CAPSTONE PROJECT
DATA SCIENTST NANODEGREE

Khaled Al-Jaghwani

16/12/18

DEFINITION

1. Project Overview

Recommending the questions that a programmer should solve given his/her current expertise is a big challenge for Online Judge Platforms but is an essential task to keep a programmer engaged on their platform.

In this practice problem, I have used the data of programmers and questions that they have previously solved along with the time that they took to solve that particular question.

As a data scientist, my task was to build a model that can predict the time taken to solve a problem given the user current status.

About Online Judge Platforms

The *SPOJ platform* is centered around an online judge system, which serves for the automatic assessment of user-submitted programs. More than 2400 contests hosted as of 2012, ranging from instant challenges to long-term e-learning courses.

About Analytics Vidhya

Analytics Vidhya provides a community based knowledge portal for Analytics and Data Science professionals. The aim of the platform is to become a complete portal serving all knowledge and career needs of Data Science Professionals and on their resources is **Hackathons** – Real life industry problems being released in form of contests, and this project is one of them.

Data

A collected user submission for various problems from an online judge platform. The submissions data consists of submissions of 3571 users and 6544 problems.

There are three training data files:

1- **train_submissions.csv** - This contains 155295 submissions, which are selected randomly from 221850 submissions.

2- **user_data.csv** - This is the file containing data of users.

3- **problem_data.csv** - This is the file containing data of the problems.

There are two testing data files:

1- **test_submissions.csv** - This contains the remaining 66555 submissions from total 221850 submissions.

2- **sample_submissions.csv** - This contains the predictions of the model on the test_submissions.csv file.

2. Problem Statement

Online judges provide a platform where many users solve problems every day to improve their programming skills. The users can be beginners or experts in competitive programming. Some users might be good at solving specific category of problems (e.g. Greedy, Graph algorithms, Dynamic Programming etc.) while others may be beginners in the same. There can be patterns to everything, and the goal of the machine learning would be to identify these patterns and model user's behavior from these patterns.

The goal of this challenge is to predict range of attempts a user will make to solve a given problem given user and problem details. Finding these patterns can help the programming committee, as it will help them to suggest relevant problems to solve and provide hints automatically on which users can get stuck.

3. Metrics

The metrics used for evaluating the performance of the model is the F1 score between the predicted and the actual value with average parameter set to 'weighted'. This matric will implement to 66555 record.

ANALYSIS

4. Data Exploration

In order to understanding the data, we can see the most important dataset we need to explore the submissions dataset.

Train / test submissions.csv

As showing in Figure 2 that the train dataset is contains 3 columns:

1- 'user_id' the ID that assigned to each user.

2- 'problem_id' the ID that assigned to each problem.

3 - 'attempts_range' denoted the range no. in which attempts the user made to get the solution accepted lies.

Define the attempts_range :-

attempts_range inside	No. of attempts lies
1	1-1
2	2-3
3	4-5
4	6-7
5	8-9

	user_id	problem_id	attempts_range
0	user_232	prob_6507	1
1	user_3568	prob_2994	3
2	user_1600	prob_5071	1
3	user_2256	prob_703	1
4	user_2321	prob_356	1

Figure 1submission dataset

Number of nans is user_id 0 problem_id 0
attempts_range 0

Number of record is 155295

We can see in test submissions Figure 3, that it is contains column (ID) "where ID is the concatenation of user_id and problem_id", user_id, and problem_id. The 'attempts_range' column is to be predicted.

Number of nans is: ID 0 user_id 0 problem_id 0

Number of record is: 66555

	ID	user_id	problem_id
0	user_856_prob_5822	user_856	prob_5822
1	user_2642_prob_2334	user_2642	prob_2334
2	user_2557_prob_2920	user_2557	prob_2920
3	user_1572_prob_4598	user_1572	prob_4598
4	user_295_prob_6139	user_295	prob_6139

Figure 2 test submission dataset

problem.csv

We can see in Figure 4 that it is contains 4 columns

1- problem_id - unique ID assigned to each problem

2- level_id - the difficulty level of the problem between 'A' to 'N' "categorical"

3- points - amount of points for the problem "numerical"

4- tags - problem tag(s) like greedy, graphs, DFS etc. "text analysis"

	problem_id	level_type	points	tags
0	prob_3649	H	NaN	NaN
1	prob_6191	A	NaN	NaN
2	prob_2020	F	NaN	NaN
3	prob_313	A	500.0	greedy,implementation
4	prob_101	A	500.0	constructive algorithms,greedy,math

Figure 3 problem dataset

Number of nans is:	
problem_id	0
level_type	133
points	3917
tags	3484

Number of record is: 6544

We can see that we have half the problems without points or tags.

user.csv

It contains the following features=-

user_id - unique ID assigned to each user
submission_count - total number of user submissions
problem_solved - total number of accepted user submissions
contribution - user contribution to the judge
country - location of user
follower_count - amount of users who have this user in followers
last_online_time_seconds - time when user was last seen online
max_rating - maximum rating of user
rating - rating of user

user_id	submission_count	problem_solved	contribution	country	follower_count
user_3311	47	40	0	NaN	4
user_3028	63	52	0	India	17
user_2268	226	203	-8	Egypt	24
user_480	611	490	1	Ukraine	94
user_650	504	479	12	Russia	4

last_online_time_seconds	max_rating	rating	rank	registration_time_seconds
1504111645	348.337	330.849	intermediate	1466686436
1498998165	405.677	339.450	intermediate	1441893325
1505566052	307.339	284.404	beginner	1454267603
1505257499	525.803	471.330	advanced	1350720417
1496613433	548.739	486.525	advanced	1395560498

Figure 4 user dataset

rank - can be one of ‘beginner’ ,’intermediate’ , ‘advanced’, ‘expert’
registration_time_seconds - time when user was registered

Number of record is: 3571

We can see that there a missing values in country column.

Number of nans is:	
user_id	0
submission_count	0
problem_solved	0
contribution	0
country	1153
follower_count	0
last_online_time_seconds	0
max_rating	0
rating	0
rank	0
registration_time_seconds	0

5. Data Visualization

Submissions DataSets Analysis

In order to understanding our submission dataset we want to now information about number of users and problems and which problem/users are not exist in test dataset.

And as look like in table in right is there a 12 users we don't have any information about how they interacted with problems we have a 436 problems without any interact.

Number of problems in test data are = 4716
Number of problems in train data are = 5776
Number of users in test data are = 3501
Number of users in train data are = 3529
Users who are in test data but they are not in training data is = 12
Problems who are in test data but they are not in training data is = 436

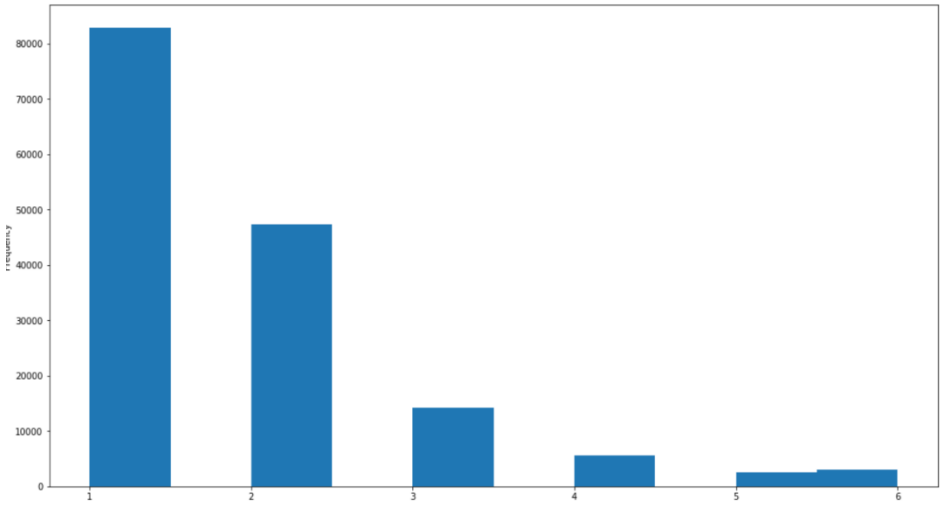


*Figure 5 submissions * attempts_range*

And as the Figure 6, 80K of problems are solved by 1 attempt, and it is reduced by half for other ranges.

As Figure 6 only few users who have solved more than 60 problems in our submissions dataset and most of them are solve between 20 and 60.

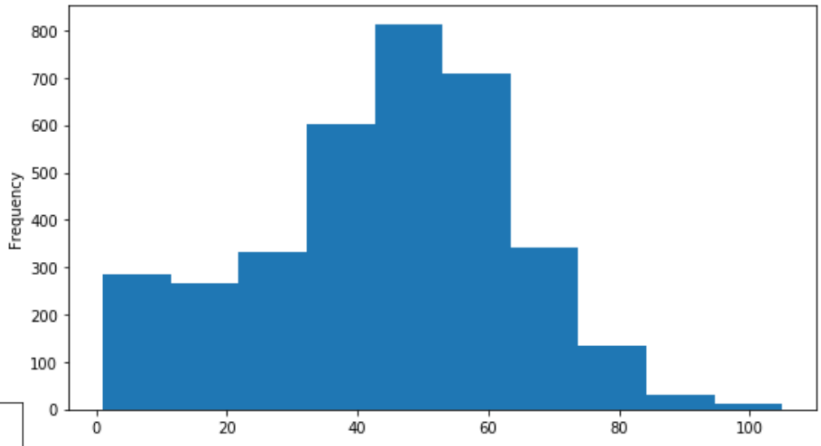


*Figure 6 submissions * user_id*

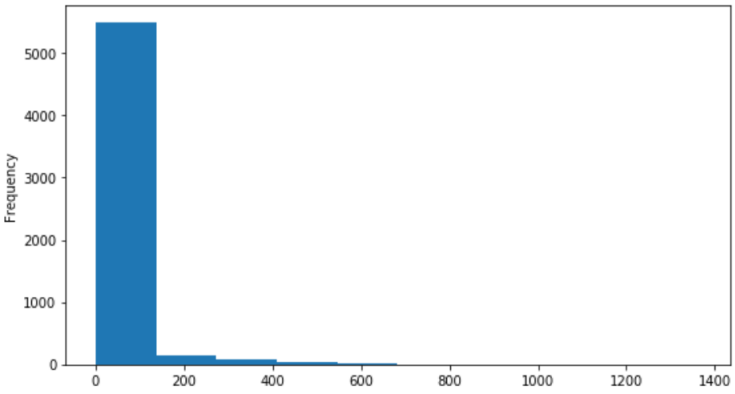


*Figure 7 submissions * problem_id*

We can see here in Figure 7 there some famous problems that have more than 100 solver until to 1400, but most of them are between 1 to 150.

Problem Dataset Analysis

We can see in Figure 8 the most of the problems are from level type A and only very few problems from type M and N.

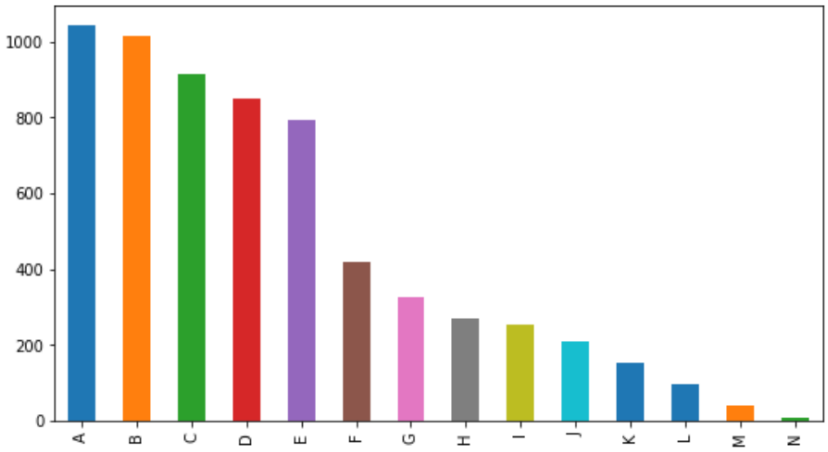


*Figure 8 problem id * level type*

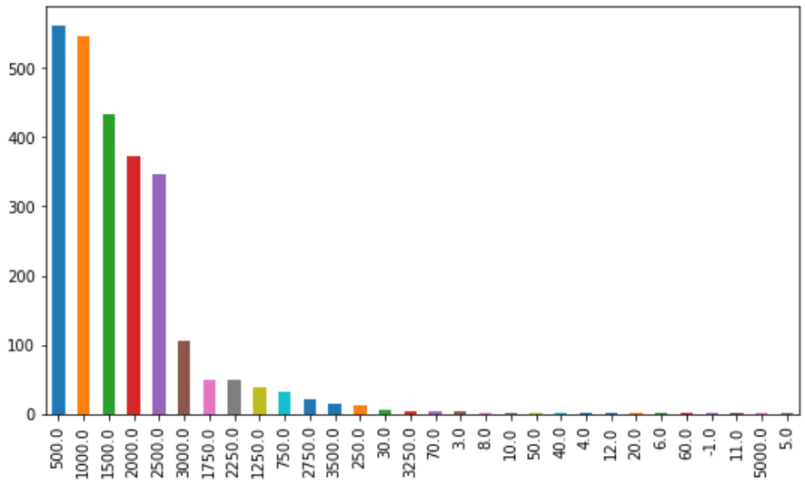


*Figure 9 problem id * points*

And regarding the points the 500&100 they are the most values in problem dataset, also there value like -1 is look as outlier.

As Figure 10 we can see that there many Nan's in problem dataset for the feature tags, we can see that appear here also is showing the tag implementation is the first word frequency in our dataset.

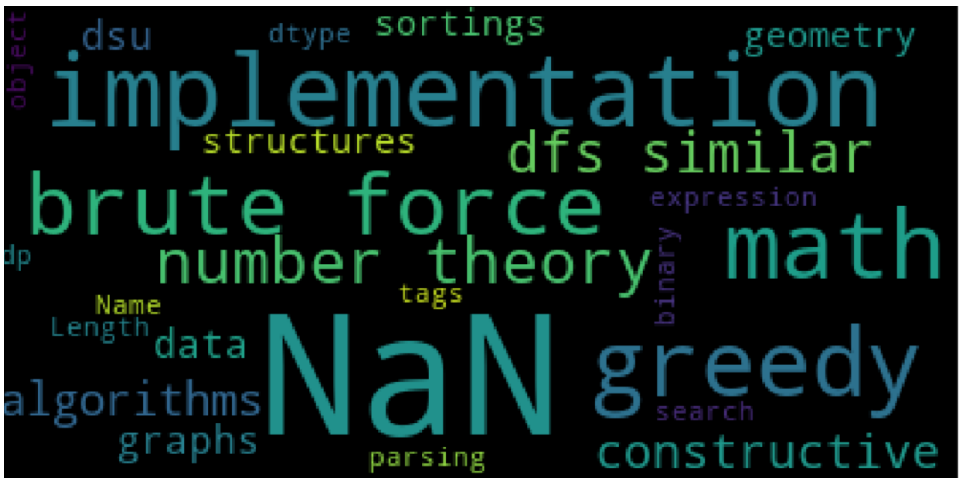


*Figure 10 porblem id * tags*

User Dataset Analysis

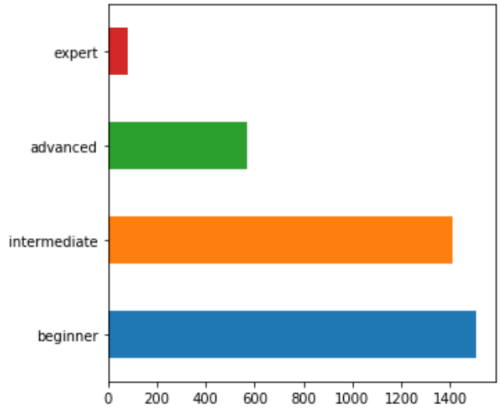


*Figure 11User*rank*

For the feature Rank in Users dataset, as Figure 11 we can see in our dataset the most of the users are beginner.

METHODOLOGY

Matrix Factorization with SVD as a technique for making recommendations. Traditional singular value decomposition a technique can be used when your matrices have no missing values. In this decomposition technique, a user-item

(A) can be decomposed as follows: $A = U\Sigma V^T$

Where:

- U gives information about how users are related to latent features.
- Σ gives information about how much latent features matter towards recreating the user-item matrix.
- V^T gives information about how much each movie is related to latent features.

In addition, in our case we have used FunkSVD was a new method that found to be a useful for matrices with missing values. The matrix factorization you decomposed a user-item (A) as follows: $A = UV^T$

Where:

- U gives information about how users are related to latent features.
- V^T gives information about how much each movie is related to latent features.

I have iterate to find the latent features in each of these matrices using gradient descent. Moreover, I have wrote a function to implement gradient descent to find the values within these two matrices.

Using this method, I were able to make a prediction for any user-problem pair in our dataset.

Collaborative filtering using FunkSVD still wasn't helpful for new users and new problem. In order to recommend these items, I have implemented Neighborhood Based Collaborative Filtering and ranked based recommendations along with your FunkSVD implementation a techniques to assess how similar or distant two users/problems were from one another.

Therefore, to reach this one I have reach to design three algorithms to help me to reach to recommendations, as follows:

* Use FunkSVD_test to use the subset (sample) to define the best parameters to use to refinement (Evaluation and Validation) our final model

* Use FunkSVD to use the best parameters we found it in to train all submission data.

* Use find_Similarity to find the similar every pair of users/problems in case we don't have any information about the attempts.

6. Data Preprocessing

Now we have understand the data and since we have many of features need to some processing to help us to solve our problem to predict how many attempts for the users by finding patterns between current data of submissions, users, and problem.

Preprocessing include make the categorical data to dummies, normalize the data, and fix missing values, so I have created some functions that help us in that as showing below:

I have create dummies dataset to help on changing the type of categorical features as (rank, country, level type)

```
def dummies (data_frame, data_col, weight = 1):

    dummy = pd.get_dummies(data_frame[data_col])
    data_frame.drop(columns = data_col, inplace = True)
    data_frame = data_frame.join(dummy * weight)

    return data_frame
```

Also I have created normalization function to help to normalize numeric features like ('submission_count', problem_solved', 'contribution', 'follower_count', 'max_rating', 'rating', 'last_online_time_seconds', registration_time_seconds', 'attempts_range', 'points')

```
def normalization (data_ser, data_frame, data_col, weight = 1):

    normal_points = data_ser.values.astype(float)
    minmaxscaler = preprocessing.MinMaxScaler()
    normalized = pd.DataFrame(minmaxscaler.fit_transform(normal_points), columns = [data_col])
    data_frame = data_frame.drop(columns = data_col)
    data_frame = data_frame.join(normalized * weight)
    return data_frame
```

Moreover, for missing values I have used a function that help to solve these problems in the data.

```
def predict_missing(df, na_cols, cols):


    y_tri_ids = df[na_cols].notnull().all(axis=1)
    y_pred_ids = df[na_cols].isnull().all(axis=1)
    df_trin = df[y_tri_ids]
    df_pred = df[y_pred_ids]
    model = MultiOutputRegressor(GradientBoostingRegressor(learning_rate=0.01, _estimators=1000,
subsample=.80,))
    model.fit(df_trin[cols], df_trin[na_cols])
    df_pred[na_cols] = model.predict(df_pred[cols])


    return df_pred, y_pred_ids
```

When I have complete all this processing in the data, I have created a two matrix using following datasets (submission, user, and problem).

- The first matrix was problem dataset that will used as the list that will help us to know which two problem are similar to each other.
- The second matrix was user dataset that will used as the list that will help us to know which two user are similar to each other.

Then I have start to create another two matrix will use to train the model this two matrix will use only submission data then we will use a FunkSVD, one of this matrix will use for validation purpose that will help us to know the best parameters will use.

In this matrix, we subset the top 100 problems in submissions then we will choose the top 100 users in this subset, in order to create a matrix 100*100. Then we have create a sample from this new subset to use it as validation.

```
parmeter_matrix = training.groupby(['user_id', 'problem_id'])['attempts_range'].max().unstack()
```

In addition, I have created a matrix that contain all submission data that will used latter to train our Model.

```
user_prob_matrix = submissions.groupby(['user_id', 'problem_id'])['attempts_range'].max().unstack()
```

Implementation

```
def FunkSVD_test(attempts_mat, val_df, latent_features, learning_rate, iters):
    '''
    This function performs matrix factorization using a basic form of FunkSVD with no regularization

    INPUT:
    attempts_mat - (numpy array) a matrix with users as rows, problems as columns, and attempts as values
    latent_features - (list) of number of latent features
    learning_rate - (list) of learning rate s
    iters - (list) of number of iterations

    OUTPUT:
    track_dict - defaultdict: a dict that have all the iterations that happend in the function
    '''

    # Set up useful values to be used through the rest of the function

    n_users = attempts_mat.shape[0]
    n_problems = attempts_mat.shape[1]
    num_attempted = np.count_nonzero(~np.isnan(attempts_mat))
    track_dict = defaultdict(list)

    # keep track of iteration and rmse

    print("Optimizaiton Statistics")
    print("Iterations | RMSE Training | RMSE Vaildation | Learning rate | Latent features | SUM RMSE'S")

    for lf in latent_features:
        for lr in learning_rate:
            for it in iters:

                # initialize the user and problem matrices with random values
                user_mat = np.random.rand(n_users, lf)
                problem_mat = np.random.rand(lf, n_problems)

                # initialize sse at 0 for first iteration
```

```
            sse_accum = 0

            # for each iteration

            for iteration in range(it):

                # update our sse

                old_sse = sse_accum
                sse_accum = 0

                # For each user-problem pair

                for i in range(n_users):
                    for j in range(n_problems):

                        # if the attempts exists

                        if attempts_mat[i, j] > 0:

                            # compute the error as the actual minus the dot product of the user and problem latent
features
                            diff = attempts_mat[i, j] - np.dot(user_mat[i, :], problem_mat[:, j])

                            # Keep track of the sum of squared errors for the matrix
                            sse_accum += diff**2

                            # update the values in each matrix in the direction of the gradient
                            for k in range(lf):
                                user_mat[i, k] += lr * (2*diff*problem_mat[k, j])
                                problem_mat[k, j] += lr * (2*diff*user_mat[i, k])

            # Validation
            train_rmse = np.sqrt (sse_accum / num_attempted)
            valid_rmse = validation_comparison (val_df, user_mat, problem_mat)

            # print results
            track_dict['it'].append(it)
            track_dict['train_rmse'].append(train_rmse)
```

```
            track_dict['valid_rmse'].append(valid_rmse)
            track_dict['lr'].append(lr)
            track_dict['lf'].append(lf)
            track_dict['train_rmse_valid_rmse'].append(train_rmse + valid_rmse)

            print("\t %d \t %f \t %f \t %f \t %d \t \t %f" % (it, train_rmse, valid_rmse, lr, lf, train_rmse +
valid_rmse))

    return track_dict
```

The FunkSVD_test is used to find the best parameters to use it for training all our submission data.

```
train_data_np = np.array(parmeter_matrix)
track_dict = FunkSVD_test(train_data_np, validtion, [28, 25, 23], [0.001,0.0015, 0.002], [200, 225, 250])
```

We have push our validation matrix with the subset we have used into a funkSVD, we have used [28, 25, 23] as latent features and used [0.001,0.0015, 0.002], as learning rate, and 200,225,250 as iterations, then we have used the best parameters we found as a sum of RMSE between actual and predicted result .

```
def validation_comparison(val_df, user_mat, problem_mat):

    '''
    This function will use to validate between actual and predicted values

    INPUT:
    val_df - the validation dataset
    user_mat - U matrix in FunkSVD
    problem_mat - V matrix in FunkSVD

    OUTPUT:
    rmse - RMSE of how far off each value is from it's predicted value
    '''

    val_users = np.array(val_df['user_id'])
    val_problems = np.array(val_df['problem_id'])
    val_attempts_range = np.array(val_df['attempts_range'])

    sse = 0
    num_attempted = 0
    preds = []

    for idx in range(len(val_users)):

        pred = predict_attempts(user_mat, problem_mat, parmeter_matrix, val_users[idx], val_problems[idx])
        sse += (val_attempts_range[idx] - pred)**2
        num_attempted+=1

    rmse = np.sqrt(sse/num_attempted)

    return rmse
```

And we have validation_comparison to calculated the difference between each predicted value and real values, that call push predict_attempts the user mat, problem mat with the user to find the predicted values.

```
def predict_attempts(user_matrix, problem_matrix, user_prob_matrix, user_id, problem_id):
    '''

    This function will help to predict
    INPUT:
    user_matrix - user by latent factor matrix
    problem_matrix - latent factor by problem matrix
    user_id - the user_id from the reviews df
    problem_id - the problem_id according the problems df

    OUTPUT:
    pred - the predicted attempts for user_id-problem_id according to FunkSVD
    '''
    # Use the training data to create a series of users and problems that matches the ordering in training
data
    user_ids_series = np.array(user_prob_matrix.index)
    problem_ids_series = np.array(user_prob_matrix.columns)

    # User row and problem Column
    user_row = np.where(user_ids_series == user_id)[0][0]
    problem_col = np.where(problem_ids_series == problem_id)[0][0]

    # Take dot product of that row and column in U and V to make prediction
    pred = np.dot(user_matrix[user_row, :], problem_matrix[:, problem_col])

    return pred
```

In addition, this predict_attempts dot the two matrix based on problem and user id's to find which is the right value.

```
def find_similar_users(user_id, user_item):
    '''
    Computes the similarity of every pair of users or problems based on the dot product Returns an
ordered


    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users or problems features


    OUTPUT:
    similar_users - (list) an ordered list where the closest users/problem (largest dot product users)
                    are listed first
    '''


    user_range = list(user_item.index)
    user_range.remove(user_id)
    user_1 = user_item.loc[user_id]
    dot_similarity = []
    for user in user_range:
        user_2 = user_item.loc[user]
        dot_similarity.append((np.dot(user_1, user_2), user))


    # sort by similarity
    dot_similarity.sort(key=lambda tup: tup[0], reverse=True)
    # create list of just the ids
    most_similar_users = [x[1] for x in dot_similarity]



    return most_similar_users, dot_similarity
```

We also used for the users and problems that don't have any submission information's we have used find_similar_users to know to computes the similarity of every pair of users or problems based on the dot product Returns an ordered.

```
def FunkSVD(attempts_mat, latent_features, learning_rate, iters):
    '''
    This function performs matrix factorization using a basic form of FunkSVD with no regularization

    INPUT:
    attempts_mat - (numpy array) a matrix with users as rows, problems as columns, and attempts as values
    latent_features - (list) of number of latent features
    learning_rate - (list) of learning rate s
    iters - (list) of number of iterations

    OUTPUT:
    track_dict - defaultdict: a dict that have all the iterations that happend in the function
    '''



    # Set up useful values to be used through the rest of the function
    n_users = attempts_mat.shape[0]
    n_problems = attempts_mat.shape[1]
    num_attempts = np.count_nonzero(~np.isnan(attempts_mat))

    # initialize the user and problem matrices with random values
    user_mat = np.random.rand(n_users, latent_features)
    problem_mat = np.random.rand(latent_features, n_problems)

    # initialize sse at 0 for first iteration
    sse_accum = 0

    # keep track of iteration and MSE
    print("Optimizaiton Statistics")
    print("Iterations | Mean Squared Error | learning_rate | diff")

    # for each iteration
    for iteration in range(iters):

        # update our sse

        old_sse = sse_accum
        sse_accum = 0
```

```
        # For each user-problem pair
        for i in range(n_users):
            for j in range(n_problems):

                # if the attempts exists
                if attempts_mat[i, j] > 0:

                    # compute the error as the actual minus the dot product of the user and problem latent
features
                    diff = attempts_mat[i, j] - np.dot(user_mat[i, :], problem_mat[:, j])

                    # Keep track of the sum of squared errors for the matrix
                    sse_accum += diff**2

                    # update the values in each matrix in the direction of the gradient
                    for k in range(latent_features):
                        user_mat[i, k] += learning_rate * (2*diff*problem_mat[k, j])
                        problem_mat[k, j] += learning_rate * (2*diff*user_mat[i, k])

        # print results
        print("%d \t\t %f \t %f \t %f" % (iteration+1, sse_accum / num_attempts, learning_rate, diff))

    return user_mat, problem_mat
```

I have implement FunkSVD to train the final model that we will use its user mat and problem mat to predict our the items that we care about, which are used Mean Squared Error to compare between predicted value with the train values without looking to validation dataset.

```
# For each item in test_submission

for i in range(test_submissions.shape[0]):

    # for each user and problem

    user_pr = test_submissions['user_id'][i]
    prob_pr = test_submissions['problem_id'][i]

    # check if the user/problem exist on submissions to see which is the prediction for it

    if user_pr in user_prob_matrix.index and prob_pr in user_prob_matrix.columns:
        val = predict_attempts(user_mat, problem_mat, user_pr, prob_pr)
        test_submissions['attempts_range'].loc[i] = val

    # if not we will see closest user/ problem and check if exist on submissions data



    else:
        if prob_pr not in user_prob_matrix.columns and user_pr in user_prob_matrix.index:
            new_prob = find_similar_users(prob_pr, mat_prop)[0]
            for pro in new_prob:
                if pro in user_prob_matrix.columns:
                    val = predict_attempts(user_mat, problem_mat, user_pr, pro)
                    test_submissions['attempts_range'].loc[i] = val
                    break
        else:
            if prob_pr in user_prob_matrix.columns and user_pr not in user_prob_matrix.index:
                new_usrs = find_similar_users(user_pr, users1_matrix)[0]
                for usr in new_usrs:
                    if usr in user_prob_matrix.index:
                        val = predict_attempts(user_mat, problem_mat, usr, prob_pr)
                        test_submissions['attempts_range'].loc[i] = val
                        break
            else:
                if prob_pr not in user_prob_matrix.columns and user_pr not in user_prob_matrix.index:
                    new_prob = find_similar_users(prob_pr, mat_prop)[0]
                    new_usrs = find_similar_users(user_pr, users1_matrix)[0]
                    for usr in new_usrs:
```

```
            for pro in new_prob:
                if pro in user_prob_matrix.columns and usr in user_prob_matrix.index:
                    val = predict_attempts(user_mat, problem_mat, usr, pro)
                    test_submissions['attempts_range'].loc[i] = val
                    break
```

Finally, I have implemented this algorithm that will predict for each value test_submissions if it's exist in our user_item matrix, if the problems exist will use our predict_attempts with them same, and if not will search for similar users in find_similar_users and predict the simialt if exist using predict_attempts function.

RESULTS

Model Evaluation and Validation

When we have try the following parameters [28, 25, 23] as latent features and used [0.001,0.0015, 0.002], as learning rate, and 200,225,250 as iterations.

We reach to following best parameters I found are:

```
(, latent_features=23, learning_rate=0.002, iters=225)
```

As they are fit our data:

Optimizaiton Statistics

Iterations	RMSE Training	RMSE Vaildation	Learning rate	Latent features	SUM RMSE'S
250	0.176013	0.793358	0.001000	28	0.969370
250	0.083811	0.867078	0.002000	28	0.950889
225	0.143702	0.835615	0.001500	25	0.979317
250	0.130930	0.856851	0.001500	25	0.987781
250	0.093296	0.874132	0.002000	25	0.967428
200	0.244786	0.797123	0.001000	23	1.041909
225	0.225267	0.809382	0.001000	23	1.034649
250	0.197123	0.809251	0.001000	23	1.006373
200	0.178877	0.777319	0.001500	23	0.956197
225	0.164467	0.849018	0.001500	23	1.013486
250	0.149252	0.845375	0.001500	23	0.994627
200	0.138172	0.848439	0.002000	23	0.986611
225	0.121249	0.798354	0.002000	23	0.919603
250	0.111355	0.877313	0.002000	23	0.988668

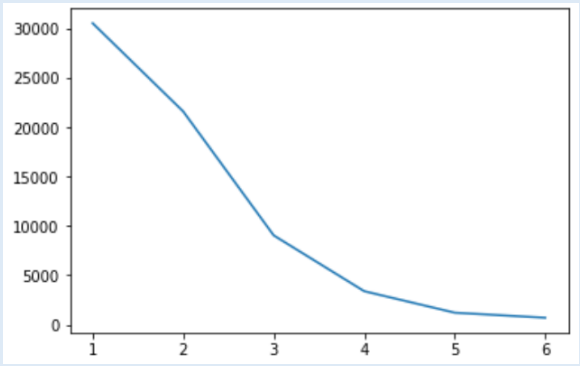
As it is the best compatibility between the RMSE training and validation so we have use it to train our final model.

Then we have used this parameters to train our final model which reach to following evaluation:

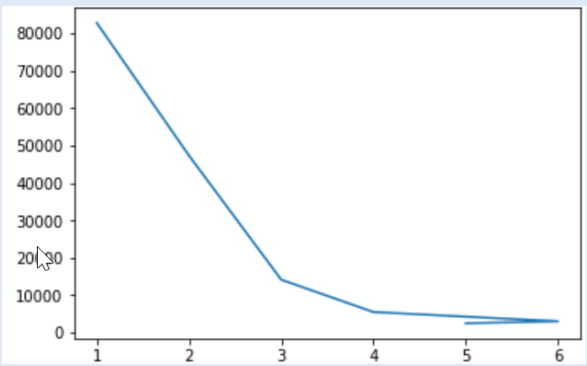
```
Iterations | Mean Squared Error
225              0.123362
```

We can see we have reach the Mean Squared Error on training data to 0.1233.

Then we have starting predicting the test submission dataset using these two matrixes, that show us a very good fitting how the attempts between submission and test submission (what we have predicted) as showing in following Figure:



sample_submissions.attempts_range



submissions.attempts_range

JUSTIFICATION

The final metrics used for evaluating the performance of the model is the F1 score between the predicted and the actual value with average parameter set to 'weighted'. This matric will implement to 66555 record.

The evaluation is happening in the solution checker in contest page , by uploading sample_submission CSV file we have reach to the score:

53	Suli	0.4391992674	

And the first one in this contest was:

1	bhushan	0.5107794039	

Therefore, we have reach 0.439, as the 53th in this contest with model is not the best one.

CONCLUSION

Reflection

When come to some of this type of problems there many possible ways to solve them by trying to benefit from any ways to understating how users and items are correlate to each other, but for my case and solution it was by following:

1. Thinking to find a latent features for problems and users based on their history data.
2. Assume this features are very relative to unseen data and make a suggestions using them.
3. Assume there a similarity of every pair of users or problems based on the dot product of their features.

The difficult of this type of problems in sometimes there no relations between features or between users.

Improvement

To reach to best user experience is will be better to implemented SVD for all the problems and user in online judge platform.

Then using the output matrixes in online platform and staring A/B testing experiments and compare between old user/ problems behavior with old data and starting improving from this point, this will give the online judge platform many ways to improve their business, as:

- To have information about the features of hardest problems that face by their student.
- To increases activates for their problems and users.
- To do more research how to focus their explanation Martials.