Programmierkurs Python II – SS 2013

Übung 10

---

1 Vektoren (1 Punkt)

Vervollständige die Wörterbuch-basierte Klasse für Vektoren. Neben den vorgegebenen Methoden für Addition, Skalarmultiplikation, Cosinus und Distanz soll die Klasse auch Methoden für Subtraktion (`__sub__`), Division (`__div__`) und Centroide (`centroid(vecs)`, statische Methode) implementieren.

2 Semantische Ähnlichkeit (4 + 1 Punkte)

Implementiere (im `Vector`-Modul) eine Klasse `VectorModel`, die auf Grundlage einer gegebenen Dokumentsammlung die zu einem Eingabewort `w` die `n` ähnlichsten Wörter in der Dokumentsammlung berechnet. `add_words(self, w_iterator, window=5)` soll ein Dokument (als Wort-Iterator, siehe Template) einlesen und für alle Wörter im Dokument entsprechende Kontextvektoren mit der Kontext-Fenster-Größe `window` berechnen. Ein Kontextvektor speichert, wie häufig ein Wort mit anderen Wörtern innerhalb des Kontextfensters – z.B. jeweils die 5 Wörter links und rechts vom aktuellen Wort – zusammen vorkommt (kookkurriert).

Die Methode `get_similar_words(self, word, n=10)` soll auf Grundlage der eingelesenen Dokumente den Vektor des gegebenen Wortes mit allen anderen gelernten Vektoren vergleichen und die n ähnlichsten Worte zu word zurückliefern. *(3 Punkte)*

Wir geben eine kleine Dokumentsammlung zum Testen vor (`nyt199407.tar.gz`). Betrachte Resultate mit verschiedenen Kontextgrößen und Ähnlichkeitsmaßen (siehe Tests), z.B. die Ähnlichsten Wörter zu *man*. Versuche, eine Einschätzung und Erklärung der unterschiedlichen Ergebnisse (kurz, höchstens eine halbe Seite, als PDF oder Textdatei). *(1 Punkt)*

Bonus: Schreibe eine Funktion `convert_to_pmi(self)` in der VectorModel-Klasse, die die Kookkurenz-Werte in den Vektoren durch PMI-Werte ersetzt (siehe Einführungsfolien zu Machine Learning).

3 WSD mit kNN I (Bonus, 4 Punkte)

Implementiere den kNN-Algorithmus in einer Klasse `KNN` und benutze ihn für Word Sense Disambiguation. Trainiere Dein Programm mit den Trainingsdokumenten aus Übung 9 und klassifiziere die Test-Dateien. Der kNN-Klassifizierer soll ähnlich aufgebaut sein wie der Bayes-Klassifizierer aus Übung 9 *(3 Punkte)*:

- `readTrainingCorpus(self, class,file)` liest ein Trainings-Korpus für eine Kategorie ein
- `classify(self, file, distance=Vector.cosine)` klassifiziert ein Test-Dokument unter Benutzung des Ähnlichkeitsmaß `distance`. (Beachte, das Kosinus eine Ähnlichkeit ausdrückt, und Euklid-Distanz eine eigentliche Distanz!)

Implementiere zusätzlich eine Methode `convert_to_tfidf(self)`, die die Vektor-Einträge in tf-idf-Werte konvertiert (relativ zu den Vektoren). Für die Berechnungs-Vorschrift kannst Du das Programmier-Projekt aus Python I ansehen. *(1 Punkt)*

Wie im vorigen Übungsblatt soll hier WSD als Dokument-Klassifikation aufgefasst werden; das k für kNN ist in dem Fall 1. Abhängig vom Ähnlichkeitsmaß werden bis zu 2 der 3 Testdateien richtig klassifiziert (Euklid, mit tf-idf).

4 WSD mit kNN II (Bonus, 3 Punkte)

Implementiere den kNN-Algorithmus nicht als dokument-basierte Disambiguierung, sondern als Disambiguierung auf Basis einzelner Wort-Auftreten. Ähnlich wie in Aufgabe 2 ist jedes Auftreten des Zielwortes ein Vektor, sowohl in den Trainings- als auch in den Testdaten. Schreibe eine zweite Klasse `KNNWindow`, bei der die beiden Methoden aus Aufgabe 3 implementiert aber entsprechend modifiziert werden. In der `__init__`-Methode soll auch die Wortfenster-Größe (siehe Aufgabe 2) übergeben werden. `classify` soll dann die Kategorie zurück geben, die für die meisten Ziel-Wort-Auftreten im Testkorpus ausgegeben wurde (Hilfreich: Print-Ausgabe aller klassifizierten Instanzen.)

Bei richtiger Implementierung, Kosinus-Maß und Fenstergröße 10 funktioniert diese WSD-Variante für alle Testdateien.

5 k-means Algorithmus (Bonus, 3 Punkte)

Implementiere den k-means Algorithmus, mit einer der in der Vorlesung vorgestellten Abbruchbedingungen. Zum initialisieren kann man entweder zufällige Werte wählen, oder aus

der Vektormenge zufällige Elemente wählen. Zufallszahlen können mit den Funktionen aus dem Modul `random` erzeugt werden. Benutze die Dokumentsammlung `nyt199407.tar.gz` zum Testen.

Hinweis: Weitere Details und Implementierungshinweise finden sich in Manning, Raghavan und Schütze (2008), Kapitel 16:

`http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html`

Abgabe bis Donnerstag, 11.07.2013, 11:00 Uhr