Universität Hamburg
Department Informatik
Wissenschaftliches Rechnen
Dr. J. Kunkel, H. Hatzel, J. Gresens

Übungsblatt 4 zur Vorlesung
Big Data Analytics
im WiSe 2015/2016
Abgabe: 13.11.2015, 525 P

Bitte dokumentieren Sie die benötigte Bearbeitungszeit für die einzelnen Aufgaben in `bearbeitungszeit.txt`. Bitte denken Sie auch daran uns Feedback zur Veranstaltung zu geben:

`http://goo.gl/forms/B01IIDHvW2`

Bei der Abgabe von Source Code kommentieren Sie diesen bitte.

Als Vorbereitung der Übungsaufgaben erstellen Sie zunächst in Hadoop einen persönlichen Ordner:

```
1 hadoop fs -mkdir hdfs://abu1.cluster/user/<ihrLogin>
```

1 Wikipedia Schema um Kategorien erweitern (60 P)

Im Folgenden wollen wir unsere Wikipedia-Datenmodell in `wiki-clean.csv` weiter mit extrahierten Informationen anreichern. Aktuell besteht die Datei aus dem Schema:

```
1 ArtikelID,Artikelname,Änderungsdatum,ArtikelText
```

Hierbei könnten wir theoretisch so vorgehen, dass wir in der CSV-Datei neue Spalten einführen. Dies ist jedoch umständlich und erfordert, dass wir die bestehenden Daten weiter mit führen. In der SQL-Datenbank können wir für neue Daten jedoch leicht das bestehende Schema anpassen und entweder neue Spalten hinzu fügen oder eine neue Tabelle mit der ArtikelID einführen. In der Datenbank ist es jedoch aus Leistungsgründen nicht sinnvoll den ArtikelText abzulegen, diese behalten wir auf diesem Grund auf jeden Fall in der CSV-Datei bei. Langfristig wird es definitiv nötig werden, die erweiterten Daten aus der Datenbank als CSV-Datei zu exportieren, d.h. an dieser Stelle wird für die kurzfristige Bequemlichkeit mit SQL zusätzlicher Aufwand mit dem Export nötig werden[1]. Entscheiden Sie sich für eine der beiden Varianten (Postgres oder CSV)[2].
Erweitern Sie das Datenmodell um eine Liste von Kategorien für einen Artikel (ArtikelID) einzufügen, die erste Kategorie bezeichnen wir hierbei als Primärkategorie. Falls, Sie sich für eine CSV-Datei entscheiden so sollten die Daten wie folgt abgelegt werden:

```
1 ArtikelID,Artikelname,Änderungsdatum,Primärkategorie,ListeMitKategorien,ArtikelText
```

Implementieren Sie in Python ein Skript, welches zu jedem Artikel seine Kategorien extrahiert und entweder in ein Datenmodell der SQL-Datenbank oder einer CSV-Datei integriert.

1.1 Hinweise

Sie können die Datei `/home/bigdata/wiki-clean.csv` als Eingabe verwenden.
Im Wikimarkup sind Kategorie Angehörigkeiten wie folgt formalisiert:

```
1 [[Category:City-states]]
```

Für einen Artikel finden Sie die zugehörigen Kategorien unter der Sektion *External links*.
Es besteht auch die Möglichkeit eine (optional auch leere) Sortierungs ID anzugeben um die Kategorien auf der Seite explizit zu sortieren.

[1] Den Export müssen Sie im Rahmen der Aufgabe nicht entwickeln.
[2] Falls Sie Interesse haben können Sie auch SQL-Lite als Alternative zu Postgres verwenden.

```
[[Category:Hamburg| ]]

[[Category:Germany| 01]]
```

Sie dürfen, auch wenn eine Sortierungs ID existiert, annehmen, dass die erste Kategorie im Markup auch die primäre ist.
Weitere Informationen zu dieser Syntax finden Sie unter `https://en.wikipedia.org/wiki/Help:Category`.

Abgabe:

`1-category-extract.py` Ihr Python Skript zur Extrahierung der Kategorien

2 Wikipedia-Kategorien als WordCloud visualisieren (45 P)

In dieser Aufgabe werden wir zu einem jeden Wort herausfinden mit welchen Kategorien es assoziiert ist. Dies wollen wir als eine Wordcloud[3] darstellen. Hierbei sollen zu einem Suchwort die relevanten Artikel und deren Primärkategorien ermittelt und die häufig vorkommenden Kategorien in der Wordcloud groß (d.h. relevant) dargestellt werden. Als relevant bezeichnen wir einen Artikel im dem das Suchwort häufiger als ein bestimmter (von der Kommandozeile setzbarer) Schwellwert vorkommt.
Schreiben Sie hierfür ein Python Skript welche Ihre Datenbank (und CSV-Datei) aus Aufgabe 1 nutzt.
Dazu iterieren Sie, wie gewohnt, über den Datensatz, und sehen falls es ausreichend viele Vorkommnisse des Suchwortes in einem Artikel gibt in der Datenbank nach welches die jeweilige primäre Kategorie ist.
Ein Beispiel: Suchwort ist „Apple” . „Apple” kommt in vielen Artikeln der Kategorie „Apple Inc” vor, allerdings auch in einem Artikel der Kategorie Isaac Newton.
So erhält man die Grafik in Abbildung 1

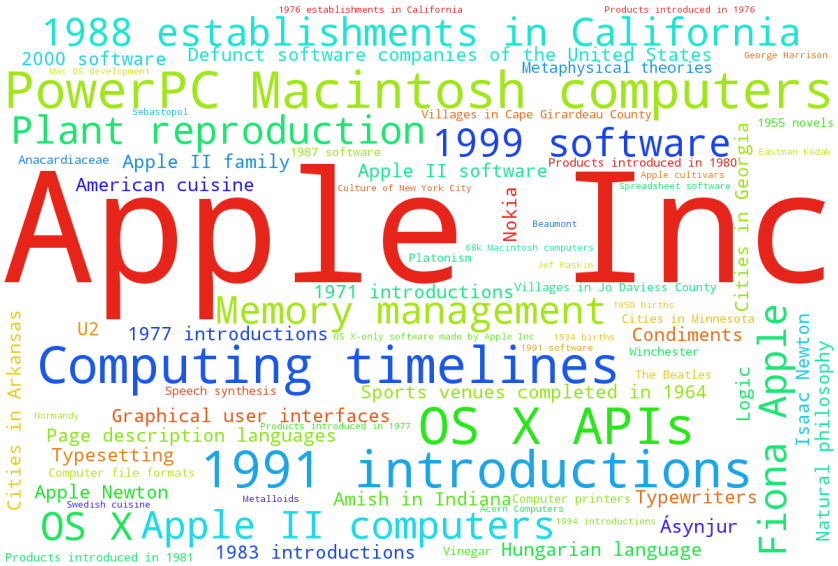

Abbildung 1: Kategorien in denen das Wort „apple” vorkommt

2.1 Hinweise

Wir verwenden das Paket `wordcloud` um die Grafik zu generieren.
Wenn Sie auf einem eigenen Rechner auch das Paket installieren wolllen nutzen Sie bitte nicht die PyPi Version sondern die folgende, da ansonsten Inkompatibilitäten auftreten.

```
pip install git+git://github.com/amueller/word_cloud.git@b35ed472b5b93
```

[3] Eine Wordcloud wird auch Tagcloud genannt `https://en.wikipedia.org/wiki/Tag_cloud`

2.1.1 Python Code-Gerüst

```python
#!/usr/bin/env python3
from wordcloud import WordCloud
import csv
import re
import sys

csv.field_size_limit(sys.maxsize)

categories = {}
searchword = "apple"
def get_cat(text):
    ...

csv_file = open("wikipedia-text.csv", "r")
reader = csv.reader(csv_file)
for _, title, _, text in reader:
    ...

wordcloud = WordCloud(relative_scaling=.5)
wordcloud.generate_from_frequencies(list(categories.items()))

image = wordcloud.to_image()
print(image.size)
image.save(searchword + ".png")
```

Abgabe:

`2-wiki-wordcloud.py` Ihr Python Skript zur Erstellung der Python wordclouds

3 Wikipedia Schema erweitern um geographische Koordinaten (90 P)

In dieser Aufgabe erweitern wir wiederum das Wikipedia-Schema (Siehe Aufgabe 1 für eine Beschreibung). Für eine der nächsten Aufgaben interessieren wir uns für Länder und Städte in Verbindung mit ihren Koordinaten. Glücklicherweise enthält der Wikipedia Datensatz bereits viele solche Informationen, in dieser Aufgabe wollen wir dies extrahieren und diese zum bestehenden Wikipedia-Schema (Siehe Aufgabe 1) hinzu fügen, d.h. Ihre CSV-Datei oder Postgres-Datenbank enthält nach der Erweiterung für jede ArtikelID noch Longitude und Latitude.

3.1 Hinweise

In vielen Artikeln zu bestimmten Ländern oder Orten befindet sich eine Infobox, der relevante Teil ist wie folgt aufgebaut:

```
{{Infobox
... # Sonstige Information, welche irrelevant ist
 | iso region          = DE-HH
 | PLZ                 = 20001-21149, 22001-22769
 | coordinates_display = title
 | latd                = 53
 | latm                = 33
 | lats                = 55
 | latNS               = N
 | longd               = 10
 | longm               = 00
 | longs               = 05
 | longEW              = E
```

```
 | date               = August 2010
}}
```

Wandeln Sie die Koordinaten von Minuten und Sekunden in eine einheitliche Dezimalform um. Behandeln Sie dabei die Nord bzw. Ost Richtung als Vorzeichen für ihren Dezimalwert. Wobei Punkte im Süden und Westen des Äquators bzw. Nullmeridians negative Werte annehmen. Bedenken Sie das nicht alle Artikel den gleichen Konventionen bezüglich Leerzeichen, Zeilenumbrüchen usw. folgen.
Speichern Sie die Dezimalwerte für Longitude und Latitude. Bedenken Sie dabei, dass wir später ausgehend vom ArtikelNamen ausgehend die Koordinaten erfragen können.

3.1.1 Python Code-Gerüst

```python
#!/usr/bin/env python3
import csv
import re
import sys
csv.field_size_limit(sys.maxsize)

f = open("/home/bigdata/wikipedia-text.csv")

def save_to_db(title, lat, lon):
    ...

def lat_to_decimal(d, m, s, NS):
    # See https://en.wikipedia.org/wiki/Decimal_degrees for data conversion
    ...
    return degree

def lon_to_decimal(d, m, s, EW):
    ...
    return degree

reader = csv.reader(f)
for line in reader:
    _, title, _, text = line
    # Regex Suche
    save_to_db(title, lat, lon)
```

3.1.2 *Optional:* Mehr Koordinaten finden

Manche Artikel zu Orten verwenden nicht die Infobox sondern stattdessen das Coord Template mit dem tag `display=title`. Parsen Sie falls es keinen Eintrag in der Infobox gibt auch diese.
Eine Erklärung dieser Syntax Elemente finden Sie unter `https://en.wikipedia.org/wiki/Template:Coord`.

Abgabe:

`3-coord-exctractor.py` Ihr Skript um Koordinaten zu extrahieren.

4 MapReduce: Verarbeitung der Wikipedia-Daten mit Python (90 P)

Hilfe von MapReduce parallelisieren wir nun die Datenverarbeitung unseres Programms zur Ermittlung der Worthäufigkeiten. Ziel ist es eine Liste mit der ArtikelID und den dazu passenden Worten mit ihren Häufigkeiten zu erhalten, d.h. Worte die nicht vorkommen in einem Artikel, werden nicht in der Liste auftauchen. Das Ausgabetupel sollte in der Form sein[4]:

[4] Es steht Ihnen frei den WortArray als JSON auszugeben und das Escaping zu korrigieren

```
"ArtikelID","Titel",Wortanzahl,"[Wort1:4,Wort2:3,...]"
```

Ermöglichen Sie darüber hinaus die Summe aller Worthäufigkeiten über alle Artikel zu berechnen und dieses als CSV abzulegen:

```
Wort1,4
Wort2,3
...
```

Verwenden Sie hierfür *wiki-clean.csv* als Eingabe[5]. Ihre Ausgabe wollen wir *wiki-clean-frequency.csv* nennen. Passen Sie hierfür Ihr Python Programm zum Zählen der Wörter in einer CSV-Datei an, damit es in einen MapReduce Job integriert werden kann.
Ermitteln Sie die Laufzeit für die Bearbeitung der Eingabedatei, führen Sie Ihr Programm einmal als Pipe in der Kommandozeile aus und einmal mit Hadoop.

4.1 Hinweise

Das Hadoop Streaming, das über das JAR hadoop-streaming bereit gestellt wird[6] bietet eine einfache Möglichkeit beliebige Programme als Map oder Reduce Funktion einzusetzen. Hierbei wird die externe Anwendung anstelle von Map oder Reduce aufgerufen und die Key/Values als Eingabe für dieses Programm verwendet. Die Text-Ausgabe des Programms wird als Ausgabe von Map bzw. Reduce verwendet. Dies kann sehr einfach dafür verwendet werden um Python einzubinden[7]:

```
yarn jar /usr/hdp/2.3.2.0-2950/hadoop-mapreduce/hadoop-streaming.jar \
  -Dmapred.reduce.tasks=1 -Dmapred.map.tasks=11 -mapper $PWD/mein-map.py -reducer $PWD/mein-reduce.py \
  -input <input> -output <output-directory>
```

Beispielsweise könnten triviale Map und Reduce Jobs wie folgt formuliert werden:

```
#!/usr/bin/python3
import sys

# This program outputs for every input tuple (line for mapper, key/value pair for reduce)
# the fixed tuple ("key", "value"). It is possible to filter and aggregate tuples (in reduce).
for line in sys.stdin:
   print("key\tvalue\n")
```

Zusätzlich vereinfacht es die Fehleranalyse, da das Programm sehr leicht getestet werden kann:

```
cat Input.csv | ./map.py | sort | ./reduce.py
```

Abgabe:

`4-map.py`	Ihre Map Funktion in Python, welche die CSV-Datei verarbeitet.
`4-reduce.py`	Ihre Reduce Funktion zum Zählen der Wörter einzelner Artikel.

5 MapReduce: Gruppierung von Daten (210 P)

In dieser Aufgabe werden wir einen neuen Datensatz verwenden: `data-energy-efficiency.csv`. Der Datensatz wurde bereits in Hadoop importiert und steht für Sie unter /user/kunkel/data-energy-efficiency.csv zur Verfügung. Der Datensatz enthält für verschiedene Experimente Zeitreihen. Ziel der Aufgabe ist es die Werte der einzelnen Experimente zu aggregieren und den Mittelwert für jede Spalte zu berechnen.

[5] Sie können gerne unsere Version verwenden.
[6] `http://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html`
[7] http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

5.1 Datensatz: data-energy-efficiency.csv

Dieser Datensatz enthält Messungen der CPU Performance Counter (Operationen die die CPU durchführte, z.B. Gleitkommaoperationen). Hierbei wurden verschiedene Experimente wiederholt durchgeführt und Zeitreihen für jedes Experiment erzeugt. Bis zu vier unterschiedliche Anwendungen werden gleichzeitig ausgeführt und ca. 260 CPU Counter gemessen[8]. Der Header der CSV-Datei ist wie folgt aufgebaut.

```
App0,App1,App2,App3,Time,AGU_BYPASS_CANCEL_COUNTCPU0,(<COUNTER>)+
```

Ein Auszug der ersten 8 Spalten der Datei:

```
ESM_example,ESM_example,ESM_example,ESM_example,2.750738,55.40407,705.32129,1041.9067
ESM_example,ESM_example,ESM_example,ESM_example,2.800752,44.90113,711.82311,1036.9053
ESM_example,ESM_example,ESM_example,ESM_example,2.850765,44.2288,719.9632,1030.6752
ESM_example,ESM_example,ESM_example,ESM_example,2.900779,45.72922,728.46558,1024.17338
ESM_example,ESM_example,ESM_example,ESM_example,2.950792,46.40987,731.63948,1013.98286
ESM_example,ESM_example,ESM_example,ESM_example,3.000805,46.91,733.64,1002.98
ESM_example,ESM_example,ESM_example,ESM_example,3.050819,78577.32608,26034.71632,54999.62226
ESM_example,ESM_example,ESM_example,ESM_example,3.100832,174338.21744,56886.73576,120846.23793
ESM_example,ESM_example,ESM_example,ESM_example,3.150846,121595.80666,57856.18875,96096.16269
...
EXX_example,EXX_example,EXX_example,EXX_example,5.652014,221434.54323,211603.92431,369932.99742
EXX_example,EXX_example,EXX_example,EXX_example,5.702031,107861.4415,179096.3755,234573.991
EXX_example,EXX_example,EXX_example,EXX_example,5.752049,31701.88224,128184.82916,122750.35068
EXX_example,EXX_example,EXX_example,EXX_example,5.802067,13279.25248,48874.78782,47248.67986
EXX_example,EXX_example,EXX_example,EXX_example,5.852085,1819.22816,661.59928,1395.22376
EXX_example,EXX_example,EXX_example,EXX_example,5.902103,1070.95888,289.96554,1154.63718
```

Würde der Mittelwert der einzelnen Messwerte für die Zeitreihen berechnet, so ergibt sich:

```
ESM_example,ESM_example,ESM_example,ESM_example 112.69512266727315,186388.93997432772,118469.79527709562
EXX_example,EXX_example,EXX_example,EXX_example 118.44219725094219,251865.2199417397,361377.21140772087
VCSexample,VCSexample,VCSexample,VCSexample     13.882197396378269,409238.7635984908,286333.5696854528
cpu,cpu,cpu,cpu                                 30.69008190862511,6085.659376959866,6830.574735098203
```

5.2 Hinweise

Ein Codegerüst in Java und die nötigen Befehle zur Kompilation wird zur Verfügung gestellt unter `04-hadoop-energy-efficiency.Java`. Zur Vorbereitung der Übungen stellen Sie sicher, dass Sie in Eclipse die nötigen JAR Dateien importiert haben (diese werden im Bash-Snippet aufgelistet).

5.2.1 Bash-Skript für die Compilation und Ausführung

```
# compilation
javac -classpath /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core.jar:
  /usr/hdp/current/hadoop-hdfs-client/hadoop-hdfs.jar:
  /usr/hdp/2.3.2.0-2950/hadoop/hadoop-common.jar -d classes AveragePerformance.java || exit 1

# build a JAR Archive from the classes
jar -cvf averagePerformance.jar -C classes . || exit 1

# Execute the JAR Archive with Hadoop
hadoop jar averagePerformance.jar de.wr.AveragePerformance data-energy-efficiency.csv summary
```

Abgabe:

`5-MapReduceAverage.Java` Ihr SourceCode für die Berechnung des Mittelwertes.

[8] Da nur einige Counter gleichzeitig gemessen können, wurden die Messungen mehrfach wiederholt.

6 Datenformate: Kompression der Wikipedia-Daten (30 P)

Hadoop unterstützt verschiedene Datenformate, die Verwendung von TEXT-Dateien im CSV-Format bringt einige Nachteile mit sich. In dieser Aufgabe wollen wir Eingangsdaten (automatisch) in verschiedene Datenformate konvertieren und komprimieren.
Zunächst wollen wir einmal betrachten wie eine Datei in Segmente aufgeteilt und über die Server verteilt wurde, hierfür können Sie den folgenden Befehl verwenden:

```
hdfs fsck <DATEI> -files -blocks -locations
```

Eine einfache Möglichkeit eine TEXT-Datei in Segmente zu zerlegen und diese zu komprimieren bietet das Hadoop Streaming, das über das JAR hadoop-streaming bereit gestellt wird (siehe oben). Der folgende Befehl nimmt die Komprimierung vor indem er Properties übergibt[9] und die Daten durch `cut` piped:

```
yarn jar /usr/hdp/2.3.2.0-2950/hadoop-mapreduce/hadoop-streaming.jar -Dmapred.output.compress=true \
  -Dmapred.compress.map.output=true \
  -Dmapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec \
  -Dmapred.reduce.tasks=0 -Dmapred.map.tasks=10 -mapper "/usr/bin/cut -f 2" \
  -input <input> -output <output-directory>
```

Die Ausgabe erfolgt in ein Verzeichnis, wobei jedes Segment individuell komprimiert wird und somit kann die Datei nachträglich parallel verarbeitet werden. Alternativ zum GzipCodec gibt es noch DefaultCodec, BZip2Codec, SnappyCodec, lzo.LzopCodec.
Passen Sie nun Ihren MapReduce Job so an, dass dieser mit den komprimierten Daten arbeiten kann. Komprimierte Eingabe wird direkt unterstützt! Hierfür können Sie folgenden Code-Schnipsel einsetzen:

```
// Settings for YARN
// Configuration conf = this.getConf();

// Compress intermediate mapper output
conf.set("mapreduce.map.output.compress","true");
conf.set("mapreduce.map.output.compress.codec","org.apache.hadoop.io.compress.GzipCodec");

// Compress MapReduce output
conf.set("mapreduce.output.fileoutputformat.compress","true");
conf.set("mapreduce.output.fileoutputformat.compress.codec","org.apache.hadoop.io.compress.GzipCodec");
// conf.set("mapreduce.output.compression.type", "BLOCK"); // this seems to be the default

// Do whatever else you need to do to startup the process
```

Downloaden Sie die Ausgabedatei und überprüfen Sie, dass der Inhalt das korrekte Ergebnis wiedergibt. Vergleichen Sie die Verteilung der einzelnen Datensegmente mit `hdfs fsck`.

Abgabe:

`6-beobachtungen.txt` Ihre Zeitmessungen und Beobachtungen der Dateiverteilung.

[9] Die Properties für die Kompression sind hierbei für Hadoop in Version 1, die entsprechenden Eigenschaften mit YARN werden unten gezeigt.