Projektarbeit

Big Data

Matthias Körschens, Kevin Reinke

5. Februar 2017

Inhaltsverzeichnis

1 Einleitung

Ziel der Projektarbeit war es, die im Rahmen der Vorlesung „Big Data“erworbenen Fähigkeiten praktisch umzusetzen. Hierfür konnten wir einen Amazon-Review-Datensatz der Stanford University nutzen. Ein Dank geht an dieser Stelle an Julian McAuley, der so freundlich war uns diesen Datensatz auf Nachfrage zur Verfügung zu stellen. Zur Bearbeitung der Daten haben wir Apache Spark, sowie den Lofar-Rechencluster der FSU-Jena benutzt. Die Fragestellung unserer Bearbeitung war das Erkennen von Mustern und Zusammenhängen in den Daten. Im Speziellen haben wir uns auf die Verteilung der Reviews und das Bewertungsverhalten der Nutzer konzentriert.

2 Datensatz

Der in dieser Arbeit verwendete Datensatz besteht aus Reviews des Onlinehändlers Amazon aus der Zeitspanne von Mai 1996 bis Juni 2014. Es wird hier ein spezieller Teildatensatz verwendet, welcher lediglich Reviews aus der Kategorie Elektronik enthält. Dieser

hat mit 7,8 Millionen Reviews von etwa 4,2 Millionen Nutzern eine Größe von 4,7 Gigabyte. Er ist vollständig im JSON-Format gehalten. Ein Beispielreview ist nachfolgend dargestellt:


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the
     piano. He is having a wonderful time playing these old
     hymns. The music is at times hard to read because we think
      the book was published for singing from more than playing
      from. Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


Die Datensätze bestehen aus der ID des Reviewers (reviewerID), der Produkt-ID (asin), dem Namen des Reviewers (reviewerName), einer Liste mit zwei numerischen Werten, die die hilfreich-Bewertungen repräsentieren (helpful), dem Inhalt der Review (reviewText), der Produktbewertung (overall), der Überschrift bzw. Zusammenfassung des Reviews (summary), dem Datum der Review (reviewTime) und dem Datum im Unix-Format (unixReviewTime). Aufgrund der gegebenen Reviewer-ID lassen sich so auch neben den Reviews Rückschlüsse über die einzelnen Reviewer ziehen.
Der Datensatz ist verfügbar unter `http://jmcauley.ucsd.edu/data/amazon/`.

3 Strukturen

Um ein erstes Gefühl für den Datensatz zu bekommen, haben wir einige Extremfälle ermittelt. So war 520 die maximale Anzahl an Reviews, die von einem Nutzer verfasst wurden. Das längste Review im Datensatz hatte 32703 Zeichen und 6465 Wörter.

3.1 Verteilung der Reviews

Eine Fragestellung ist die Verteilung der Reviews auf die Nutzer. Hierfür haben wir die Reviews mit gleicher ReviewerID aufaggregiert und die Anzahl der Reviews, die ein Nutzer abgegeben hat, dem Nutzer zugeordnet. Dabei ist aufgefallen, dass die erfassten Nutzer eher wenige Reviews erstellen. Beispielsweise gibt es etwa 2,88 von 4,2 Millionen Nutzern, die nur ein einziges Review abgaben. In Abbildung 3.1 ist die Anzahl der Nutzer, die eine spezifische Anzahl an Reviews abgegeben haben, dargestellt. Für eine bessere Visualisierung wurden beide Achsen logarithmiert. Der unlogarithmierte Graph begindet sich im Anhang. Es liegt ein exponentieller Zusammenhang vor. Da ferner die

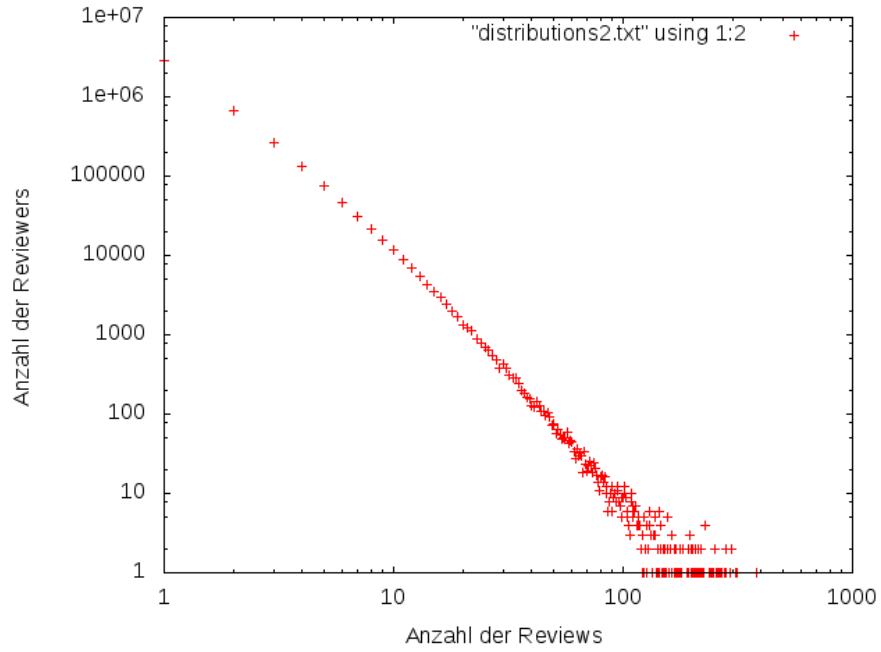


Abbildung 1: Reviewverteilung

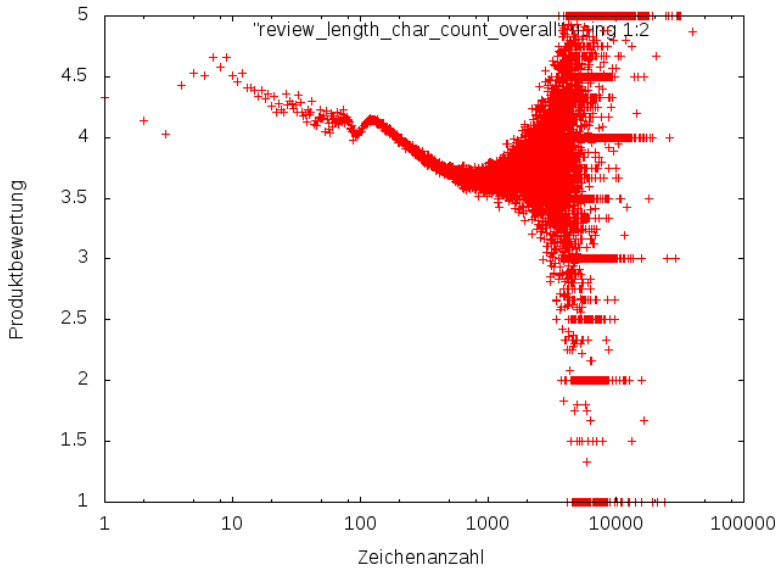
Gruppen nach der Anzahl der Reviews sortiert wurden, erinnert dies an das Zipfsche Gesetz, welchen seinen Usprung in der Linguistik hat. Dieses besagt, dass die Wahrscheinlichkeit des Auftretens der Elemente einer Menge umgekehrt proportional zu ihrer Position in der nach Häufigkeit sortierten Menge ist. Somit gilt

$$p(n) \sim \frac{1}{n}, \tag{1}$$

wobei p die Auftrittswahrscheinlichkeit und n die Position in der nach Häufigkeit sortierten Menge ist.

3.2 Reviewerverhalten bei steigender Reviewanzahl

Zunächst haben wir die Gruppierung der Nutzer nach ihrer Anzahl an Reviews beibehalten und untersucht, ob diese Gruppen unterschiedliches Verhalten aufzeigen. Innerhalb einer Gruppe wurden für die Merkmale die Gruppendurchschnitte errechnet. Die Merkmale und ihr Zusammenhang in Abhängigkeit von der Reviewanzahl sind in den Grafiken im Anhang dargestellt.
Wie im vorigen Abschnitt bereits angesprochen überwiegen die Nutzer mit wenigen Reviews. Daher ist die Durchschnittsbildung über die Merkmale bei den Gruppen mit wenigen Reviews (mehr Reviewer je Gruppe) stabiler. Für Gruppen mit mehr als 100 Reviews, kann es unter Umständen sein, dass diese nur aus einem einzigen Reviewer besteht. Dies zeigt sich in der starken Streuung der vier Grafen mit zunehmender Reviewanzahl. Aus diesem Grund wurde die Achse der Reviewanzahl logarithmiert, um so



die relevanteren Gruppen in den Fokus zu rücken. Auch, wenn der Durchschnitt für einige Gruppen nicht sehr aussagekräftig ist, so lassen sich dennoch deutliche Zusammenhänge erkennen. Nutzer die mehrere Reviews verfassen, schreiben ausführlichere Bewertungstexte und werden von anderen Nutzern als hilfreicher eingestuft. Ebenso scheinen sie Produkte leicht besser zu bewerten.

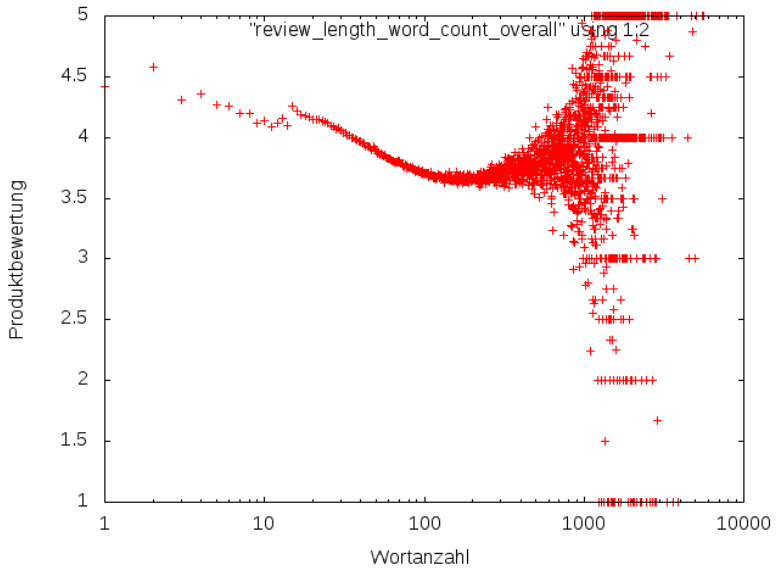
3.3 Textlänge und Produktbewertung

Ein interessanter Zusammenhang hat sich bei der Beziehung zwischen der Textlänge des Reviews und der Produktbewertung gezeigt. Die Textlänge haben wir im Folgenden auf Zeichen- und Wortanzahl untersucht. Dabei wurden alle Reviews mit gleicher Länge zusammengruppiert und über diese Gruppen die durchschnittlichen Produktbewertungen bestimmt. Die Ergebnisse dieses Versuchs sind in den Abbildungen und 3.3 dargestellt. Hier sind die Produktbewertungen mit wenigen Wörtern anfänglich zwischen 4,0 und 4,5. Mit zunehmender Textlänge bis etwa 180 Wörtern fällt die Bewertung auf 3,6 erholt sich danach aber wieder leicht. Ab 1000 Wörtern sind zuverlässige Aussgen wieder nicht mehr möglich, da die Gruppen zu klein werden.

4 Cluster

Um die Nutzer untereinander in Beziehung zu setzen, mussten wir jedem Reviewer einen Merkmalsvektor zuordnen. Diesen haben wir über arithmetische Mittelung der vom Nutzer abgegebenen Reviews bestimmt. Die Merkmale sind dabei:

- Wortanzahl des Bewertungstextes
- Zeichenanzahl des Bewertungstextes

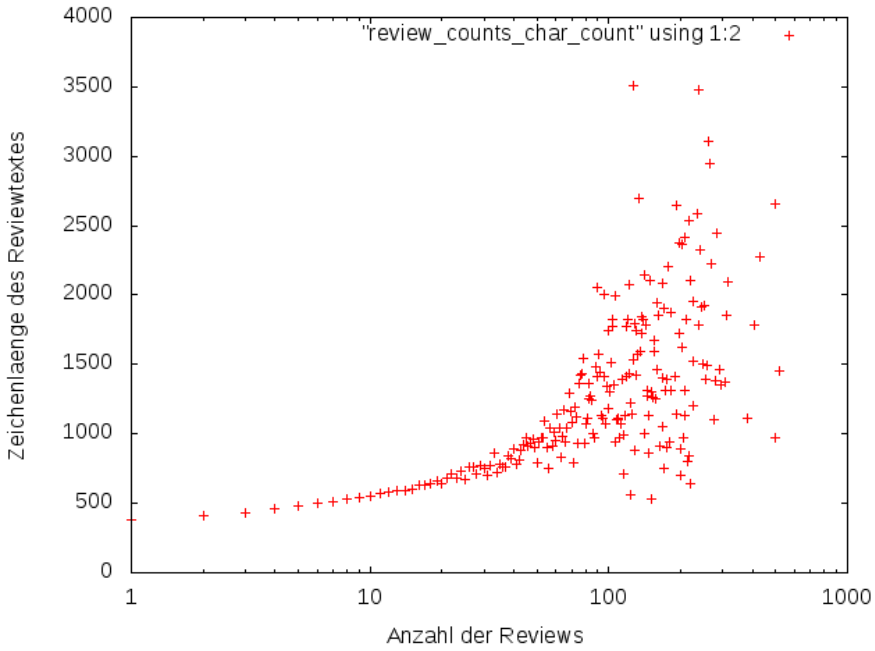


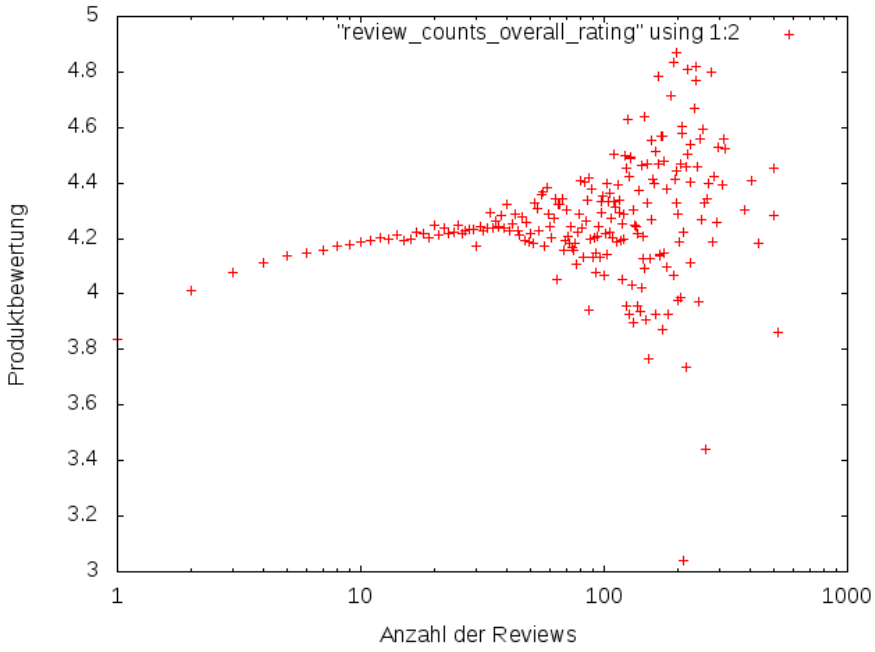
- Zeitstempel des Reviews
- Zeichenanzahl des Reviewtitels
- wie hilfreich das Review war prozentual
- wieviele das Review ingesamt als hilfreich oder nicht hilfreich bewertet haben
- Produktbewertung
- Anzahl der Reviews

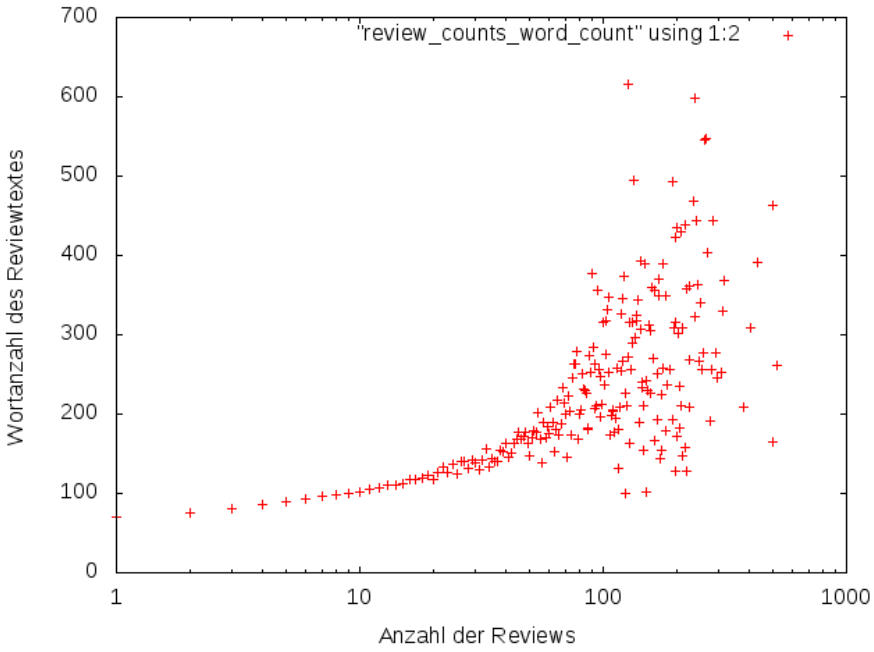
Agglomerative Clusterverfahren, welche ein Distanzmaß zwischen sämtlichen Reviewern verwenden, haben wir verworfen, da hierfür ein quadratischer Aufwand bezüglich der Anzahl der Reviewer anfallen würde.
Als ersten Schritt haben wir die Daten mit einer Hauptachsentransformation bearbeitet. Hierfür wurde die Bibliothek pyspark.mllib.feature benutzt. Leider hat sie als Rückgabewerte keine Transformationsmatrix oder Eigenwerte geliefert, sodass keine Aussagen über die Einflüsse der einzelnen Merkmale möglich waren. Im Anschluss wurden die Daten mit dem K-means Algorithmus der Bibliothek pyspark.mllib.clustering gruppiert. Der Algorithmus wurde mit 10 zufälligen Startkonfigurationen ausgeführt und über die entstandenen Merkmalsverktoren der Cluster gemittelt. Als Clusteranzahl haben wir $k = 44$ und 65 gewählt. Wir konnten die Mittelpunkte der Cluster bestimmen, jedoch gelang es uns auch nach einigen Analyseversuchen der gefundenen Cluster nicht, definitive Gemeinsamkeiten, Unterschiede und Muster in diesen zu finden. Somit können an dieser Stelle leider keine weiteren Ergebnisse vorgestellt werden.

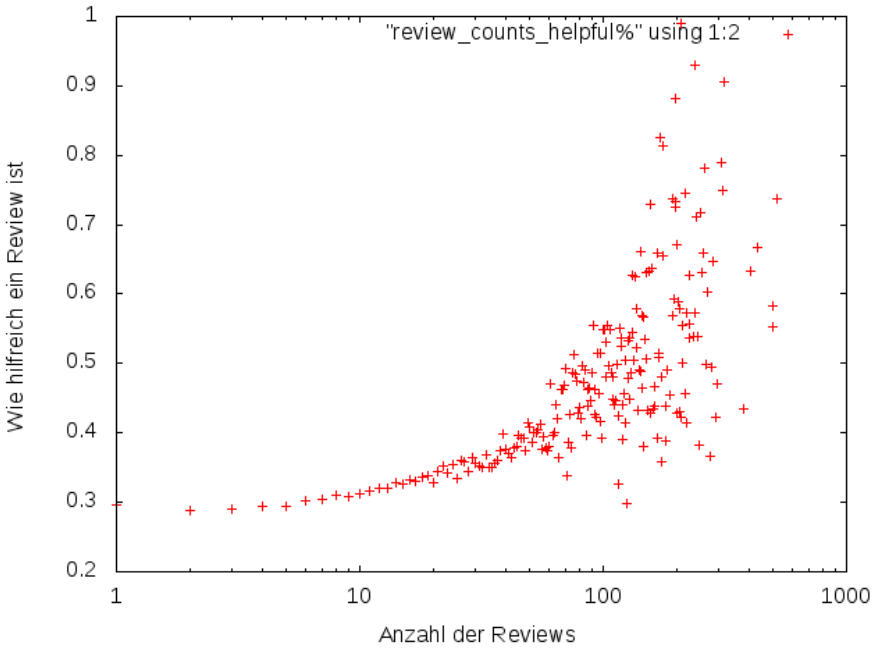
Anhang

Grafiken

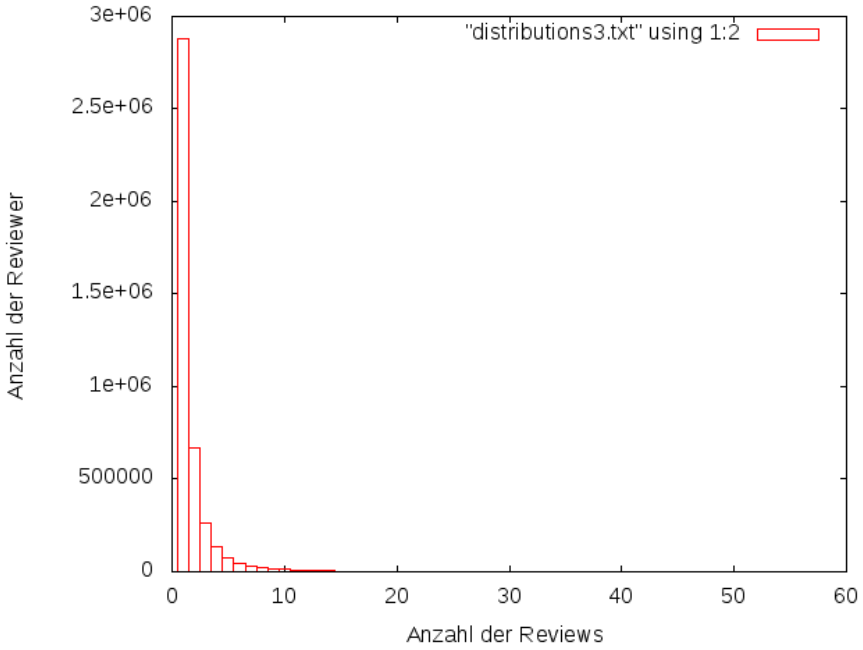


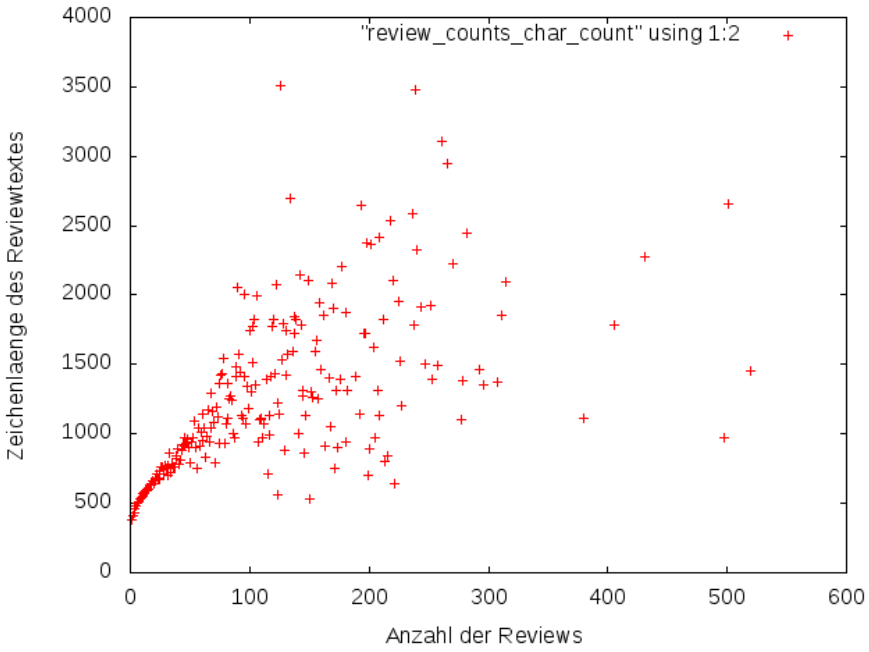


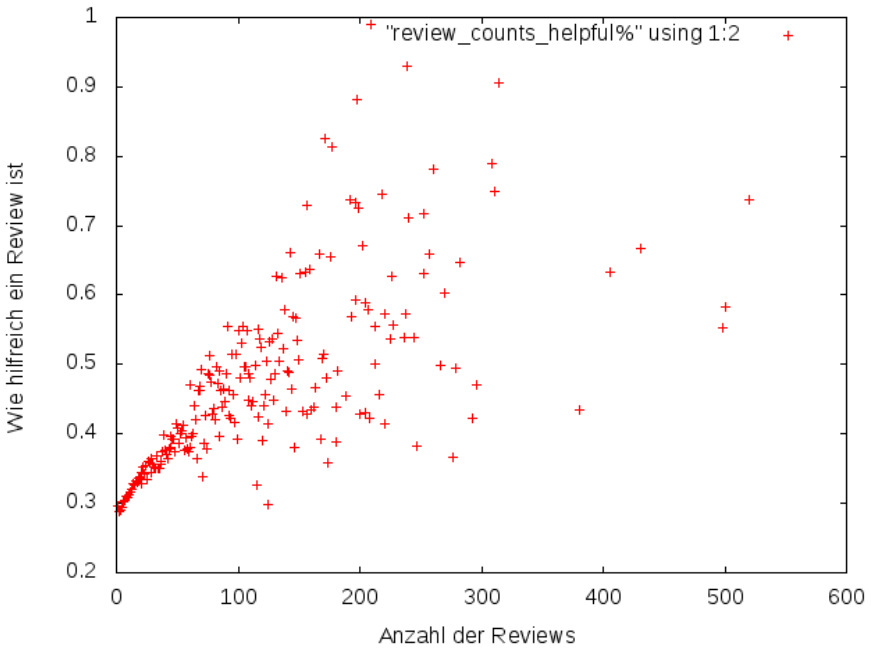

"review_counts_word_count" using 1:2
Wortanzahl des Reviewtextes
Anzahl der Reviews
0
100
200
300
400
500
600
700
1
10
100
1000


"review_counts_helpful%" using 1:2
Wie hilfreich ein Review ist
Anzahl der Reviews
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
10
100
1000

unlogarithmierte Graphen






"review_counts_helpful%" using 1:2
Wie hilfreich ein Review ist
Anzahl der Reviews
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
100
200
300
400
500
600

Code

run.sh

```
#!/usr/bin/bash

hdfs dfs -rm -r -f reviews/reviewerReviewsSample
hdfs dfs -rm -r -f reviews/kmeans_model
if [ $# -eq 0 ]
        then
        /cluster/spark/bin/spark-submit ReviewerReviews.py
        else
        /cluster/spark/bin/spark-submit ReviewerReviews.py -s
           $1
fi

cat results.txt
```

ProcessCluster.py

```
from pyspark import SparkConf, SparkContext
import json
import argparse
from os import makedirs

#run on cluster
confCluster = SparkConf()
confCluster.setMaster("yarn-client")
confCluster.set("spark.driver.host","frontend")
confCluster.setAppName("AmazonReviews")

sc = SparkContext(conf = confCluster)

result_collection = {}

input_filename = "cluster_output64"

# Read the data in json format line by line
file_data = sc.textFile("reviews/" + input_filename)

file_data = file_data.map(lambda x: x.split(" "))
file_data = file_data.map(lambda x: (int(x[0]), (x[1], float(x
    [2]), int(x[3]), float(x[4]), int(x[5]), float(x[6]), float(
    x[7]), float(x[8]), float(x[9]))))

# Reduce each review to vector of usable data elements
#vector_data = file_data.map(lambda d: (d["reviewerID"], (float
    (d["helpful"][0]) / d["helpful"][1] if d["helpful"][1] != 0
    else 0, d["helpful"][1], d["overall"], d["unixReviewTime"],
    len(d["reviewText"]), len(d["summary"]), len(d["reviewText
    "].split(" ")), 1)))

# Reduction of reviews per reviewer to vector representing the
    reviewer
# vectors: (reviewerID, (helpful%, helpful_voted, overall,
    unixReviewTime, len(reviewText), len(summary),
    word_count_review_text, reviewCount))

def averages_all_keys(rdd, value_choice_fun):
        trdd = rdd.map(lambda x: (value_choice_fun(x)[0],
            value_choice_fun(x)[1] + [1]))
        trdd = trdd.reduceByKey(lambda a, b: tuple(a[i] + b[i]
```

```
            for i in range(len(a))))
        trdd = trdd.map(lambda x: (x[0], tuple(val / x[1][-1]
            for val in x[1][:-1])))

        trdd = trdd.sortByKey()

        trdd = trdd.map(lambda x: str(x[0]) + " " + "".join("
            ".join("{}".format(val) for val in x[1])) + "\n")
        #with open("results.txt", "w") as f:
        #       f.write(str(trdd.first()))

        return trdd.reduce(lambda a, b: a + b)


# Get average char count, word count, overall rating and
    helpful% per review count
averages = averages_all_keys(file_data, lambda x: (x[0], [x
    [1][1], x[1][2], x[1][3], x[1][4], x[1][5], x[1][6], x
    [1][7], x[1][8]]))

result_collection[input_filename] = averages

# Write results to local disk
try:
        makedirs("results/cluster_results")
except OSError:
        pass

for key in result_collection:
        with open("results/cluster_results/" + key, "w") as f:
                f.write(str(result_collection[key]))
```

ReviewerReviews.py

```
from pyspark import SparkConf, SparkContext
import json
import argparse
from os import makedirs
from pyspark.mllib.linalg import Vectors
from pyspark.mllib.feature import PCA
from pyspark.mllib.clustering import KMeans, KMeansModel

# Program parameters
parser = argparse.ArgumentParser()

parser.add_argument("-s", "--sample", type=float, default=0,
    help="Use a subset of the data. Has to be a number between 0
     and 1, determining the size of the subset to use.")

args = parser.parse_args()

subset_size = args.sample

#run on cluster
confCluster = SparkConf()
confCluster.setMaster("yarn-client")
confCluster.set("spark.driver.host","frontend")
confCluster.setAppName("AmazonReviews")

sc = SparkContext(conf = confCluster)

result_collection = {}

# Read the data in json format line by line
file_data = sc.textFile("reviews/reviews_Electronics.json")
# Each dataset is divided by \n --> parse each line into dict
file_data = file_data.map(json.loads)

# Draw a sample from the dataset
if subset_size != 0:
        file_data = file_data.sample(False, subset_size,
           79347234)

# Reduce each review to vector of usable data elements
vector_data = file_data.map(lambda d: (d["reviewerID"], (float(
   d["helpful"][0]) / d["helpful"][1] if d["helpful"][1] != 0
```

```
    else 0, d["helpful"][1], d["overall"], d["unixReviewTime"],
    len(d["reviewText"]), len(d["summary"]), len(d["reviewText
    "].split(" ")), 1)))

# Reduction of reviews per reviewer to vector representing the
    reviewer
# vectors: (reviewerID, (helpful%, helpful_voted, overall,
    unixReviewTime, len(reviewText), len(summary),
    word_count_review_text, reviewCount))
reviewer_vectors = vector_data.reduceByKey(lambda a, b: tuple(a
    [i] + b[i] for i in range(len(b))))
reviewer_vectors = reviewer_vectors.map(lambda x: (x[0], [val /
     x[1][7] for val in x[1][:-1]] + [x[1][-1]]))

def averages_per_key(rdd, value_choice_fun):
        trdd = rdd.map(lambda x: (value_choice_fun(x)[0],
            value_choice_fun(x)[1] + [1]))
        trdd = trdd.reduceByKey(lambda a, b: tuple(a[i] + b[i]
            for i in range(len(a))))
        trdd = trdd.map(lambda x: (x[0], tuple(val / x[1][-1]
            for val in x[1][:-1])))

        trdd = trdd.sortByKey()

        trdd = trdd.map(lambda x: tuple("{}\t{}\n".format(x[0],
            x[1][i]) for i in range(len(x[1]))))
        #with open("results.txt", "w") as f:
        #       f.write(str(trdd.first()))

        return trdd.reduce(lambda a, b: tuple(a[i] + b[i] for i
            in range(len(a))))


# Get average char count, word count, overall rating and
    helpful% per review count
review_counts_cc, review_counts_wc, review_counts_or,
    review_counts_hp = averages_per_key(reviewer_vectors, lambda
     x: (x[1][7], [x[1][4], x[1][6],  x[1][2], x[1][0]]))

result_collection["review_counts_char_count"] =
    review_counts_cc
result_collection["review_counts_word_count"] =
    review_counts_wc
result_collection["review_counts_overall_rating"] =
```

```
    review_counts_or
result_collection["review_counts_helpful%"] = review_counts_hp


# Get average overall rating per review length (char count)
review_length_cc, = averages_per_key(reviewer_vectors, lambda x
    : (x[1][4], [x[1][2]]))
review_length_wc, = averages_per_key(reviewer_vectors, lambda x
    : (x[1][6], [x[1][2]]))

result_collection["review_length_char_count"] =
    review_length_cc
result_collection["review_length_word_count"] =
    review_length_wc

# Conduct PCA
reviewer_vectors_real = reviewer_vectors.map(lambda x: Vectors.
    dense([val for val in x[1]]))

pca_model = PCA(8).fit(reviewer_vectors_real)
transformed = pca_model.transform(reviewer_vectors_real)

current_best = None
current_best_cost = float("inf")

# Run K-Means
for k in range(2, 70, 7):
        kmeans_model = KMeans.train(transformed, k,
            maxIterations = 100, runs = 10)

        cost = kmeans_model.computeCost(transformed)

        if cost < current_best_cost:
                current_best_cost = cost
                current_best = kmeans_model

#current_best.save(sc, "reviews/kmeans_model")

predicted = current_best.predict(transformed)

predicted_clusters = predicted.zip(reviewer_vectors.map(lambda
    x: tuple([x[0]] + [val for val in x[1]])))

predicted_clusters = predicted_clusters.sortByKey()
```

```
cluster_output = predicted_clusters.map(lambda x: str(x[0]) +
    "".join("".join(" {}".format(val)) for val in x[1]) + "\n").
    reduce(lambda a, b: a + b)

result_collection["cluster_output"] = cluster_output

# Write results to local disk
try:
        makedirs("results")
except OSError:
        pass

for key in result_collection:
        with open("results/" + key, "w") as f:
                f.write(str(result_collection[key]))
```