprob größer als 0.5 gewählt werden.
Eine weitere Idee wäre, dass sofern bekannt ist wie wahrscheinlich eine Kategorie auftritt, initprob daran angepasst wird. Dadurch kann diese Gewichtung mit in die Kategorisierung einfließen, aber eigentlich wird dann kein richtiger Naiver Baes-Classifier mehr implementiert.

3. Was könnten Sie mit dem in dieser Übung implementierten Classifier noch klassifizieren? Geben Sie eine für Sie interessante Anwendung an.

Es könnte ein selbstlernendes, benutzergesteuertes Recommender System, das ohne große Benutzerdatenbanken arbeitet, implementiert werden.
Buchbeschreibungen werden automatisch durchsucht, und anhand der Bücher, die eine Person gelesen hat und positiv bewertet hat werden neue Bücher vorgeschlagen. Wenn diese Person zum Beispiel nur Fantasy-Romane liest, sollte die Klassifizierung sehr gut funktionieren.
Es könnten Kochrezepte klassifiziert werden. Da Rezepte nur aus Zutaten und Verarbeitungsanweisungen bestehen ist der häufig vorkommende Wortschatz viel kleiner und konsistenter als bei allgemeinen Mails, oder Presseartikeln. Daher sollten sich schon durch eine kleinere Menge Trainingsdaten bessere Ergebnisse erreichen lassen. Es ist sowohl ein persönliches Recommender System vorstellbar auch ein “ähnliche Rezepte und Varianten”-Finder.

4. Das einmal trainierte, sollte eigentlich persistent abgespeichert werden. Beschreiben Sie kurz wie Sie das für dieses Programm machen würden.

Am Geschicktesten wäre es vermutlich, mit einer Datenbank zu arbeiten, in der alle Wörter und die Feeds dazu gespeichert werden. Diese Datenbank kann dann bei Programmstart wieder eingelesen werden.

Alternativ könnte auch das Objekt mit dem gelernten Modell serialisiert und auf der Festplatte gespeichert werden.
Dabei müssten die Dictionaries fc, cc und in unserem Fall das Array classes gespeichert werden, was relativ trivial sein sollte. Schwieriger wird es, die Methode getfeatures zu speichern und zu laden. Allerdings könnte hier eine Factory programmiert werden, die für ein bestimmtes Token die richtige Funktion erzeugt.
Dies kann einfach durch die Bibliothek pickle[1] realisiert werden, die das komplette Classifier-Objekt inklusive Funktionen serialisieren und deserialisieren kann, was wir auch testweise implementiert haben.

Wenn der Classifier nicht direkt gespeichert werden kann und keine Datenbank zur Verfügung steht, sollten zumindest die einzelnen RSS-Feeds (zum Beispiel in Text-Dateien) gespeichert werden, damit nicht ständig auf die Webserver zugegriffen wird.

[1] pickle - Eine Bibliothek zum Serialisieren und Deserialisieren von Python Objekten. Informationen dazu unter https://docs.python.org/2/library/pickle.html