Insert Assignment Title Here
02807 Computational Tools for Big Data

Rasmus Haarslev
Troels Thomsen

September 28. 2015

1 Exercise 1.1

The following pipeline:

1 Deletes all punctuation, commas and quotes from file

2 Translates whitespace to newline

3 Sorts it

4 Counts occurrence of each word

5 Sorts it numerically in reverse (largest number first)

6 Prints the top 10 lines

```
tr -d ",.'" < test | tr ' ' '\n' | sort | uniq -c | sort -n -r | head -n 10
```

2 Exercise 1.2

The following unix script deletes all lines that contains a number with 5 or more digits

```
sed "/[0-9]\{5,\}/d" < test2
```

3 Exercise 1.3

The following pipeline:

1 Translates all tabs into spaces in the shakespeare.txt file

2 Removes all characters satisfying `[ ^ a-zA-Z ]`

3 Translates all spaces to newlines

4 Translates upper case to lower case

5 Sorts the lines

6 Keeps only unique lines

7 Uses dict file as plain string to match on the entire individual lines and print only the lines that don't match anything in dict.

8 counts the lines i.e. the misspelled words.

```
tr '\t' ' ' < shakespeare.txt | sed 's/[^a-zA-Z ]//g' | tr ' ' '\n' | tr A-Z a-z |
   sort | uniq | grep -F -x -v -f dict | wc -l
```

4 Exercise 1.4

We chose to use gbar instead of AWS.

5 Exercise 1.5

Git pull on gbar:

```
gbarlogin1(s152165) $ git pull
remote: Counting objects: 15, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 15 (delta 4), reused 1 (delta 1), pack-reused 2
Unpacking objects: 100% (15/15), done.
From https://github.com/ttsoftware/computational-tools
   1d522e1..407cabb  master     -> origin/master
Merge made by recursive.
 ENyYffaq.txt |   40 ++++++++++++++++++++++++++++++++++++++++
 comm         |    1 +
 matrix3      |    3 +++
 week2.py     |   52 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 week3.py     |   15 +++++++++++++++
 5 files changed, 111 insertions(+), 0 deletions(-)
 create mode 100644 ENyYffaq.txt
 create mode 100644 comm
 create mode 100644 matrix3
 create mode 100644 week2.py
 create mode 100644 week3.py
~/computational-tools
```

For a reference of commits see https://github.com/ttsoftware/computational-tools/commits/master.

6 Exercise 2.1

```
def read_matrix(filename):
    """
    Reads a file and tries to construct a matrix from data in the file

    :param filename: name of file it be read
    :return: list of lists of the numbers in the file
    """
    f = open(filename, 'r')

    # Reads all lines from file into list
    lines = f.readlines()

    matrix_list = []
    for line in lines:
        # Using list comprehension to construct a new list of the numbers
        # contained in each line
        matrix_list.append([float(c) for c in re.split("[, ]", line.rstrip())])
    f.close()
    return matrix_list
```

The `read_matrix` method, when run with the file mentioned in the excercise, returns the following:

```
[[0.0, 1.0, 1.0, 3.0, 0.0], [0.0, 2.0, 3.0, 4.0, 10.0], [8.0, 2.0, 2.0, 0.0, 7.0]]
```

```
def write_matrix(array, filename):
    """
    Interprets a matrix (list of lists) and writes it to filename

    :param array: Matrix (list of lists) to be written to file
    :param filename: Name of the file to write to
    """
    f = open(filename, 'w')
    output = ""
    for l in array:
        for c in l:
            output += str(c) + ' '
        output = output[:-1] + '\n'
    f.write(output)
    f.close()
```

The `write_matrix` method, when run with a random matrix, creates the following file output:

```
2 3 5
1 4 2
```

7 Exercise 2.2

```
def bit_strings(N):
    """
    Takes a number N and constructs all bit permutations of size N

    :param N: Size of permutations
    :return: List of all permutations of bit strings of size N
    """
    pools = [[0, 1]] * N
    result = [[]]
    for pool in pools:
        result = [x+[y] for x in result for y in pool]
    return result
```

For $N = 3$, this method yields the following result:

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
```

8 Exercise 2.3

```
def bag_of_words(filename):
    """
    Reads a very specific json file and constructs a bag-of-words representation
    of people's comments and their ability to get free pizza.

    :param filename: Name of file to be read
    :return: Bag-of-words representation
    """
    f = open(filename)
    lines = f.readlines()
    request_texts = []
    theD = {}
    cnt = 0

    # Regular expression that matches request_text line
    pat = re.compile('\s+"request_text": "(?P<text>.+)"')
    # Regular expression that matches requester_received_pizza line
    patr = re.compile('\s+"requester_received_pizza": (?P<pizza>.+),')
    results = []
    for line in lines:
        # If line is matches request_text
        result = re.search(pat, line)
        if result:
            # Saves the request_text for later use
            request_texts.append(result.group('text').lower())

            # Iterate over all words
            for word in re.findall("[\w\']+", result.group('text')):
                word = word.lower()

                # Put words not already found into dictionary
                if word not in theD.keys():
                    theD[word] = cnt
                    cnt += 1

        # Make list of all the results of pizze beggars.
        pizza_result = re.search(patr, line)
        if pizza_result:
            results.append(0) if pizza_result.group('pizza') == 'false' else results.append(1↩
                )

    # Use word dictionary to create bag-of-words
    bag = []
    for i, text in enumerate(request_texts):
        vec = [0] * len(theD) + [results[i]]
        for word in re.findall("[\w\']+", text):
            vec[theD[word]] += 1
        bag.append(vec)

    return bag
```

9 Exercise 3.1

```
def solve_matrix(filename):
    mat = np.matrix(week2.read_matrix(filename))
    A = np.matrix(mat[:, range(len(mat))])
    last_column = np.array(mat[:, len(mat)])

    return np.linalg.solve(A, last_column)
```

Running this script for the file mentioned in the exercise, we get the following result:

```
[[-5.09090909]
 [ 1.18181818]
 [ 2.24242424]]
```

10 Exercise 3.2

```
 def roots(filename):
    points = np.matrix(week2.read_matrix(filename))

    x = np.asarray(points[:, 0]).squeeze()
    y = np.asarray(points[:, 1]).squeeze()

    z = np.polyfit(x, y, 3)
    f = np.poly1d(z)

    root = newton(f, 0)

    return root
```

Running this script for the file mentioned in the exercise, we get the following result:

```
-1.43463628748
```

11 Exercise 3.3

```
def panda_movie_merge():
    movies = pandas.read_table('week3/movies.dat', sep='::', names=['movie id', 'title', '↩
        genre'])
    ratings = pandas.read_table('week3/ratings.dat', sep='::', names=['user id', 'movie id', ↩
        'rating', 'timestamp'])
    users = pandas.read_table('week3/users.dat', sep='::', names=['user id', 'gender', 'age',↩
         'occupation code', 'zip'])

    return movies.merge(
        ratings.merge(users, on='user id'),
        on='movie id'
    )
```

```
def panda_top_movies(movie_data):
    group_by_movie = movie_data.groupby(['movie id'])
    ratings_by_movie = group_by_movie['rating'].agg({'rating count': sum})

    top_five = ratings_by_movie.sort(columns='rating count', ascending=False).iloc[0:5]
    active_titles = ratings_by_movie[ratings_by_movie['rating count'] >= 250]

    movies_ratings = movie_data.merge(active_titles.reset_index(), on='movie id')
    # filter gender
    female_ratings = movies_ratings[movies_ratings['gender'].isin(['F'])]
    male_ratings = movies_ratings[movies_ratings['gender'].isin(['M'])]

    top_3_female_ratings = (
        female_ratings
        .groupby(['movie id'])['rating']
        .agg(['mean'])
        .sort(columns='mean', ascending=False)
        .iloc[0:3]
    )
    top_3_male_ratings = (
        male_ratings
        .groupby(['movie id'])['rating']
        .agg(['mean'])
        .sort(columns='mean', ascending=False)
        .iloc[0:3]
    )

    gender_means = (
        female_ratings
        .groupby(['movie id'])['rating']
        .agg(['mean'])
        .reset_index()
        .merge(
            (
                male_ratings
                .groupby(['movie id'])['rating']
                .agg(['mean'])
                .reset_index()
            ),
            on='movie id'
        )
    )

    gender_means['mean_diff'] = gender_means['mean_x'] - gender_means['mean_y']
    gender_means_diff_sorted = gender_means.sort(columns='mean_diff', ascending=False)

    top_10_female_mean_diff = gender_means_diff_sorted.iloc[0:10]
    top_10_male_mean_diff = gender_means_diff_sorted.iloc[-10:].sort(columns='mean_diff', ↩
        ascending=True)

    top_5_std_rating = (
        movies_ratings
        .groupby(['movie id'])['rating']
        .agg(['std'])
        .sort(columns='std', ascending=False)
        .iloc[0:5]
    )

    return (
        top_five,
        active_titles,
        top_3_female_ratings,
        top_3_male_ratings,
        top_10_female_mean_diff,
        top_10_male_mean_diff,
        top_5_std_rating
    )
```

With this, we get the following result:

11.1 Top five

```
movie id   rating count
2858              14800
260               13321
1196              12836
1210              11598
2028              11507
```

11.2 Active title

movie id	rating count
1	8613
2	2244
3	1442
4	464
5	890
6	3646
7	1562
9	271
10	3144
11	3919
12	378
13	323
14	542
15	359
16	2587
17	3363
18	524
19	965
20	406
21	4914
22	1266
23	360
24	1984
25	3578
26	353
28	726
29	1637
30	270
31	439
32	5962
34	6814
36	3673
39	4935
41	958
42	634
43	572
44	867
45	1863
46	516
47	4669
48	1137
50	8054
52	1569
57	337
58	2051
60	1147
62	2000
63	317
65	295
69	1166
70	2885
72	338
73	855
74	357
76	508

```
79                297
81                525
82                345
85                660
86                801
                  ...
[2161 rows x 1 columns]
```

11.3 Top 3 female ratings

```
movie id  mean
745       4.644444
1148      4.588235
3022      4.575758
[3 rows x 1 columns]
```

11.4 Top 3 male ratings

```
movie id  mean
2905      4.639344
858       4.583333
2019      4.576628
[3 rows x 1 columns]
```

11.5 Top 10 female difference

```
      movie id    mean\_x    mean\_y  mean\_diff
1129      2084  3.861111  2.666667   1.194444
13          15  3.200000  2.341270   0.858730
579       1088  3.790378  2.959596   0.830782
864       1592  3.057143  2.233766   0.823377
924       1707  2.486486  1.683761   0.802726
818       1460  3.156250  2.435484   0.720766
116        203  3.486842  2.795276   0.691567
1382      2468  3.254717  2.578358   0.676359
299        506  3.862745  3.190476   0.672269
373        650  3.800000  3.136364   0.663636
[10 rows x 4 columns]
```

11.6 Top 10 male difference

```
      movie id    mean\_x    mean\_y  mean\_diff
2019      3658  2.900000  3.779661  -0.879661
1934      3487  3.000000  3.765625  -0.765625
1315      2377  2.250000  2.994152  -0.744152
781       1382  2.100000  2.837607  -0.737607
1696      3036  2.578947  3.309677  -0.730730
638       1201  3.494949  4.221300  -0.726351
298        504  2.300000  2.994048  -0.694048
2079      3760  2.878788  3.555147  -0.676359
1194      2165  2.888889  3.536585  -0.647696
1713      3066  3.090909  3.737705  -0.646796
[10 rows x 4 columns]
```

11.7 Top 5 standard deviation

```
movie id  std
1924      1.455998
2314      1.372813
3864      1.364700
2459      1.332448
231       1.321333
[5 rows x 1 columns]
```

12 Exercise 3.4

```
def bag_prediction(bag):
    training_data = bag[:-int(math.floor(len(bag)*0.1))]
    test_data = bag[-int(math.ceil(len(bag)*0.1)):]
    dataset = [row[:-1] for row in training_data]
    t_vec = [row[-1] for row in training_data]

    classifier = LogisticRegression().fit(dataset, t_vec)

    test_dataset = [row[:-1] for row in test_data]
    test_t_vec = [row[-1] for row in test_data]

    return classifier.predict(test_dataset), classifier.score(test_dataset, test_t_vec)
```

With this, we get a prediction accuracy if `0.67258883248730961`, which is pretty good considering, that the data is sometimes extremely different, which means it is very hard to predict. If we also take into consideration, that the time of posting may also have a say in whether or not the poster gets pizza, which isn't taken into consideration in the classifier, an accuracy of around 67% is actually very good.

13 Exercise 3.5

```
import time


def sum_quadrant():
    quadrant_list = []
    for i in range(1, 10000):
        quadrant_list += [(1/(i**2))]
    return sum(quadrant_list)

def run():
    start_time = time.time()
    for i in range(500):
        sum_quadrant()
    print("It took %s seconds to compute the sum 500 times." % (time.time() - start_time))


if "__main__" == __name__:
    run()
```

```
python week3_cython.pyx It took 0.771023988724 seconds to compute the sum 500 times.
```

```
python week3_cython_run.py It took 0.586192846298 seconds to compute the sum 500 times.
```

We notice the reduce the running time by approximately 0.18 seconds or approximately 20%.

14 Exercise 4.1

The following is the code used to calculate the clusters.

```python
import os
import pickle
from scipy.sparse import csr_matrix
from scipy.spatial.distance import jaccard, pdist, squareform
import time


def scan(filename, epsilon, min_size):

    pkl_file = open(filename, 'rb')
    data = pickle.load(pkl_file)
    dataset = [{
        "data": x,
        "visited": False,
        "in_cluster": False,
        "is_noise": False
    } for x in squareform(pdist(csr_matrix(data).todense(), 'jaccard'))]

    clusters = []

    for datapoint in dataset:
        if datapoint['visited']:
            continue

        datapoint['visited'] = True
        neighbour_points = region_query(dataset, datapoint, epsilon)

        if len(neighbour_points) < min_size:
            datapoint['is_noise'] = True
        else:
            cluster = expand_cluster(dataset, datapoint, neighbour_points, epsilon, min_size)
            clusters.append(cluster)

    noise_cluster = {'datapoints': [], 'is_noise': True}
    for datapoint in dataset:
        if datapoint['is_noise']:
            noise_cluster['datapoints'].append(datapoint)

    clusters.append(noise_cluster)
    return clusters


def expand_cluster(dataset, datapoint, neighbour_points, epsilon, min_size):
    """
    Expands cluster with epsilon-neighbouring datapoints not belonging to an existing cluster
    :param datapoint:
    :param neighbour_points:
    :param cluster:
    """
    cluster = {'datapoints': [datapoint], 'is_noise': False}
    datapoint['in_cluster'] = True

    for new_datapoint in neighbour_points:

        if not new_datapoint['visited']:

            new_datapoint['visited'] = True
            new_neighbour_points = region_query(dataset, new_datapoint, epsilon)

            if len(new_neighbour_points) >= min_size:
                neighbour_points += new_neighbour_points

        if not new_datapoint['in_cluster']:
            cluster['datapoints'].append(new_datapoint)
            new_datapoint['is_noise'] = False
            new_datapoint['in_cluster'] = True

    return cluster


def region_query(dataset, datapoint, epsilon):
    """
    Returns a list of new datapoints in datapoint's epsilon-neigbourhood
    :param datapoint:
    :return:
    """
    neighbours = []
    for i, distance in enumerate(datapoint['data']):
        if distance <= epsilon:
            neighbours.append(dataset[i])
    return neighbours

if "__main__" == __name__:
    start_time = time.time()
    print len(scan(os.path.dirname(__file__) + "/../week4/test_files/data_10000points_10000↩
        dims.dat", 0.15, 2))
    print("--- %s seconds ---" % (time.time() - start_time))
```

We start by building a dataset containing all datapoints and their inter-jaccard-distances, which we can later look up in the region query.

The rest of the implementation follows the pseudo code closely.

Our results are as follows:

```
10 dimension-set: 0.000533103942871 seconds
100 dimension-set: 0.00982999801636 seconds
1000 dimension-set: 1.14800882339 seconds
10000 dimension-set: 1059.263767 seconds
100000 dimension-set: <never finished>
```