# CSC311 Final Project

Nimit Bhanshali, Dhairya Jain, Eren Findik

December 2, 2022

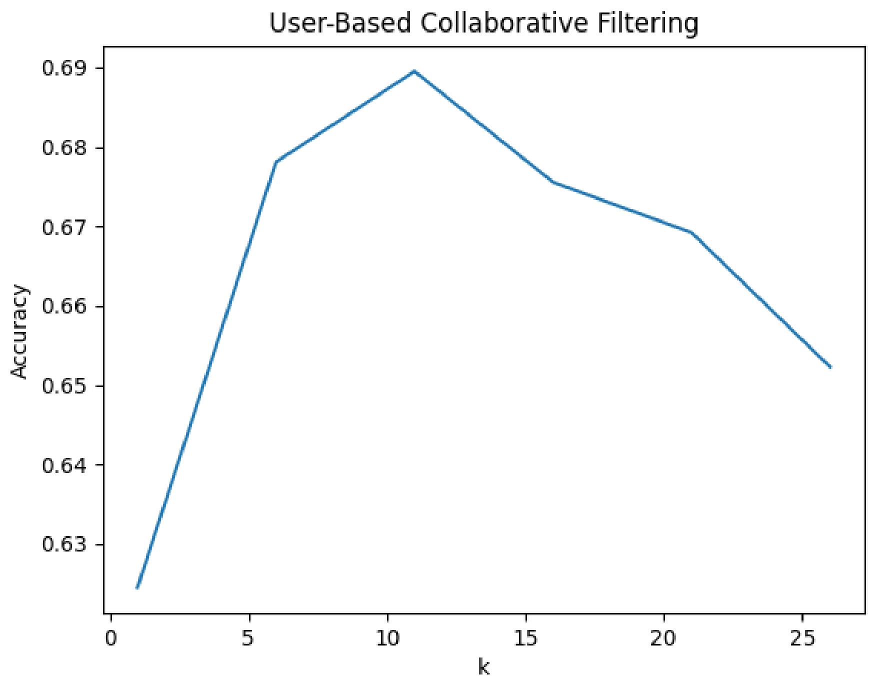
## Part A

### 1. k-Nearest Neighbor

#### (a)

The function **main** is completed in **knn.py**.

When $k = 1$, validation accuracy is 0.6244707874682472.
When $k = 6$, validation accuracy is 0.6780976573525261.
When $k = 11$, validation accuracy is 0.6895286480383855.
When $k = 16$, validation accuracy is 0.6755574372001129.
When $k = 21$, validation accuracy is 0.6692068868190799.
When $k = 26$, validation accuracy is 0.6522720858029918.



#### (b)

$k^* = 11$, chosen based on the value of $k$ that has the highest performance on the validation data.

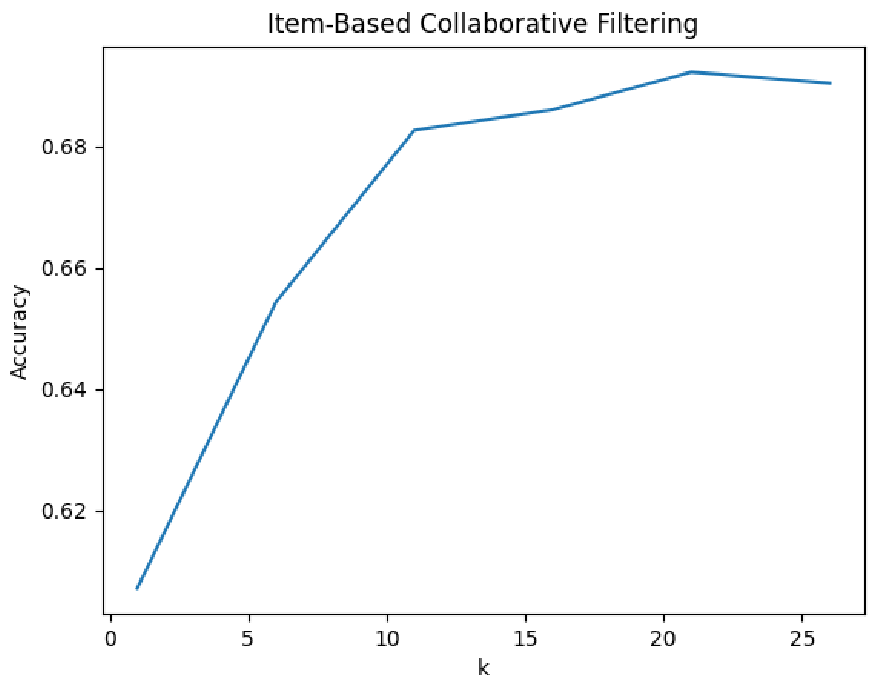
The final test accuracy is 0.6841659610499576.

#### (c)

The function **knn_impute__item** is completed in **knn.py**.

The underlying assumption on item-based collaborative filtering is that if question A has the same students answering it correctly and the same students answering it incorrectly as question B, then the correctness of a specific student's answer to question A matches that of question B.

When $k = 1$, validation accuracy is 0.607112616426757.
When $k = 6$, validation accuracy is 0.6542478125882021.
When $k = 11$, validation accuracy is 0.6826136042901496.
When $k = 16$, validation accuracy is 0.6860005644933672.
When $k = 21$, validation accuracy is 0.6922099915325995.
When $k = 26$, validation accuracy is 0.69037538808919.



$k^* = 21$, chosen based on the value of $k$ that has the highest performance on the validation data.

The final test accuracy is 0.6816257408975445.

### (d)

Since, $0.6841659610499576 > 0.681625740897544$, the test performance for user-based collaborative filtering is better than the test performance for item-based collaborative filtering. Thus, we can conclude that user-based collaborative filtering performs better than item-based collaborative filtering.

### (e)

One potential limitation of kNN for the task we are given is the curse of dimensionality. Both user-based collaborative filtering and item-based collaborative filtering have high-dimensional datasets with 1774 and 542 features respectively. As we know, in high dimensions most points can be considered to be far apart and approximately the same distance from each other. Hence, the nearest neighbors to any input may not reflect the same correctness as that of the input.

Another potential limitation of kNN for the task we are given is the computational cost. The kNN algorithm conducts all of its computations at test time and none at training time. For each input from the test set, the algorithm will at most carry out $ND$ computations to calculate the $D$-dimensional Euclidean distance with $N$ data points in the training set. In our case, this could amounts up to $1774 \times 542 = 961508$ computations. Additionally, the algorithm will also sort these distances to determine the nearest neighbors which will require at most $N \log(N)$ computation. In our case, this could amount up to either $542 \log(542) \approx 4993$ computations or $1774 \log(1774) \approx 19146$ computations. Hence, we see that kNN for our task would require large numbers of computations resulting in high computational cost.

## 2. Item Response Theory

**(a)**

$$p(c_{i,j}=1 \mid \theta_i, \beta_j) = \frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}$$

$$p(C \mid \theta, \beta) = \prod_i \prod_j \frac{\exp((\theta_i - \beta_j)c_{ij})}{1+\exp(\theta_i - \beta_j)}$$

$$\log p(C \mid \theta, \beta) = \log\left(\prod_i \prod_j \frac{\exp(C_{ij}(\theta_i - \beta_j))}{1+\exp(\theta_i - \beta_j)}\right)$$

$$= \sum_i \sum_j \log\left(\frac{\exp(C_{ij}(\theta_i - \beta_j))}{1+\exp(\theta_i - \beta_j)}\right)$$

$$= \sum_i \sum_j \left[\log(\exp(c_{ij}(\theta_i - \beta_j))) - \log(1+\exp(\theta_i - \beta_j))\right]$$

$$= \sum_i \sum_j \left[c_{ij}(\theta_i - \beta_j) - \log(1+\exp(\theta_i - \beta_j))\right]$$

$$\frac{\partial \log p(C \mid \theta, \beta)}{\partial \theta_i} = \sum_j \left[C_{ij} - \frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}\right]$$

$$\frac{\partial \log p(C \mid \theta, \beta)}{\partial \beta_j} = \sum_i \left[-C_{ij} - \frac{(-1)\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}\right]$$

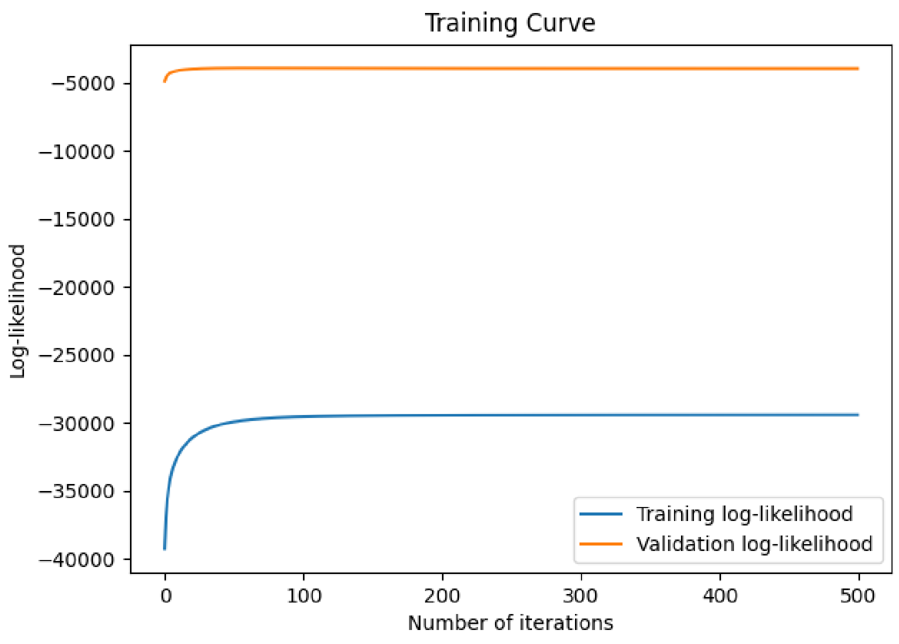
$$= -\sum_i C_{ij} + \sum_i \frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)}$$

$$= \sum_i \left[\frac{\exp(\theta_i - \beta_j)}{1+\exp(\theta_i - \beta_j)} - C_{ij}\right]$$

**(b)**

The missing functions are implemented in **item_response.py**.

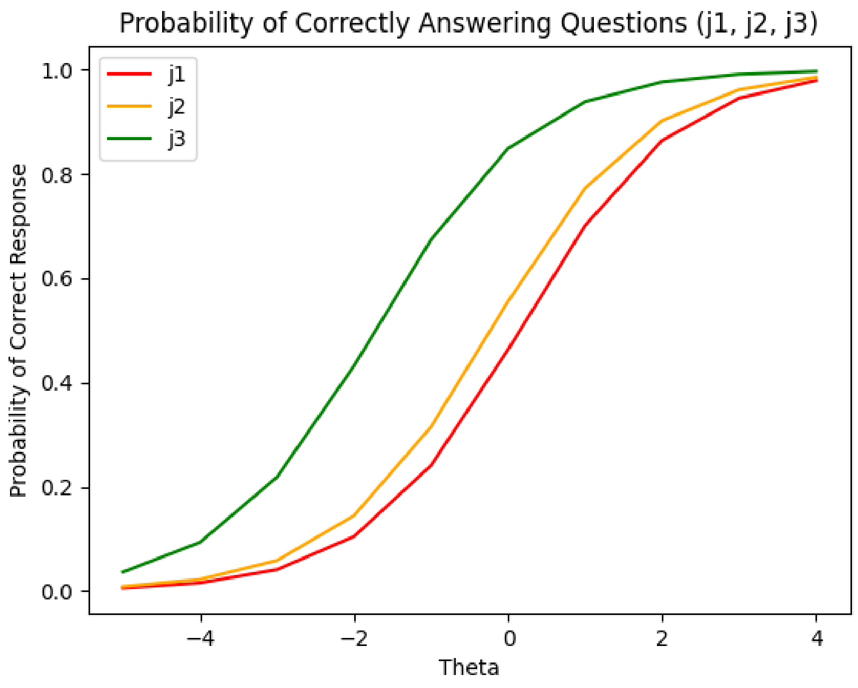
The selected hyperparameters are:
Learning Rate = 0.005
Number of Iterations = 500



**(c)**

The final validation accuracy is 0.7074513124470787.
The final test accuracy is 0.7084391758396839.

**(d)**



The plot shows curves with the shape of a sigmoid graph representing the probability of the correct response as a function of $\theta$ given some question $j$. The curves have the shape of a sigmoid graph as we were given that the probability of a correct response follows a sigmoid function. These curves also show that the higher the value of $\theta$, the higher the probability of a correct response to a given question. This represents that the higher the student's ability is, which is measured by $\theta$, the higher the probability of a correct response to some question.

## 3. Matrix Factorization

### (a)

$k = 25$ is the best $k$ that is chosen using the validation set.
Final validation accuracy when $k = 25$: 0.6594693762348293.
Final test accuracy when $k = 25$: 0.6556590460062094.

### (b)

We fill the missing entries with the mean of other entries to be able to perform SVD. Therefore, although this helps us simplify our data it may make the transformed data hard to understand.

### (c)

The functions **als** and **update_u_z** is completed in **matrix_factorization.py**.

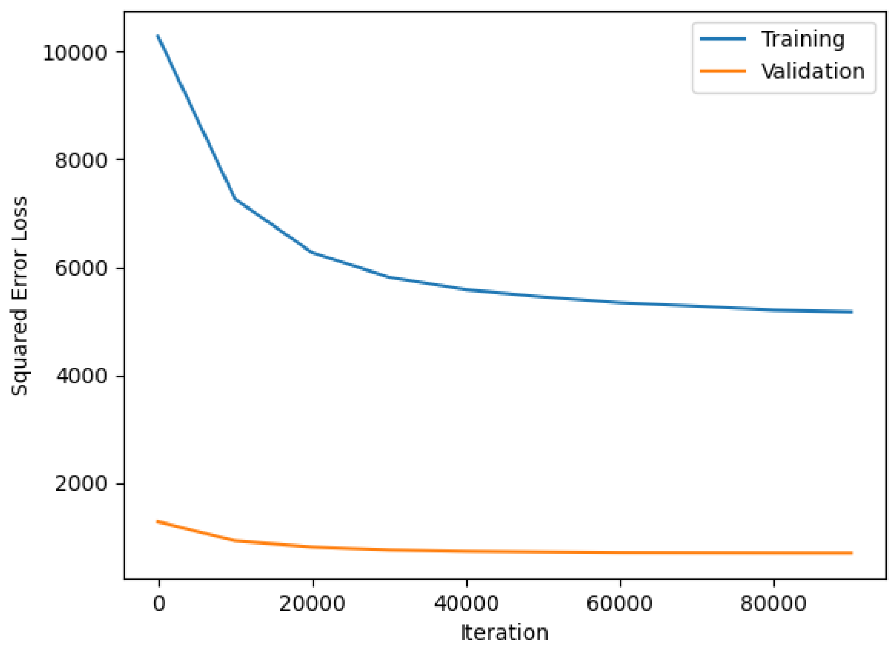
### (d)

After tuning the learning rate and the number of iterations, we chose the following hyperparameters:

Learning Rate = 0.1
Number of Iterations = 100000

$k^* = 30$, is the best value of $k$ as it achieves the highest validation accuracy.

**(e)**



Above is a plot of how the training and validation squared-error-losses change as a function of iteration. We see that the plot follows a reciprocal function where the initial decrease in the squared-error-losses is very large as the iterations increase and then continue to decrease at a slower rate till it converges to a constant.

The final validation accuracy when $k^* = 30$: 0.6965848151284222.
The final test accuracy when $k^* = 30$: 0.6996895286480384.

## 4. Ensemble

We decided to use Item Response Theory from Q2 as the base model of our ensemble. The ensemble process we used consists of:

1. Bootstrapping the training dataset by selecting random samples from the training data with replacement. The size of these new datasets are equivalent to the original training data.
2. Training a Item Response Theory model on 3 bootstrapped datasets and storing the associated theta and beta.
3. Calculating the validation and test predictions.
4. Averaging the predictions by the following formula (equivalent to majority voting):

$$y_{ensemble} = \sum_{i=1}^{3}(\frac{y_i}{3} > 0.5)$$

5. Finally, the accuracy is calculated by the predictions made.

Final Validation Accuracy: 0.7029353655094552
Final Test Accuracy: 0.7039232289020604

Does using the ensemble obtain better performance? Why or why not?

The IRT model originally used in Q2 has a final validation accuracy of 0.7074513 and a final test accuracy of 0.7084391. That means the ensemble produced a negligibly lower accuracy than the base model by itself. Thus it did not obtain better results.

This could be because the model already had enough data to effectively train the parameters. Also if the test data had the same distribution as the training data, generalizing more data would not positively impact the performance of the model.

## Part B

**Formal Description:**

In this part, we have chosen to modify the k-Nearest Neighbors algorithm to hopefully predict students' answers to the diagnostic questions with higher accuracy. From the previous part, we learnt that the major limiting factor in the performance of the k-Nearest Neighbors algorithm was the curse of dimensionality. For both user-based collaborative filtering and item-based collaborative filtering, we see that we are working with 1774 and 542 dimensions, respectively. We know that the effectiveness of the k-Nearest Neighbors algorithm decreases with higher dimensions, as in high dimensions most points are considered to be far apart and equidistant from each other. Therefore, the concept of "nearest neighbors" is lost as even the closest neighbor, in terms of Euclidean distance, may seem far away from the input data point being considered and hence each neighbor may seem equidistant.

Therefore, a proposed modification to help address this problem is to change the distance metric that is being used. So far we have been using the Euclidean distance metric to determine the distance between a data point and its neighbor. However, given the curse of dimensionality, we will change this and use the Hamming distance metric instead as it is a distance metric that is preferable for higher dimensions.

The Hamming distance will be used on our binary classification of responses where 1 indicates a correct response and 0 indicates an incorrect response. The Hamming distance is calculated by first performing the XOR operation on the two inputs vectors, then counting the number of 1s in the resultant vector and lastly dividing it by the number of dimensions.

More formally, let $\mathbf{x}^{(a)}, \mathbf{x}^{(b)}$ be two samples from the training set where each has $d$ features. In other words, they are $d$-dimensional vectors. Then, we can compute the Hamming distance between the two vectors using the following formula:

$$hamming(\mathbf{x}^{(a)}, \mathbf{x}^{(b)}) = \frac{1}{d}\sum_{i=1}^{d} x_i^{(a)} \oplus x_i^{(b)}$$

Intuitively, this distance metric works better in higher dimensions because unlike Euclidean distances it is not as sensitive to extreme values. With higher dimensions, we can expect there to be some outlier cases where the difference between the specific vector coordinates is larger in one of the dimensions than in the others. Using the Euclidean distance, we would see this difference be exaggerated due to the squaring of distances. This leads to the problem of having large distances for the distance metric computed between the input vector and the neighbor, making it seem as if all neighbors are far apart. Whereas, using the Hamming distance, there is no such exaggeration in the computation of the distance. The Hamming distance simply computes the fraction of features where the features between two vectors don't match and that in itself is the resultant distance measure. Therefore, when working in higher dimensions with the Hamming distance, we do not face such severe consequences of the curse of dimensionality as it is far less sensitive to extreme values than the Euclidean distance.

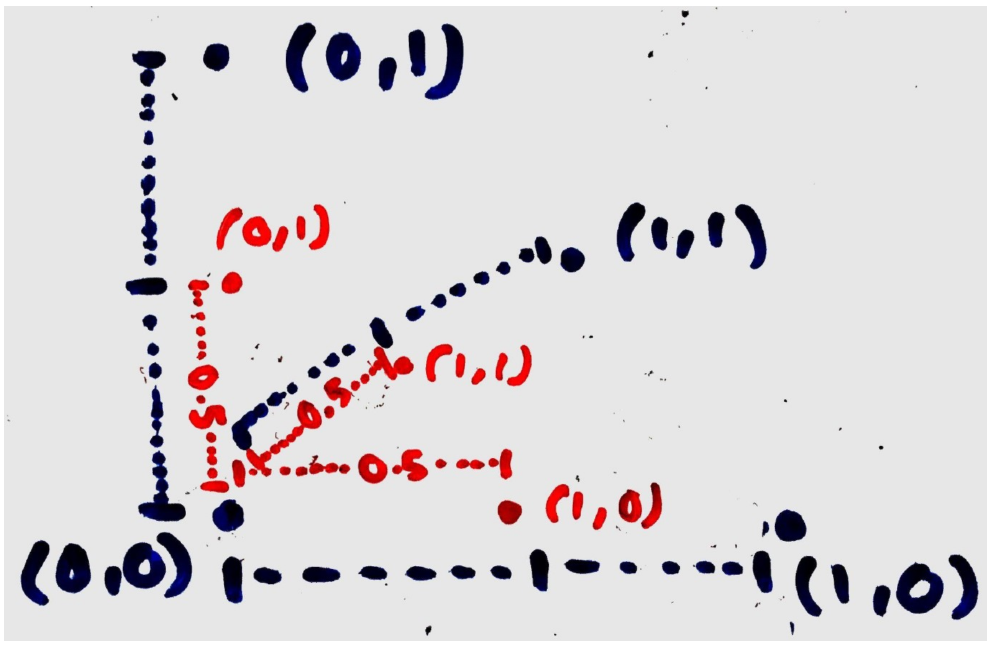
Thus, this is how our resultant modified k-Nearest Neighbors algorithm would look like,

**Algorithm:**

1. Let $\mathbf{x}$ be the input vector.
2. For each sample in the training set, calculate the Hamming distance between the sample and the input vector.
3. Find the $k$ training samples with the smallest computed Hamming distance, to determine the $k$ nearest neighbors.
4. Classify output of $\mathbf{x}$ by majority vote based on the labels of the $k$ training samples.

**Diagram:**

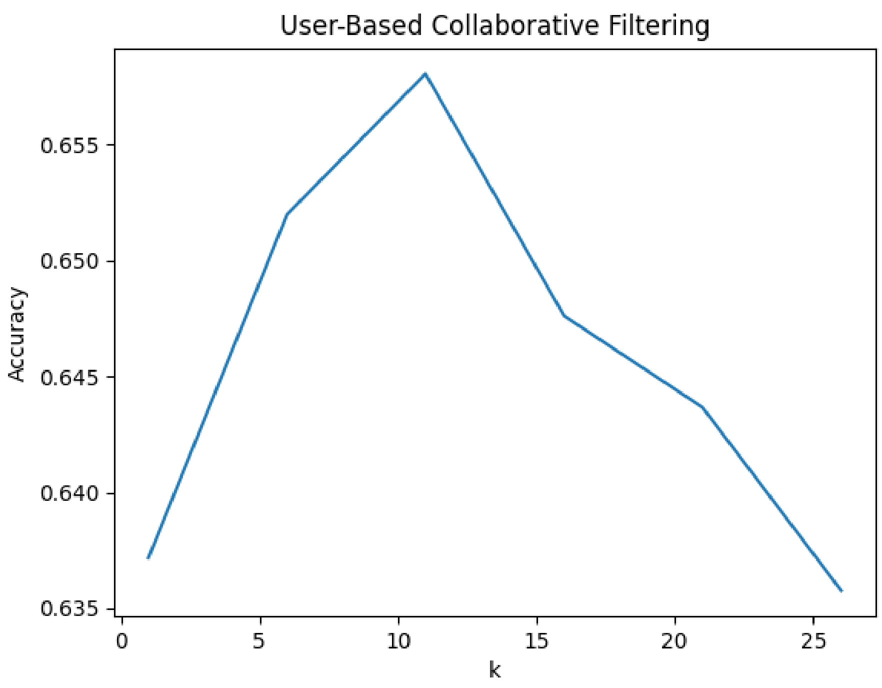
In the diagram below, we see the representation of the Euclidean distance in the blue and we see the representation of the Hamming distance in red. We see that between the same coordinates/vectors, the Euclidean distance measured between the points is 1 unit whereas the measured Hamming distance between the points is 0.5 units. This illustrates the overall idea of our proposed modification to change distance metrics. We can visually see here that the neighbors are closer together when the Hamming distance is our distance metric. Although, this is a visualization in the 2-dimensional space where the curse of dimensionality doesn't apply, this idea extends to higher dimensions. In higher dimensions, we will continue to have smaller distance values between the input vector and its neighbors when using the Hamming distance when compared to the Euclidean distance. Therefore, most points will not be considered far apart and the neighbors will not appear equidistant either.



## Comparison:

The modified algorithm is completed in **knn_modified.py**. We will now look at the performance of this modified algorithm. We will first tune the hyperparameters using the validation set.

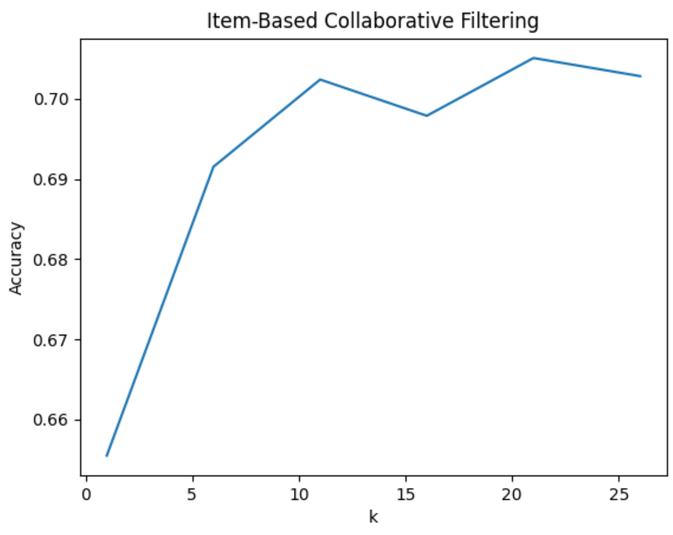
Starting with user-based collaborative filtering,
When $k = 1$, validation accuracy is 0.6371718882303133.
When $k = 6$, validation accuracy is 0.6519898391193903.
When $k = 11$, validation accuracy is 0.658058142816822.
When $k = 16$, validation accuracy is 0.6476150155235676.
When $k = 21$, validation accuracy is 0.643663561953147.
When $k = 26$, validation accuracy is 0.6357606548123059.



$k^* = 11$, chosen based on the value of $k$ that has the highest performance on the validation data for user-based collaborative filtering.

The final test accuracy for user-based collaborative filtering is 0.6446514253457521.

Now looking at item-based collaborative filtering,
When $k = 1$, validation accuracy is 0.6555179226644087.
When $k = 6$, validation accuracy is 0.6915043748235958.
When $k = 11$, validation accuracy is 0.7023708721422524.
When $k = 16$, validation accuracy is 0.6978549252046289.
When $k = 21$, validation accuracy is 0.7050522156364663.
When $k = 26$, validation accuracy is 0.7027942421676545.



$k^* = 21$, chosen based on the value of $k$ that has the highest performance on the validation data for item-based collaborative filtering.

The final test accuracy for item-based collaborative filtering is 0.6951735817104149.

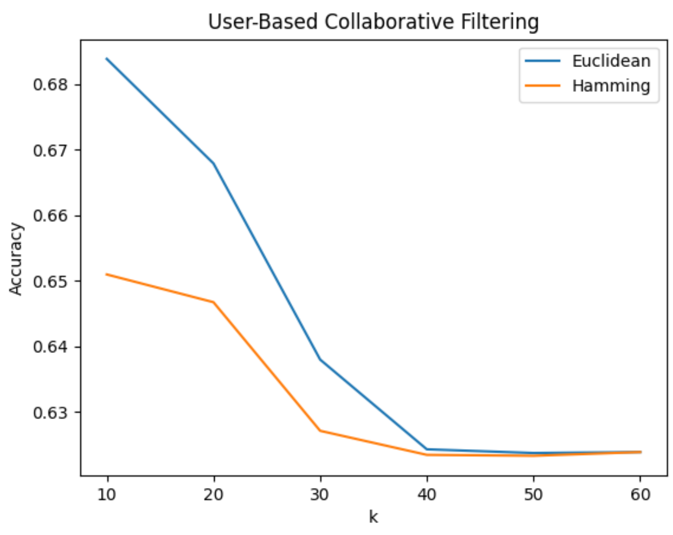
Below we have a table comparing the test accuracies of our modified kNN algorithm to the other baseline models that we investigated in Part A:

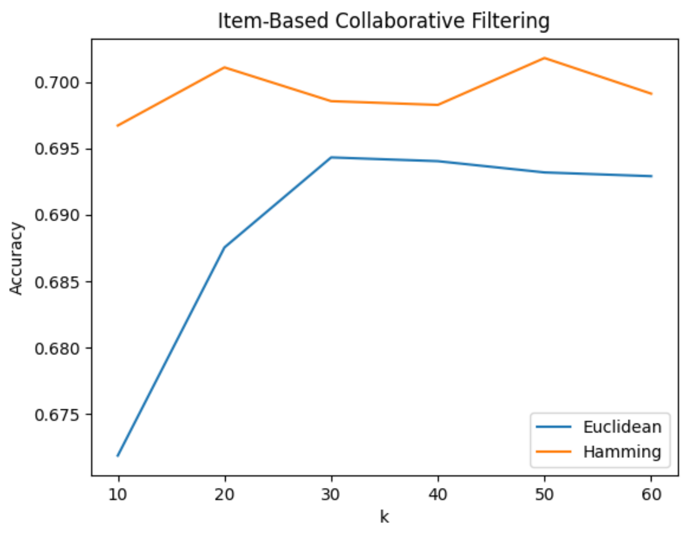
| Models | User-Based Test Accuracy | Item-Based Test Accuracy |
|---|---|---|
| kNN - Hamming | 0.6446514253457521 | 0.6951735817104149 |
| kNN - Euclidean | 0.6841659610499576 | 0.6816257408975445 |
| IRT | 0.7084391758396839 | |
| Matrix Factorization (SVD) | 0.6556590460062094 | |
| Matrix Factorization (ALS) | 0.6996895286480384 | |
| Ensemble | 0.7039232289020604 | |

Here we can see that our improved model performed better than the the original kNN model in Item-Based Collaborative Filtering but was much worse in User-Based Collaborative Filtering. It also performed notably better than Matrix Factorization (SVD) model. The performance of this modified model is still slightly behind the other baseline models but is within the range of approximately 1% in terms of the test accuracy.

Earlier in this part, we made the argument that this change in distance metrics from Euclidean distance to Hamming distance will help improve the accuracy of the kNN model as it will dampen, if not eliminate, the effect of the curse of dimensionality. To test this hypothesis, we have designed the following experiment. To know if the benefits of the modified model are truly changing the distance metrics, we must know that the the neighbors that are being looked at are in fact the "nearest" and not random neighbors that are being chosen because all neighbors are considered to be far and equidistant. To verify this, we will run both the original and modified kNN models on different values of $k$ and compare their accuracies. The intuition behind this is that although $k$ is a hyperparameter that needs to be tuned, when comparing these two kNN models against each other we will be determine whether the quality of neighbors being selected by the modified model are better than the original model. Hence, this will allow us to make a better judgement on whether the proposed modification directly improves the kNN algorithm, as argued above.

The experiment is completed in **experiment.py**.





In the plots above, we see that for user-based collaborative filtering the accuracy initially is much higher for the Euclidean distance model than the Hamming distance model. However, as the value of $k$ increases we see the accuracies for both models converge to a similar value. Although this disagrees with our hypothesis, it can be explained by the extension of data that we go into more detail in the limitations section below.

For item-based collaborative filtering, we see that the accuracy for the Hamming distance model is notably greater than that of the Euclidean distance model. Therefore, this follows our hypothesis and shows that the Hamming model selects better neighbors than the Euclidean model, supporting our argument for making the modification to the original kNN model.

## Limitations:

Despite our goal to optimize the kNN algorithm by minimizing the effect of the curse of dimensionality, the quantitative results show that, with the given test and validation data, our extended model had a negligible impact on the accuracy performance metric.

- Under user-based collaborative filtering, the kNN model with a Hamming distance metric performed worst than the model with the Euclidean distance metric. This implies that the similarity between students is less dependent on the number of specific questions they have answered correctly or incorrectly. This presents a limitation on our model and by extension the data we are working with.

- There is a large variability between the number of questions each student has answered. Thus, two similar students are considered dissimilar due to the number of questions each one has answered. Our model treats unanswered questions by one student in the same light as a mismatched answer which leads to this issue.

- Our model does not consider that even though two students may have very similar answers to a certain type of question, opposing answers to unrelated questions can decrease their similarity. Thus, our model does not take into account the nuance in a students ability to answer questions in different fields of knowledge.

- Under item-based collaborative filtering the kNN with a hamming distance measure performed slightly better than the euclidean distance. This would imply that similar questions are better determined by the number of alterations required to match every students performance on each question rather than their euclidean distance. This could be because there is less overall variability in the number of times each question has been answered.

- There are distinct limitations when working with a kNN model as it is considered a fairly simple model without the ability to effectively make more nuanced predictions based on all the data as a whole. Instead it purely focuses on the $k$ nearest neighbours to make its prediction.

Some possible extensions to the current model include creating a secondary algorithm that uses a separate model to more effectively generate predictions by increasing the weight of specific students or questions that the kNN deems to be most similar. This could involve a deeper consideration being placed on the specific subject of each question.

Another possible extension would be to introduce weights to each of the neighbors such that neighbors that are considered to be ”closer”, in terms of the imposed distance metric, are prioritized more than the ones that are considered to be ”farther” away.

Furthermore, an open problem and a possible extension could be to determine the overall difficulty of each question and use the age of students to more definitively determine that harder math problems can’t be solved by younger students.