PES University, Bangalore

UE16CS322 - Data Analytics   Session: Aug – Dec 2018

Assignment 5 - Worksheet

Date of Submission: 27th September 2018 **Max Marks:** 20

NOTE: *This assignment is to be done in Python. Each team has to make a well-documented ipython notebook and submit it, along with an additional text file containing the code.*

TOPIC: Collaborative Filtering

Virtually everyone has had an online experience where a website makes personalized recommendations in hopes of future sales or ongoing traffic. Amazon tells you "Customers Who Bought This Item Also Bought", Udemy tells you "Students Who Viewed This Course Also Viewed". And Netflix awarded a $1 million prize to a developer team in 2009, for an algorithm that increased the accuracy of the company's recommendation system by 10 percent.

Source: *https://datascienceplus.com/*

Recommendation Systems have become such a common part of our everyday lives, and there are a number of methods and techniques involved in building them, Collaborative Filtering being one of the main ones.

The dataset can be found here as 'train.csv' and 'test.csv'.

Note:

This assignment uses the numpy package. Learn how numpy arrays work here.

The worksheet has hints and code snippets attached to every question, which tells you what code to use for that part of the question. These code snippets must be used in your code.

For the comments that start with an asterisk (*), those sections of the code have to be filled in by you.

Question 1 (12 points)

Read the files into dataframes using Pandas. (Hint: Reading from a CSV file)

a). Find the average rating given by each user.

b). Find the average rating given for each item.

(**Hint**: Use the groupby and aggregate functions of Python.
E.g. `df.groupby('key').aggregate(max)`
More information can be found here)

c). Write a function to compute the Root Mean Squared Error.

(**Hint**: Writing a function in Python.
Use `np.mean_squared_error()`)

Calculate the RMSE if we assign overall average rating to each user/item pair.

d). Convert the train and the test data into a matrix. These matrices, in this format, will be used as input to the functions defined for further predictions.
Also calculate the Sparsity of these matrices.

Hints:

The code to convert to a matrix:

```
#data - dataframe

rows = data.user_id.unique()
cols = data['book'].unique()
data = data[['user_id', 'book', 'book_rating']]

idict = dict(zip(cols, range(len(cols))))
udict = dict(zip(rows, range(len(rows))))

data.user_id = [ udict[i] for i in data.user_id ]
data['book'] = [ idict[i] for i in data['book'] ]

nmat = data.as_matrix()
```

e). Write a function to predict in the following manner:

For any (user i, item j), assign it with:
(mean rating given by user i) + (mean rating received by item j) – (average rating over the entire dataset)

i.e., umean[i] + imean[j] – amean

Calculate the RMSE for these predictions.

Hints:

```
#basic function
def predict_naive(user, item):
    prediction = imean1[item] + umean1[user] - amean1
    return prediction
```

To get an imean array and a umean array from the matrix:

```
naive = np.zeros((len(rows),len(cols)))
for row in nmat:
    naive[row[0], row[1]] = row[2]

amean1 = np.mean(naive[naive!=0])
umean1 = sum(naive.T) / sum((naive!=0).T)
imean1 = sum(naive) / sum((naive!=0))
```

Make sure to remove NA values from the imean and umean arrays.

To calculate the RMSE:

Create a predictions array and a targets array.

Perform the following for each row in the matrix:

```
    user, item, actual = row[0], row[1], row[2]
    predictions.append(predict_naive(user, item))
    targets.append(actual)
```

Question 2 (8 points)

On the test file:

a). For a user x:

Find k similar users using cosine similarity and Pearson correlation. Use k=5

Hints:

These functions are used to get the similarity based on cosine similarity and Pearson correlation.

```
def cos(mat, a, b):
    if a == b:
        return 1
    aval = mat.T[a].nonzero()
    bval = mat.T[b].nonzero()
    corated = np.intersect1d(aval, bval)
    if len(corated) == 0:
        return 0
    avec = np.take(mat.T[a], corated)
    bvec = np.take(mat.T[b], corated)
    val = 1 - cosine(avec, bvec)
    if np.isnan(val):
        return 0
    return val

def pr(mat, a, b, imean):
    if a == b:
        return 1
    aval = mat.T[a].nonzero()
    bval = mat.T[b].nonzero()
    corated = np.intersect1d(aval, bval)
    if len(corated) < 2:
        return 0
    avec = np.take(mat.T[a], corated)
    bvec = np.take(mat.T[b], corated)
    avec1 = avec - imean[a]
    bvec1 = bvec - imean[b]
    val = 1 - cosine(avec1, bvec1)
    if np.isnan(val):
        return 0
    return val

def itemsimilar(mat, option):
    # *Calculate amean, umean and imean as before

    n = mat.shape[1]
    # *initialize a zero matrix with dimensions n,n to get the similarity matrix

    if option == 'pr':
        #print("PR")
        for i in range(n):
            for j in range(n):
                sim_mat[i][j] = pr(mat, i, j, imean)
        sim_mat = (sim_mat + 1)/2
    elif option == 'cos':
        #print("COS")
        for i in range(n):
            for j in range(n):
                sim_mat[i][j] = cos(mat, i, j)
    else:
        #print("Default")
        sim_mat = cosine_similarity(mat.T)

    return sim_mat, amean, umean, imean
```

b). Write a function to predict the rating jor the user x and item j, based on K similar user to x.

Hints:

```
def predict(user, item, mat, item_similarity, amean, umean, imean,  k=20):
    nzero = mat[user].nonzero()[0]
    if len(nzero) == 0:
        return amean
    baseline = imean + umean[user] - amean
    choice = nzero[item_similarity[item, nzero].argsort()[::-1][:k]]

    prediction = ((mat[user, choice] - baseline[choice]).dot(item_similarity[item,
choice])/ sum(item_similarity[item, choice])) + baseline[item]
```

```
    if np.isnan(prediction):
        prediction = amean
    if prediction > 10:
        prediction = 10
    if prediction < 1:
        prediction = 1
    return prediction
```

c). Report the train and test RMSE errors for these predictions.

Hints:

```
def get_results(train_data, test_data,  option, rows, cols, k):
    # *initialize a zero matrix called full_mat with dim(rows,cols)
    for row in train_data:
        full_mat[row[0], row[1]] = row[2]

    item_similarity, amean, umean, imean = itemsimilar(full_mat, option)

    # *initialize empty preds and real lists.

    # *For each row in train_data, call the predict function using the above
values as parameters, as explained before for the naïve prediction. Keep updating
the preds and real lists.


    # *Calculate the RMSE error of both lists.

    print('Train Error')
    print('RMSE : %.4f' % err1)

    # *Reinitialize the preds and real lists to empty lists

    # *For each row in the test_data, call the predict function using the above
values as parameters, as explained before for the naïve prediction. Keep updating
the preds and real lists.

    # *Calculate the RMSE error of both lists.

    print('Test Error')
    print('RMSE : %.4f' % err2)
```

d). How do you find the optimal value of k while using kNN?