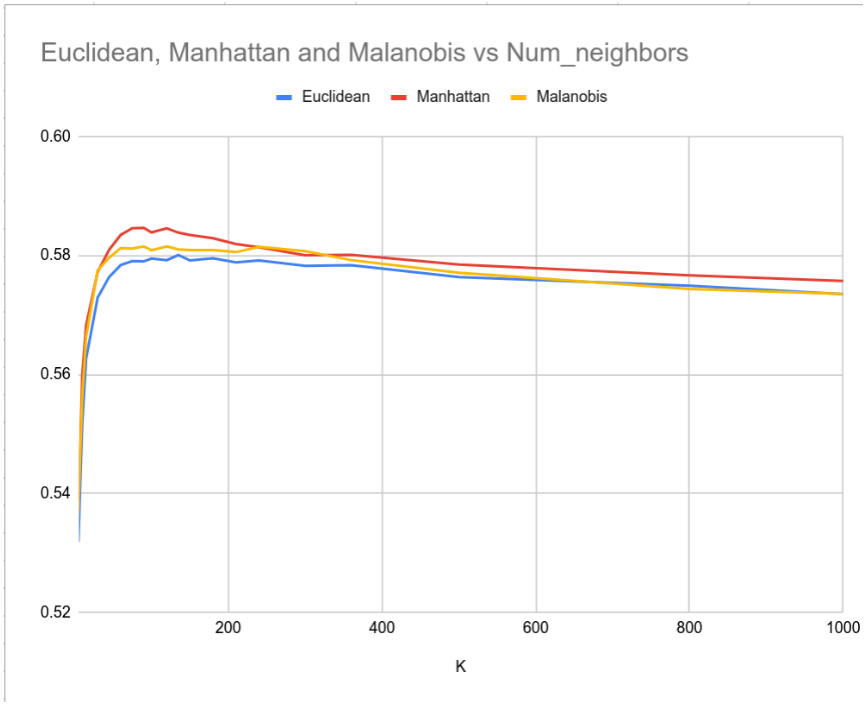
CS506 Midterm

How I did it: Francisco Xavier Turrubiate

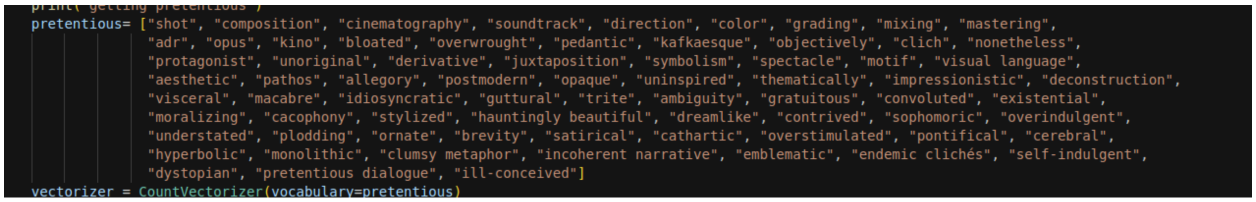
Coming out of this midterm, I managed to get an accuracy of 63.451% on the private leaderboard, but the journey to that number taught me so much more than I could have imagined; let's get into that!

Jumping into the assignment, I first got the starting code working, and tried to get cuML installed to take advantage of my GPU; although I'm familiar with Python virtual environments, this was my first time really working with Anaconda in my own volution. I then created a little preprocessing (using a standard scalar) and started playing with searching for the optimal number of neighbors in KNN, I also started comparing different distance metrics and found Manhattan was the one to stick with.

One I tried, Mahalanobis, was nightmarish to figure out how to implement, due to it needing the covariance matrix, but when I was learning how to implement it efficiently I was so confident it would work better. I was devastated when Manhattan ended up being better, but it helped jog my linear algebra knowledge and might be helpful to know in the future, who knows?

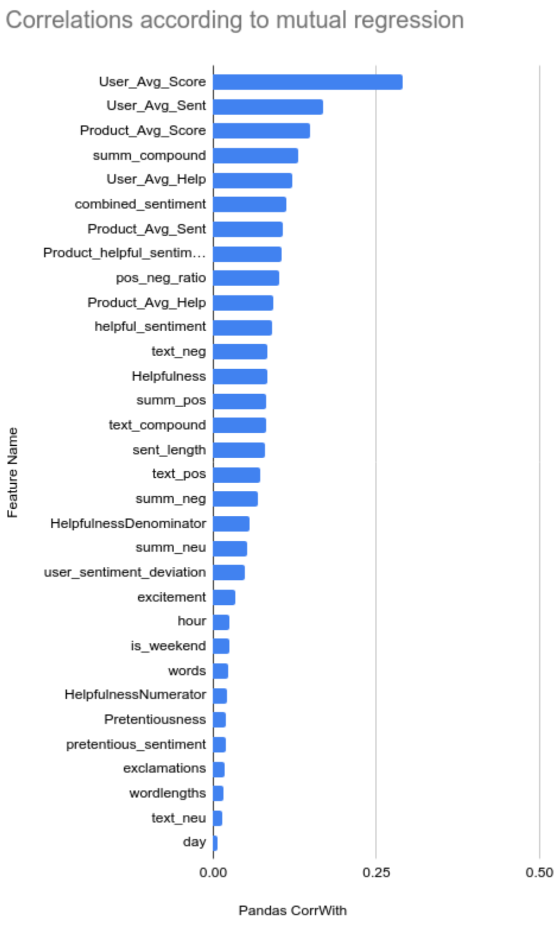


This was also where I started thinking about my first features. As someone active in the music community, I have a bit of experience with how people behave in spaces like music/movie reviews, the first thing I think of when it comes to movie reviews is how pretentious people can be sometimes, so why not target this? This is where my first new feature, **Pretentiousness** was born. I created a list of words that captures what is commonly referred to in pretentious reviews



```
pretentious= ["shot", "composition", "cinematography", "soundtrack", "direction", "color", "grading", "mixing", "mastering",
              "adr", "opus", "kino", "bloated", "overwrought", "pedantic", "kafkaesque", "objectively", "clich", "nonetheless",
              "protagonist", "unoriginal", "derivative", "juxtaposition", "symbolism", "spectacle", "motif", "visual language",
              "aesthetic", "pathos", "allegory", "postmodern", "opaque", "uninspired", "thematically", "impressionistic", "deconstruction",
              "visceral", "macabre", "idiosyncratic", "guttural", "trite", "ambiguity", "gratuitous", "convoluted", "existential",
              "moralizing", "cacophony", "stylized", "hauntingly beautiful", "dreamlike", "contrived", "sophomoric", "overindulgent",
              "understated", "plodding", "ornate", "brevity", "satirical", "cathartic", "overstimulated", "pontifical", "cerebral",
              "hyperbolic", "monolithic", "clumsy metaphor", "incoherent narrative", "emblematic", "endemic clichés", "self-indulgent",
              "dystopian", "pretentious dialogue", "ill-conceived"]
vectorizer = CountVectorizer(vocabulary=pretentious)
```

and directly counted it per row. I also created average scores per product and per user, but I was really paranoid about accidentally 'cheating' while Implementing it; I created a whole separate *post-processing* function after the data was split, and it runs on X_train first, before applying its findings to X_test, and X_submission separately of their scores.



I then decided to experiment with the text. Using what I learned in class, I immediately tried TF-IDF and LSA to get a set of components; I found a function in pandas, Corrwith, to find which components had the highest correlation with score, and found that the LSA columns had extremely low correlation. Looking back, I should have still included these as features in my model, but I ended up scrapping all of it and lost the code that calculated it when I wanted it (I WISH I DIDN'T DELETE IT).

Corrwith (I also found another correlation metric with mutual info regression) ended up being helpful though, as it helped me test features and see what felt helpful; The ones I liked were time-related features like **day** (the weekday), **hour**, and **is_weekend** as movie reviews had correlations with when people went to the theaters. This is where I then decided to attack the text again, and really add more features.

I found VADER, a sentiment analysis that specializes in stuff like reviews, and felt it would have much stronger correlations with scores compared to the TF-IDF/LSA setup I tried, and added the four categories of scores it outputs in pre-processing. This drastically improved my accuracy. I then made a bunch of combined features that I felt represented what I know from my music rating experience, like **pretentious_sentiment** which multiplies both features to emphasize when someone is both pretentious and feels strongly about something (pretentious people like to either praise something as "peak" or be contrarians), but also **helpful_sentiment** combining helpfulness and the compound sentiment (people like to put helpful when sentiment leans one way strongly. I also added average helpfulness and sentiments per product and user, and also the standard deviation of users' sentiments, to highlight when some reviewers like skewing rating numbers more than anything (very common in music spaces). In the chart above this, you can see the features and their corrWith/mutual-info-regression scores. I knew music-reviewing trends would be reflected here, but I was surprised by the results this gave me.

After trying Scikit's GridSearchCV and RandomizedSearchCV to find good KNN parameters, I hit a wall in the highest accuracy I was able to hit with these features, which led me to search for alternatives. Discovering the concept of RandomForests, and then gradient boosting through XGBoost right after, I found my accuracy once again jumped to 63% (after searching for good parameters), which is what became my final submission.

Combing through the documentation of the parameters, and learning how randomForests and gradient boosting work was one of the most valuable things I got out of this midterm, I feel like the hands-on experience drilled down what I was unsure about in TF-IDF, LSA, and what makes a good feature space. Looking back, with the knowledge I have right now, I would have had a completely different approach to this type of problem, and I adore how it feels like I learned just as much as I was tested. (I also learned how to make cool progress bars, and handle better parallel processing better!)

Most of my searching around took the form of browsing the documentation of cuML (it has fewer parameters and functions supported, reducing my options on models that kept it much more approachable than Scikit), and XGBoost, alongside occasionally referring to chatGPT to help debug weird errors (often due to CUDA being weird with what objects it accepts), and jog my knowledge on things.

Citations:

https://chatgpt.com/
https://medium.com/dropout-analytics/beginners-guide-to-knn-with-cuml-ddca099f9e9d
https://developer.nvidia.com/blog/accelerating-k-nearest-neighbors-600x-using-rapids-cuml/
https://docs.rapids.ai/api/cuml/stable/api/
https://ibmsoe.github.io/snap-ml-doc/v1.6.0/knndoc.html
https://scikit-learn.org/1.5/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
https://scikit-learn.org/1.5/modules/generated/sklearn.feature_selection.mutual_info_classif.html
https://scikit-learn.org/1.5/modules/generated/sklearn.covariance.LedoitWolf.html
https://www.nltk.org/
https://developer.nvidia.com/cuda-toolkit
https://pypi.org/project/vaderSentiment/
https://rapids.ai/
https://en.wikipedia.org/wiki/Mahalanobis_distance
https://pypi.org/project/tqdm/2.2.3/
https://tqdm.github.io/
https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html
https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/
https://xgboost.readthedocs.io/en/stable/parameter.html
https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.LabelEncoder.html