

## Blatt 9

Abgabe bis Montag, 14. Juni 2010, 13:00 Uhr bei Dr. Konrad Rieck (rieck@cs.tu-berlin.de)

Auf diesem Übungszettel sollen Features für Textmining untersucht werden. Die praktischen Aufgaben bestehen aus einem Pythonteil und einem Matlabteil. Informationen zu python finden sich z.B. auf <http://python.org/>

1. **(15 Punkte)** Vervollständige die Klasse DocumentAnalyzer in `sheet09.py`. Die Klasse soll die Worthäufigkeiten für eine Reihe von Dokumenten berechnen und speichern, sowie zählen, in wievielen Dokumenten Wörter vorkommen, und am Ende die Information in einem für Matlab lesbaren Format (als Skript) speichern.
  - `clean_word` soll aus einem String sämtliche Satzzeichen entfernen und den gesäuberten String zurückgeben.
  - `count_words` bekommt einen Dateinamen und ein Klassenlabel (1 oder -1) übergeben und lädt die Datei und zählt die Wörter.
  - `write_as_matlab` schreibt die Ergebnisse als Matlabskript. Die folgenden drei Variablen sollen definiert werden:
    - `words` Ein Cell-Array, das alle gefundenen Wörter auflistet.
    - `word_counts` Eine Matrix, in der jede Zeile die Worthäufigkeiten für ein Dokument enthält. Die Spalten sind so angeordnet, wie in `words`.
    - `doc_counts` Ein Vektor, der auflistet, in wievielen Dokumenten ein Wort auftaucht.
    - `doc_class` Ein Vektor, der die Klassenzugehörigkeiten enthält.
2. **(15 Punkte)** Anschließend werden die Daten in Matlab analysiert und visualisiert.
  - `tf` berechnet aus der `word_counts`-Matrix die Term-Frequencies durch Normalisierung der Zeilen.
  - `tf_idf` berechnet aus der `word_counts`-Matrix die TF.IDF Scores.
  - `similarities` berechnet die Skalarprodukte zwischen allen Merkmalen.
  - `plot_sim` Plottet die Ähnlichkeiten. Hierbei sollen um die jeweiligen Klassen noch Quadrate gezeichnet werden. Es kann davon ausgegangen werden, dass die Klasse mit Label 1 zuerst auftritt.
  - `show_top_words` listet aus einem Merkmalsvektor die ersten  $n$  Wörter mit den höchsten Werten auf.
  - `collect_word_counts` fasst die Worthäufigkeit für die angegebenen Indizes zusammen.

---

```
from glob import glob
import re
```

```
class Histogram(object):
    """A simple class which is basically a dictionary
    of counts. inc increases counts for keys by one. The
    __str__ method is maybe a bit more complex than it needs
    to be because it plots the entries sorted in descending
    order by count."""
    def __init__(self):
        self.counts = dict()

    def inc(self, key):
        self.counts[key] = self.counts.get(key, 0) + 1
```

```

def keys(self):
    return self.counts.keys()

def __str__(self):
    out = []
    def cmp_freq(x, y):
        if self.counts[x] < self.counts[y]:
            return 1
        elif self.counts[x] > self.counts[y]:
            return -1
        else:
            return cmp(x, y)
    for word in sorted(self.counts.keys(), cmp_freq):
        out.append("%s: %d" % (word, self.counts[word]))
    return "\n".join(out)

def __getitem__(self, w):
    return self.counts.get(w, 0)

class DocumentAnalyzer(object):
    def clean_word(self, w):
        # ...

    def count_words(self, fn, cl):
        # ...

    def write_as_matlab(self, fn):
        # ...

counts = DocumentAnalyzer()
for fn in sorted(glob('data/money*')):
    print "analyzing " + fn
    counts.count_words(fn, 1)

for fn in sorted(glob('data/not-money*')):
    print "analyzing " + fn
    counts.count_words(fn, -1)

print "writing results"
counts.write_as_matlab('data.m')

```

---

```

function out = sheet09

data % load data prepared by sheet09.py

% compute word counts, tf-idf, and term frequencies
word_counts = word_counts;
tf = tf(word_counts);
tfidf = tf_idf(word_counts, doc_counts);

% plot similarities (scalar products) for all three
% features
figure(1)
S = similarities(word_counts);
plot_sim(S, doc_class);

```

```

title('similarities using the raw word_counts');

figure(2)
S = similarities(tf);
plot_sim(S, doc_class);
title('similarities using the term-frequencies');

figure(3)
S = similarities(tfidf);
plot_sim(S, doc_class);
title('similarities using the tf-idfs');
caxis([0, 1])

% collect word counts to compare classes
wc1 = collect_word_counts(word_counts, doc_class == 1);
wc2 = collect_word_counts(word_counts, doc_class == -1);

% first, we take term-frequencies
wc1 = tf(wc1);
wc2 = tf(wc2);

fprintf('Top 20 words for positive and negative class using\n');
fprintf('term frequencies\n');

fprintf('Top 20 positive words:\n');
show_top_words(20, wc1, words)
fprintf('Top 20 negative words:\n');
show_top_words(20, wc2, words)

% now, we weight the term frequencies by the invers document
% frequency
tfidf1 = tf_idf(wc1, doc_counts);
tfidf2 = tf_idf(wc2, doc_counts);

fprintf('Top 20 words for positive and negative class using\n');
fprintf('tf-idf scores\n');

fprintf('Top 20 positive words:\n');
show_top_words(20, tfidf1, words)
fprintf('Top 20 negative words:\n');
show_top_words(20, tfidf2, words)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Your solution below!
%

% 1. compute term frequencies from word counts
function word_counts = tf(word_counts)
% ...

% 2. compute the TF-IDF statistic
function score = tf_idf(word_counts, doc_counts)
% ...

% 3. compute linear similarities (scalar products between
% all *rows* of matrix feats.
function S = similarities(feats)
% ...

% 4. plot similarities. Also plott boxes around the classes. You can
% assume that the class where doc_class == 1 come first.

```

```

function plot_sim(S, doc_class)
% ...

% 5. print the top n entries in each *row* of feats. Use the words
% cell-array to print the real names.
function show_top_words(n, feats, words)
% ...

% 6. from word_counts, sum rows in index I
function wc = collect_word_counts(word_counts, I)
% ...

```

---

Für Fragen zum Übungsblatte bitte in der Google Group <http://groups.google.com/group/mikiobraun-lehre> registrieren und die Frage an die Mailingliste stellen.