Kaggle Name: Chris Bredernitz

Option Choice: Option 1

Overview:

I chose to use the SVD model after much trial and error on my end. I tried a number of different algorithms that were built into Surprise, but settled on the SVD algorithm. KNN (both KNNBasic and KNNwithMeans) seemed to have severe memory issues killing my terminal around 60 GB of memory usage. I played around with the number of `k` clusters, from 4-50 and nothing seemed to fix the memory issue. This made me stop trying this algorithm and go back to testing the SVD algorithm for my predictions. I also tried the SVD++ algorithm and while this seemed promising, the time for learned took around 50+ minutes for a slight bump in predictions. With all of these different algorithms tested, I had the best results using SVD and thus moved forward with tweaking my parameters in SVD.

The parameters I chose were {'n_epochs': 100, 'reg_all': 0.2, 'n_factors': 10, 'lr_all': 0.005} which was giving me a cross validation score of around 1.01. I went with these parameters since these performed the best when running a cross validation and when splitting my training data into a train/test split of 75-25 ratio. I found these parameters through various trials of using the GridSearchVS function built into the surprise library. At first, I had the regularization rate of all values cranked up pretty high (0.4-0.6), but once I started to lower the number of factors (n_factors) from 100 down to ~10, I found that the regularization rate parameter needed to be much lower. After speaking with Professor Jurgens, it made more sense why the lower number of factors would help tune the model better. The number of factors is like the number of layers in a neural network. With that being default to 100 in Surprise, the model was not clustering properly by trying to make more `buckets` of similar items that would otherwise not be treated as their own group. At first, I was having trouble with my predictions capping out around 1.08 with the same parameters above other than. As a last ditch attempt, I figured it might be that my model was not converging properly so I doubled the value to 100. This helped dramatically raise my score on Kaggle to 1.005 and will be submitting this model.

Trial and Error:

Below are all the steps I took to get the model presented in this report

1.	Tried the flat SVD Algo to see how that went. It preformed decent around 1.16.
2.	Looked into trying KNNBasic. There is a serious memory issue with this algo as it kills my terminal after about 15 minutes topping out at ~60GBs.
3.	Tried KNNwithMeans this time and ran into the same issue.

4	Thought that modifying the simulation options with pearsons_correlation might do the trick for KNN. This did help memory a bit. Looked as if the algorithm was going to converge, but yet again my memory ran out.
6-7	Moved back to the SVD algorithm for train/testing. This appears to give me the best results. I tried a cross validation on the data which gave me a some metrics of the RMSE. I plugged in the Standard Deviation to the model and it gave me the best results on Kaggle thus far.
8	Trained a model with the same specs using the SVDpp algo. The difference was minuscule given the time commitment it took to run, but I may lean with this algorithm moving forward as the final piece after testing parameters using GridSearchVS on SVD.
9	Ran a GridSearchVS overnight on everything to get some "best parameters" I'll need to do this once more as I ran it with the incorrect rating range (ran it 0-5 not 1-5).
10	After looking back at the directions I tried to round() each of my estimations through predict to the nearest whole number since a user wouldn't rate something a 3.01213123 in practice. This in-fact hurt my predictions which was a shock to me. I'll keep playing around with parameters. Maybe I'm still using the evaluation piece wrong though surprise as I can't seem to get anything lower that 1.08.
11-18	Trying different parameters and pushing up the results for each to Kaggle. Again, trying to round the results vs not rounding. All still not getting me below the 1.08.
19-22	Went back to just the baseline SVD() algorithm with no parameters. Still getting worse results. I'm not sure even what's going wrong at this point....
23-26	Starting a jupyter notebook to get save time of loading everything each run. Again, can't get below the best results of 1.085. I ran through my code a couple times to see if the error I was getting with these results not matching up was something with my environment and could be solved through a notebook approach. It didn't help and I was getting the same results
27 -28	Went through some tutorials online. They had their learning rate cranked up to 0.4 and 0.6 whereas mine is at 0.15. From the GridSearchVS I found that the higher regularization was the best so I tried this approach. Still nothing...
29	param_grid = {'n_epochs':[5, 10, 15], 'reg_all': [0.4, 0.6], 'n_factors':[50, 100, 200]} gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=5) 1.0306705562904341 {'n_epochs': 15, 'reg_all': 0.4, 'n_factors': 50} I know that 50 epochs is better, but for time I used a lower number to see how each performed knowing that these will be best case parameters once I get it to converge a bit lower

30-31	30. param_grid = {'n_epochs':[15, 30], 'reg_all': [0.2, 0.4], 'n_factors':[10, 15, 20]} gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=5) 1.011501376310423 {'n_epochs': 30, 'reg_all': 0.2, 'n_factors': 10} These are the best results I've gotten thus far when running a cross validation. Going to try these on the dataset that I upload to Kaggle and see where this leads. Uploaded to Kaggle and still only got a 1.083. I believe there is something wrong with my data values when loading. It's strange since when looking at my predictions nothing is really at the `5` rating. I might see if I can up my prediction range to 1-6 to see how everything works.
32	I ran a comparison on the dev csv data just to see how accurate the predictions are in practice. The model appears to give an ~80% accuracy rate when compared to the actual predictions. I feel as though this model is tuned appropriately, but still not sure as my Kaggle standing doesn't seem to improve much with each submission. With the cross validation score I received, I feel as though once Kaggle runs the remaining 90% it will be lower, but only time will tell.
33	I believe I have the best model I can get. I'm not entirely sure why my model does not want to go lower than 1.08 on Kaggle. My evaluation of the model for this submission comes from cross validating with Surprise's built in testing piece which is giving me roughly 1.01. I'm again going to look at how I'm writing my csv file for output to check if something is going wrong here, but from what I can tell my code seems correct.
34	Just as another sanity check I split the data into a train/test set to evaluate against the training set. I split the training data into a 75-25 set and evaluate in the test_accuracy() function. Again, my model appears to do fairly well getting an RMSE of ~1.009. I'm going to move forward with what I have and see what the final results bring me. I have triple checked my csv functions and hope that the work I put into this assignment isn't hurt by the Kaggle results I'm receiving.
35	As a last-ditch attempt, I went back through last night to test if wrapping my prediction values in an str() function was the cause of these predictions being off. My intuition was that some of the `asin` values were integers and Surprise is expecting a string. This didn't appear to be my problem as my Kaggle results didn't appear to change yet again.
36	Well, I figured it out. With lowering the number of factors in my model, I needed to triple the number of epochs that the model trained. This gave me the best results of 1.005 which were my best results thus far. Can't believe it took me this long to figure out that the number of epochs I was using previously on my higher learning rate model needed to be increased to converge properly. I also added a .strip() function to my prediction strings in this attempt too. The intuition was that when reading the csv file, there could have been a mistreatment of keys with the same values not pairing correctly.

Chris Bredernitz

Error Analysis:

1) A2TAAFL2CH550H,6304431856,"1",4.474289570008867
2) A3QEPSZ62L6BU5,B000CQO6IY,"1",4.271996205495925
3) A2A2SUY3IOQKY1,0790732254,"2",3.1831679593899165
4) APYCNRS112N2B,0783230346,"1",3.3702678866219946
5) A3UKG41JF058ZM,B008220C38,"1",2.3688969098469626

After looking at these closer in the training data, it would seem that these items were not represented enough in the data. Item `6304431856`, for example, was only represented 5 times in the training data. Surprise fills in the mean score of the item if there is no prior knowledge of how this user will rate a specific item so that would appear to be the cause of the drastic score change. Similarly, most of the users here that are off on their predictions have very little recommendations in the training set. User ` A3QEPSZ62L6BU5` for example only rated 4 items. Because of this, the model doesn't have enough information on what they like/don't like to make a more accurate recommendation.

To fix this in the future, I would recommend that a user must select 10 items to rate so they are more represented in the training data. Twitter solves this issue by forcing users to `follow` up to 10 other users when signing up so they have a better idea what the user likes/doesn't like. This would lead to more accurate recommendations overall.

Files:

- Recommender_Trials.py:
    - This includes code used to test different algorithms I have this included to show some progress made and what I tried
- Recommender.py:
    - This is the **final** code file used to get the Kaggle submission I have. This file contains all the code used for the working model
- outputsSvd.csv:
    - Contains all the datapointID's and predictions that my model gave. This is the submission that gave me the best results on Kaggle.
- evaluateSvd.csv
    - Used to compare actual with the predicted value for error analysis.