CSC311 Final Report

Benson Li

April 25, 2023

1 Part A

1.1 k-Nearest Neighbor

1.1.1 (a)

I got the following output as my result:

```
=== User-based ===
k = 1  Validation Accuracy: 0.6244707874682472
k = 6  Validation Accuracy: 0.6780976573525261
k = 11  Validation Accuracy: 0.6895286480383855
k = 16  Validation Accuracy: 0.6755574372001129
k = 21  Validation Accuracy: 0.6692068868190799
k = 26  Validation Accuracy: 0.6522720858029918
```

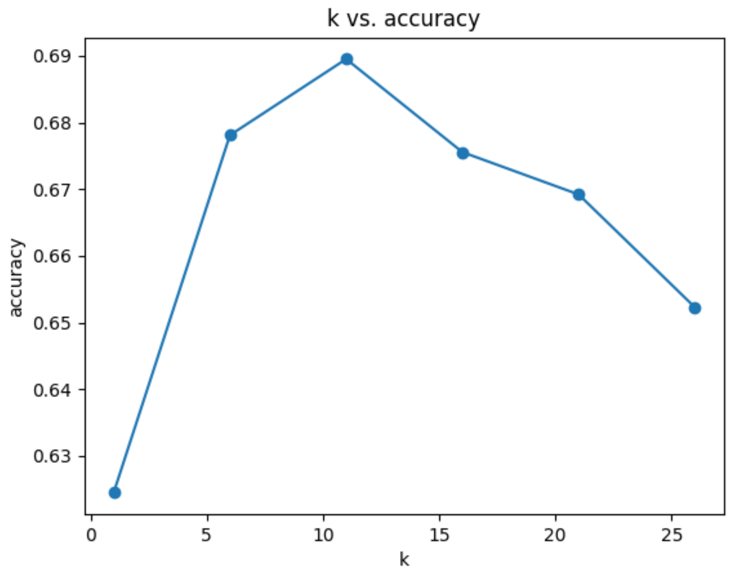

Figure 1: user_based_knn

1.1.2 (b)

The core underlying assumption is that if a student has the correct and incorrect answers on a similar question, they can answer this question correct/incorrect. I choose $k^* = 11$, with the validation accuracy of 0.68953

The corresponding test accuracy is 0.68417

1.1.3 (c)

I got the following output as my result:

```
=== Item-based ===
k = 1  Validation Accuracy: 0.607112616426757
k = 6  Validation Accuracy: 0.6542478125882021
k = 11  Validation Accuracy: 0.6826136042901496
k = 16  Validation Accuracy: 0.6860005644933672
k = 21  Validation Accuracy: 0.6922099915325995
k = 26  Validation Accuracy: 0.69037538808919
```

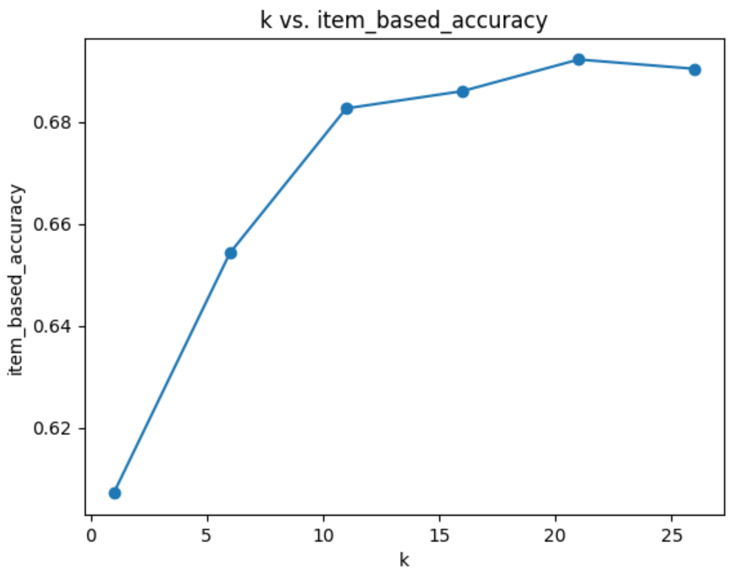

Figure 2: item_based_knn

I choose $k^* = 21$, with the validation accuracy of 0.69221
The correcponding test accuracy is 0.68163

1.1.4 (d)

According to the test accuracy reported, user-based collaborative filtering is slightly better than test accuracy from item-based collaborative filtering.
Besides the test accuracy, we also notice that the computational time of user-based collaborative filtering is much lower than that of item-based collaborative filtering.
Overall, user-based collaborative filtering is the better algorithm.

1.1.5 (e)

1. The computation is very expensive. Comparing with the IRT algorithm, the computational time is longer.

2. Since there are many users and questions, the dimension of data is very high. In this case, most data points have approximately the same distance and KNN is not very representative for prediction.

1.2 Item Response Theory.

1.2.1 (a)

The log-likelihood for all students and questions in the one-parameter IRT model is given by:

$$\begin{aligned}\log p(C|\theta,\beta) &= \sum_{i=1}^{n}\sum_{j=1}^{m}\left[c_{ij}\log\left(\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right)+(1-c_{ij})\log\left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right)\right]\\ &= \sum_{i=1}^{n}\sum_{j=1}^{m}\left[c_{ij}(\theta_i-\beta_j)-\log(1+\exp(\theta_i-\beta_j))\right]\end{aligned}$$

where C is the sparse matrix indicating which questions were answered correctly by which students (i.e., $c_{ij}=1$ if student i answered question j correctly, and $c_{ij}=0$ otherwise).

To compute the derivative of the log-likelihood with respect to θ_i, we have:

$$\begin{aligned}\frac{\partial \log p(C|\theta,\beta)}{\partial\theta_i} &= \sum_{j=1}^{m}\left[c_{ij}-\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right]\\ &= \sum_{j=1}^{m}\left[c_{ij}-p(c_{ij}=1|\theta_i,\beta_j)\right]\\ &= \sum_{j=1}^{m}\left[c_{ij}-\hat{c}_{ij}\right]\end{aligned}$$

where $\hat{c}_{ij}$ is the predicted probability that student i answers question j correctly, given by the one-parameter IRT model.

Similarly, the derivative of the log-likelihood with respect to β_j is given by:

$$\begin{aligned}\frac{\partial \log p(C|\theta,\beta)}{\partial\beta_j} &= \sum_{i=1}^{n}\left[c_{ij}-\frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}\right]\\ &= \sum_{i=1}^{n}\left[c_{ij}-p(c_{ij}=1|\theta_i,\beta_j)\right]\\ &= \sum_{i=1}^{n}\left[c_{ij}-\hat{c}_{ij}\right]\end{aligned}$$

where $\hat{c}_{ij}$ is the predicted probability that student i answers question j correctly, given by the one-parameter IRT model.

The gradient update rules for θ_i and β_j are:

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \eta\sum_{j=1}^{m}\left[c_{ij}-\hat{c}_{ij}\right]$$

$$\beta_j^{(t+1)} = \beta_j^{(t)} + \eta\sum_{i=1}^{n}\left[c_{ij}-\hat{c}_{ij}\right]$$

where η is the learning rate, c_{ij} is the observed response of student i to item j, and $\hat{c}_{ij}^{(t)}$ is the predicted probability of student i answering item j correctly using the current estimates of θ_i and β_j at iteration t.

1.2.2 (b)

Please refer to the python file for implementation.

1.2.3 (c)

We use the hyperparamters: lr = 0.005; iterations = 10 (with batch gradient ascent) and got the following output as result:

```
Iteration: 1   Training NLLK: 39527.68772312085   Validation Accuracy: 0.4875811459215354
Iteration: 2   Training NLLK: 36818.025389521936   Validation Accuracy: 0.6406999717753317
Iteration: 3   Training NLLK: 35500.69642752717   Validation Accuracy: 0.6704769968952865
...
Iteration: 8   Training NLLK: 33029.07726708416   Validation Accuracy: 0.69616144510302
Iteration: 9   Training NLLK: 32773.87149892885   Validation Accuracy: 0.6970081851538245
Iteration: 10   Training NLLK: 32550.30815697092   Validation Accuracy: 0.6991250352808355

Final Validation Accuracy: 0.7012418854078465
Test Accuracy: 0.69037538808919
```

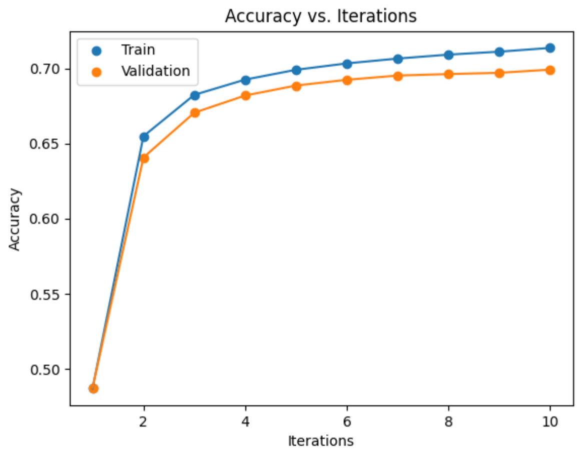

Figure 3: accuracy v.s. iteration

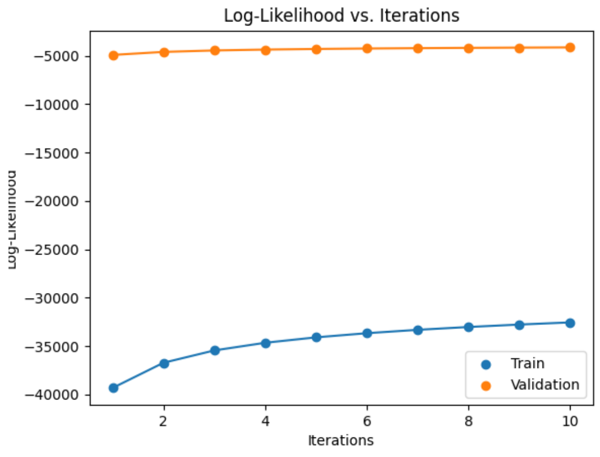

Figure 4: log-likelihood v.s. iteration

1.2.4 (d)

We select question $j_1 = 311$, $j_2 = 817$, $j_3 = 188$. Question 817 has the highest difficulty; while Question 188 has the lowest difficulty (i.e., the value of β_j). Best Theta is our final trained theta for each question.

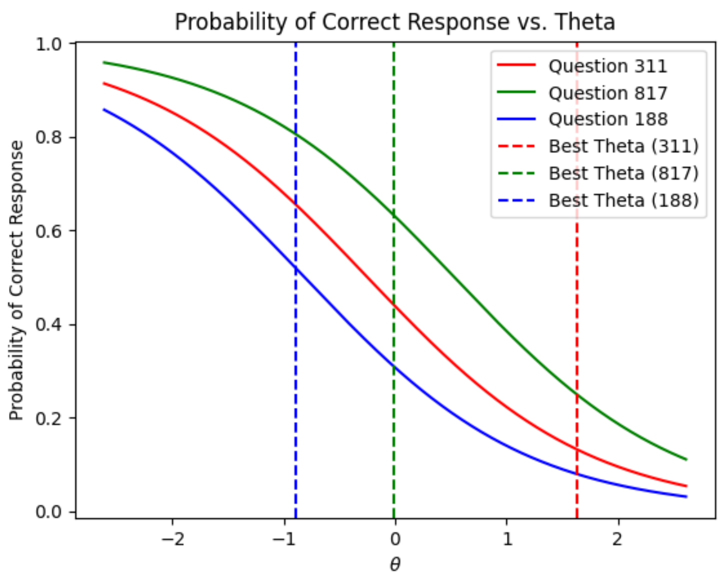

Figure 5: Correct Response v.s. Theta

The shape of the curves depends on the difficulty of the question (β_j). A question with a lower difficulty (i.e., a lower value of β_j) will have a curve that rises steeply and quickly reaches high probabilities of correct response as θ increases. Conversely, a question with a higher difficulty (i.e., a higher value of β_j) will have a curve that rises more slowly and reaches lower probabilities of correct response as θ increases.

The curves represent the relationship between the ability of a student and the probability of correctly answering a given question. They can be used to evaluate the difficulty of the questions and the performance of the students.

1.3 Neural Networks

1.3.1 (a)

Three differences between ALS and neural networks:

1. Their objective function are different. In ALS, the objective function is typically the sum of squared errors between the predicted and actual ratings, whereas in neural networks, the objective function can be chosen based on the task at hand, such as cross-entropy loss for classification or mean squared error for regression.

2. The optimization landscape for ALS is convex with respect to the user and item factors, meaning that there is a unique global minimum. In contrast, the optimization landscape for neural networks is highly non-convex, with many local minima and saddle points, making it more challenging to optimize.

3. They have different complexity. ALS is a simple algorithm that factorizes the user-item matrix into two lower-dimensional matrices. In contrast, neural networks can be arbitrarily complex, with potentially thousands or millions of parameters that can learn highly non-linear relationships between inputs and outputs.

1.3.2 (b)

Please refer to the python file for implementation.

1.3.3 (c)

After tuning hyper-parameters for several times, we choose our final hyper-parameter as $k = 100, lr = 0.01, epoch = 20$.

Our training result is:

```
Epoch: 1  Training Cost: 13449.899532  Valid Acc: 0.626728760937059
Epoch: 2  Training Cost: 12399.049363  Valid Acc: 0.6445103020039514
Epoch: 3  Training Cost: 11609.948041  Valid Acc: 0.6560824160316117
Epoch: 4  Training Cost: 10867.719300  Valid Acc: 0.6677956534010725
Epoch: 5  Training Cost: 10165.494972  Valid Acc: 0.6738639570985041
Epoch: 6  Training Cost: 9494.693832  Valid Acc: 0.6766864239345187
...
...
...
Epoch: 16  Training Cost: 4117.026158  Valid Acc: 0.674428450465707
Epoch: 17  Training Cost: 3783.146533  Valid Acc: 0.6761219305673158
Epoch: 18  Training Cost: 3481.521630  Valid Acc: 0.6761219305673158
Epoch: 19  Training Cost: 3209.066128  Valid Acc: 0.67499294383291
Epoch: 20  Training Cost: 2962.982248  Valid Acc: 0.6721704769968952
```

1.3.4 (d)

The corresponding plot for training loss and validation accuracies are:

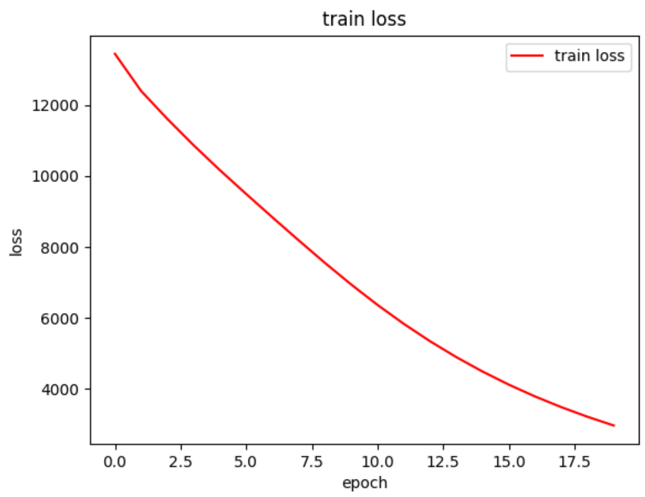

Figure 6: Training Loss

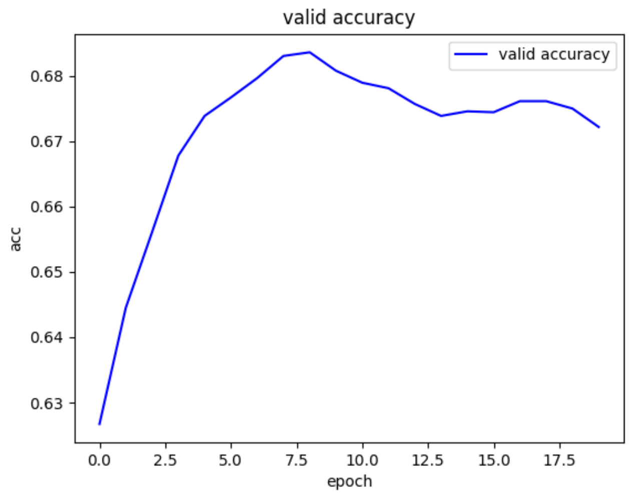

Figure 7: Validation Accuracy

Test Accuracy: 0.6655376799322608

1.3.5 (e)

After tuning the regularization penalty, we choose $\lambda = 0.001$.
Our training result is:

```
Epoch: 1  Training Cost: 13723.521222  Valid Acc: 0.617132373694609
Epoch: 2  Training Cost: 12799.675704  Valid Acc: 0.6240474174428451
Epoch: 3  Training Cost: 12606.185385  Valid Acc: 0.6279988710132656
Epoch: 4  Training Cost: 12470.424174  Valid Acc: 0.630397967823878
Epoch: 5  Training Cost: 12329.719478  Valid Acc: 0.6318092012418854
...
...
...
Epoch: 16  Training Cost: 10446.711379  Valid Acc: 0.6718882303132938
Epoch: 17  Training Cost: 10293.218835  Valid Acc: 0.6734405870731018
Epoch: 18  Training Cost: 10145.020149  Valid Acc: 0.6745695738075078
Epoch: 19  Training Cost: 10001.484535  Valid Acc: 0.6785210273779283
Epoch: 20  Training Cost: 9862.009696  Valid Acc: 0.6788032740615297
Test Accuracy: 0.677109793959921.
```

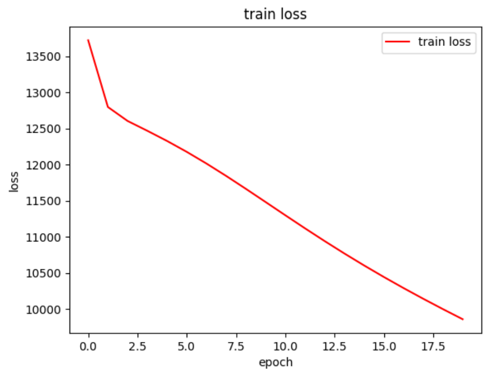

Figure 8: Training Loss after regularization

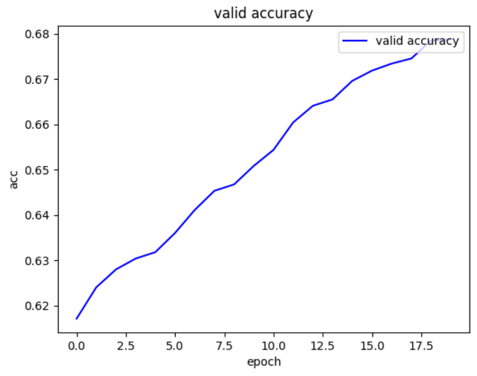

Figure 9: Validation Accuracy after regularization.

While we added a regularization penalty to our model, there is a slight improvement in performance. This may be because our model is overfitting, which is the primary purpose of regularization. Despite this, the improvement on test accuracy is only about 0.011 Further investigation is needed to determine if regularization is a suitable technique for improving our model's performance.

1.4 Ensemble

We decide to apply ensemble methods on Item Response Theory.
We still use the hyper-paramters: lr = 0.005; iterations = 10 (with batch gradient ascent).

Our ensemble is as follow: select and train 3 base models with bootstrapping the training set. To predict the correctness, generate 3 predictions by using the base model and average the predicted correctness.

Our training result is:

```
-------------Training IRT for model 1 start--------------
Iteration: 1   Training NLLK: 39456.84632527884   Validation Accuracy: 0.5045159469376235
Iteration: 2   Training NLLK: 36609.02129153779   Validation Accuracy: 0.6502963590177815
Iteration: 3   Training NLLK: 35197.711704258865   Validation Accuracy: 0.6756985605419137
...
Iteration: 8   Training NLLK: 32458.79462356221   Validation Accuracy: 0.6926333615580017
Iteration: 9   Training NLLK: 32168.172853995085   Validation Accuracy: 0.6934801016088061
Iteration: 10   Training NLLK: 31912.62255366905   Validation Accuracy: 0.6950324583686142
-------------Training IRT for model 1 complete-----------

-------------Training IRT for model 2 start--------------
Iteration: 1   Training NLLK: 39359.00849010213   Validation Accuracy: 0.5018346034434096
Iteration: 2   Training NLLK: 36453.67168875151   Validation Accuracy: 0.6435224386113463
Iteration: 3   Training NLLK: 34969.94427932852   Validation Accuracy: 0.6673722833756703
...
Iteration: 8   Training NLLK: 31990.463496860724   Validation Accuracy: 0.6913632514817951
Iteration: 9   Training NLLK: 31667.073936468114   Validation Accuracy: 0.6926333615580017
Iteration: 10   Training NLLK: 31381.76693310954   Validation Accuracy: 0.6944679650014113
-------------Training IRT for model 2 complete-----------

-------------Training IRT for model 3 start--------------
Iteration: 1   Training NLLK: 39415.75183866743   Validation Accuracy: 0.497883149872989
Iteration: 2   Training NLLK: 36412.56306748571   Validation Accuracy: 0.650578605701383
Iteration: 3   Training NLLK: 34875.71977964353   Validation Accuracy: 0.6780976573525261
...
Iteration: 8   Training NLLK: 31691.053969307362   Validation Accuracy: 0.6929156082416031
Iteration: 9   Training NLLK: 31334.361775911766   Validation Accuracy: 0.6929156082416031
Iteration: 10   Training NLLK: 31018.07208181133   Validation Accuracy: 0.6929156082416031
-------------Training IRT for model 3 complete-----------

Final Validation accuracy: 0.6984194185718318
Test accuracy: 0.6881174146203782
```

According to our result, The individual model achieved a final validation accuracy of 0.701 and a test accuracy of 0.690, while the ensemble model achieved a final validation accuracy of 0.698 and a test accuracy of 0.688.

Based on the results presented, the individual model achieved slightly better performance than the ensemble model in terms of the validation and test accuracy. This outcome might be due to the small number of individual models in the ensemble, where the increase in the diversity of the models may not be significant enough to improve the performance. Additionally, the individual model might have learned more robust features than the individual models in the ensemble due to the larger amount of training data available to it.

However, the difference in the performance between the individual and ensemble models is very small (0.003 for validation), and in practice, the ensemble model may still be preferred due to its increased reliability, especially when it comes to generalization.

2 Part B

2.1 Motivation

In our neural network implementation, we utilize an autoencoder. An autoencoder is an unsupervised learning technique that aims to learn a compressed representation of input data. Specifically, in our implementation, it aims to learn a compressed representation of the user response data.

Since the autoencoder is an unsupervised learning technique, the reconstruction error is used as the evaluation metric to assess the model's performance. However, relying solely on the reconstruction error on the training set can result in the problem of overfitting.

In the absence of L_2 regularization, the training cost decreases over epochs, indicating that the model is learning to fit the training data better. However, the validation accuracy does not consistently increase, which suggests that the model may be overfitting to the training data.

To improve the accuracy, we made two significant changes to our architecture: Cross Validation and Dropout.

2.2 Description

2.2.1 Cross Validation

Cross validation is a popular method for evaluating the performance of machine learning models. The basic idea of cross validation is to partition the data into a training set and a validation set. The training set is used to train the model, while the validation set is used to evaluate the model's performance. Cross validation can be used to estimate the generalization error of a model, which is the error that the model would make on new, unseen data.

Kohavi et al. (1995) introduced the k-fold cross validation method, which is a widely used approach for evaluating machine learning models. In k-fold cross validation, the data is divided into k equally sized folds, denoted as $D = d_1, d_2, ..., d_n$, where n is the size of the dataset. Then, for each iteration i, a fold d_i is used as the validation set, and the remaining folds are used as the training set. This process is repeated k times, with each fold used once for validation. The results from the k validation sets are then averaged to obtain an estimate of the model's generalization error.

Mathematically, k-fold cross validation can be expressed as follows:

$$D^i_{train} = \{d_1, d_2, ...d_{i-1}, d_{i+1} ..., d_n\}$$

$$\epsilon_{gen} = E_{P_{val}}[L(f(x;\theta), y)]$$

where the training set for iteration i is denoted as $D^{(i)}_{train}$, P_{val} is the probability distribution over the test data, f is the model, θ is the model parameters, and L is the loss function.

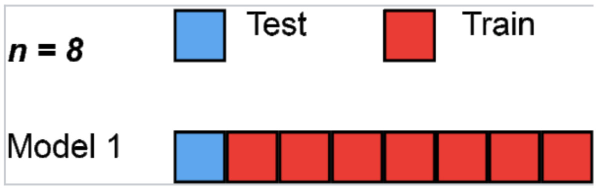

Figure 10: Cross Validation Visualization Source

n our implementation, we utilized the KFold function from the scikit-learn library to perform k-fold cross validation. Specifically, we divided the data into k folds and trained the model on k-1 folds while using the remaining fold and a separate validation dataset for evaluation. This approach was chosen to maximize the efficiency of our dataset and obtain a more comprehensive evaluation of our model. The process was repeated k times. Finally, we evaluated the performance of our model on the original validation dataset (as well as the test dataset) to obtain an measure of its generalization ability.

2.2.2 Dropout

Dropout is a regularization technique that has been shown to be effective in preventing overfitting in deep neural networks. The basic idea of dropout is to randomly drop out (i.e., set to zero) some of the neurons in the network during training. By doing so, dropout forces the network to learn more robust features, which can improve its generalization performance.

Srivastava et al. (2014) introduced dropout as a regularization technique for deep neural networks. They showed that dropout can improve the generalization performance of deep networks on a variety of tasks, including image classification, speech recognition, and natural language processing. Dropout has since become a widely used regularization technique in deep learning.

Mathematically, dropout can be expressed as follows:

Let x be the input to a layer in the network, and let d be a dropout mask where $d = 0$ with probability p and $d = 1$ with probability $(1 - p)$. In other words, d follows Bernoulli(p). Then, the output y of the layer with dropout can be expressed as:

$$y = d \odot (\frac{x}{1-p})$$

where $\odot$ denotes element-wise multiplication.

During testing, the dropout mask is not applied, and instead the output is scaled by $(1 - p)$ to ensure that the expected output remains the same. This is expressed as:

$$y = (1 - p) \odot x$$

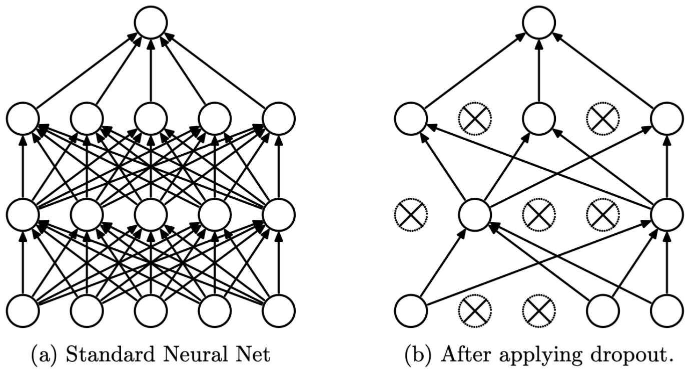

Figure 11: Dropout Visualization Source

In our implementation, we added dropout regularization to the AutoEncoder class by inserting '$nn.Dropout(p = 0.3)$' layers between the two linear layers. This ensures that 30% of the neurons are randomly dropped out during training, which helps to reduce overfitting and improve the generalization performance of the model.

2.2.3 Discussion

Cross validation and dropout are two widely used techniques in machine learning for preventing overfitting. These techniques have been shown to be effective in improving the generalization performance of machine learning models, including neural networks. By incorporating dropout regularization and k-fold cross validation, we expect our proposed method to perform better than the original algorithm by reducing overfitting and improving the generalization performance.

2.3 Model Architecture

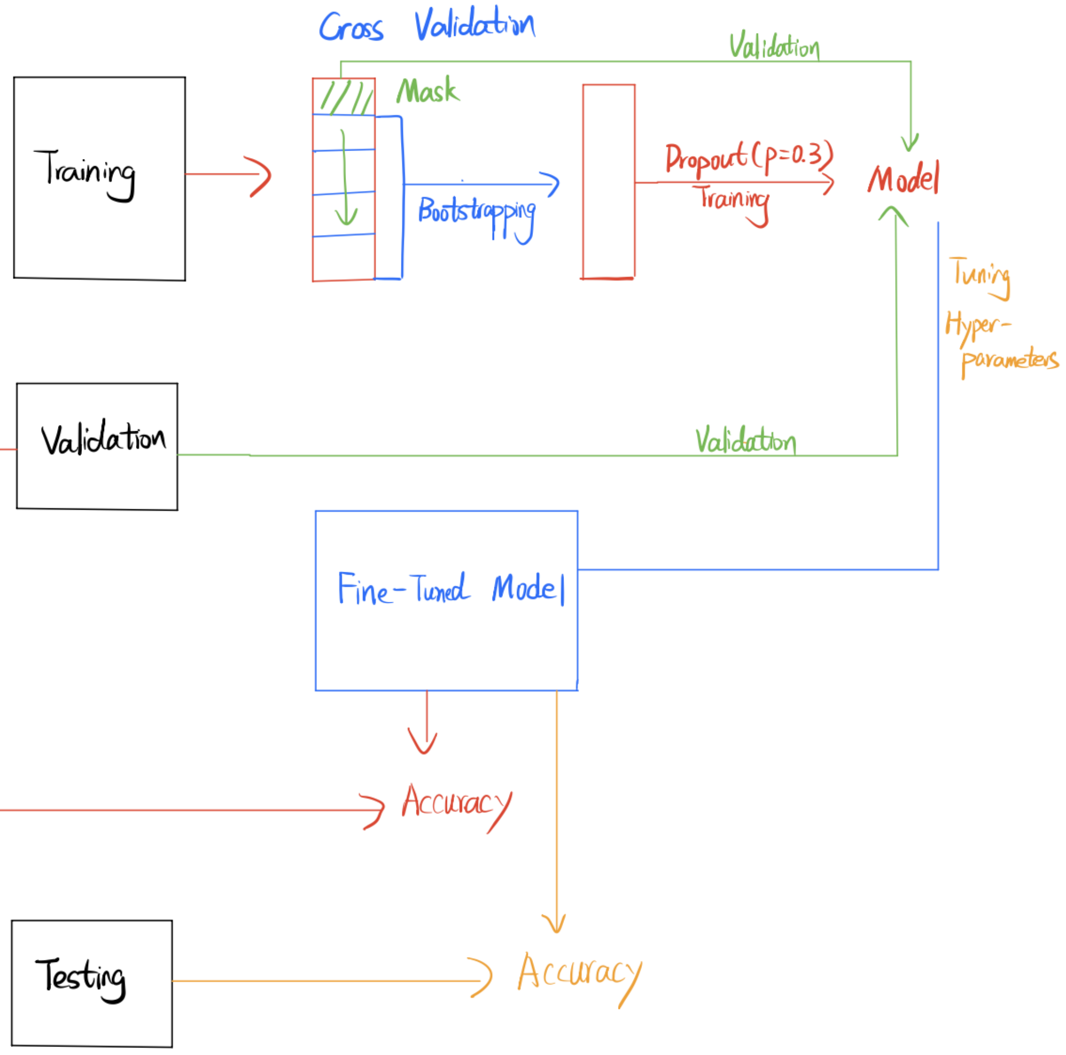

Figure 12: Architecture

The architecture of our model is presented in Figure 12. To improve the generalization performance of the model, we first perform k-fold cross validation on our training dataset. For each iteration, we set aside one fold for validation and use the remaining k-1 folds for training. To decrease the variance, we apply the bootstrapping method, which creates a new dataset by randomly sampling with replacement from the original training dataset.

Next, we apply dropout regularization to the model during training with the probability 0.3. This technique randomly drops out a specified percentage of neurons in the network to force the model to learn more robust features and reduce overfitting.

To maximize the efficiency of our dataset, we use the k-th fold training dataset and the validation dataset for validation. We then tune the hyperparameters based on the validation accuracy, enabling us to obtain a comprehensive overview of the generalization of our dataset and to make comparisons with our baseline model.

Finally, we evaluate our model on the original validation dataset and the testing dataset to obtain the final accuracy result.

2.4 Experiment

2.4.1 Training Result

We use the neural network model without L_2 regularization (part 1.3.3) as our baseline model. According to the training result of that model, the validation accuracy doesn't show a consistent increase, which suggests that the model may be overfitting to the training data.

Therefore, we train three other models for comparison here: model_with_dropout, model_with_cross_validation, model_with_both.

We also show a part of output for our last model (model_with_both) for reference.(red lines in figures represent a new k-fold iteration begins)

```
Iteration:  1
Epoch: 1  Training Cost: 10288.807813  Valid Acc: 0.6213660739486311
Epoch: 2  Training Cost: 10101.897815  Valid Acc: 0.6217894439740334
...
...
...
Iteration:  5
Epoch: 1  Training Cost: 7783.036713  Valid Acc: 0.672876093705899
Epoch: 2  Training Cost: 7668.930660  Valid Acc: 0.6730172170476997
...
Epoch: 6  Training Cost: 7295.345186  Valid Acc: 0.6768275472763196
Epoch: 7  Training Cost: 7208.977428  Valid Acc: 0.679109793959921
The validation accuracy of the model on the validation set is:  0.6797911374541349

The final validation accuracy of the model is:  0.6797911374541349
The testing accuracy of the model is:  0.6821902342647473
```

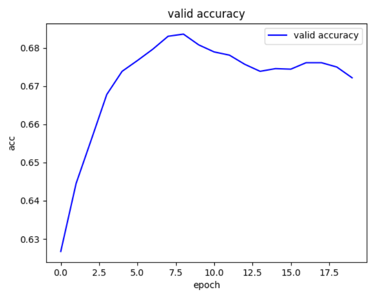

Figure 13: baseline model

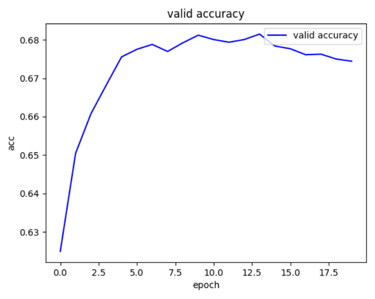

Figure 14: model_with_dropout

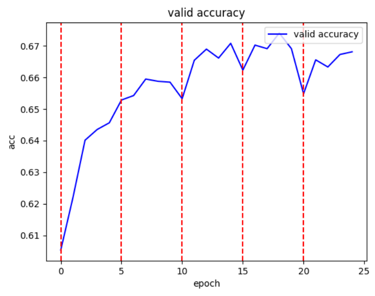

Figure 15: model_with_cross_validation

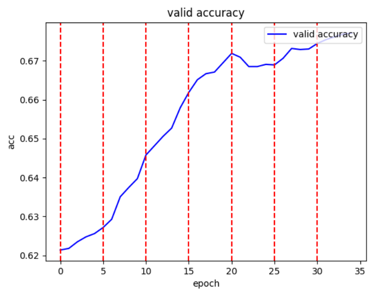

Figure 16: model_with_both

2.4.2 Model Comparison

Model	Final Validation Accuracy	Test Accuracy
baseline model	0.672	0.666
model_with_dropout	0.674	0.675
model_with_cross_validation	0.666	0.669
model_with_both	0.680	**0.682**

Based on the table and figures presented above, we can observe the effectiveness of both cross-validation and dropout techniques in preventing overfitting. Among the three models compared against our baseline model without L2 regularization, the model with only dropout shows a slower decrease in validation accuracy after epoch 10 and achieves a slightly improved final validation accuracy and test accuracy. The model with only cross-validation, on the other hand, has a validation accuracy that decreases slightly at the beginning of each new k-fold iteration, but then increases steadily within the same fold. Although its final validation accuracy is slightly lower than our baseline model, it achieves a higher test accuracy, suggesting better generalization performance.

Finally, the model with both dropout and cross-validation achieves the highest final validation accuracy and test accuracy, confirming the effectiveness of combining both techniques to enhance the model's performance.

2.5 Limitation

Despite their effectiveness in preventing overfitting, cross-validation and dropout have certain limitations. Our experiment results showed that applying cross-validation and dropout did not result in a significant improvement in accuracy. However, these techniques can still be useful if used correctly.

The limitation of k-fold cross-validation is that its effectiveness depends on how representative the folds are of the entire dataset. If the dataset has imbalances, such as class imbalance, or a temporal component where the distribution of data changes over time, k-fold cross-validation may not accurately estimate the generalization error of the model. In our experiment, one possible solution is to combine the training, validation, and testing datasets and separate them again to ensure similar distributions. This can maximize the efficiency of cross-validation.

The limitation of dropout is that its performance is highly dependent on the dropout rate. Setting the dropout rate too high can lead to underfitting, while setting it too low can lead to overfitting. Therefore, finding the optimal dropout rate requires more experimentation. In our experiments, we can try tuning the hyperparameter of dropout rate (e.g., 0.1-0.5). Moreover, dropout may not be effective for all types of neural networks or datasets, and may need to be combined with other regularization techniques (e.g., L_2 regularization) to achieve the best results. We can try combining dropout with L_2 regularization to improve our model's performance.

While k-fold cross-validation and dropout can help prevent overfitting, they do not guarantee that the model will generalize well to new data. Other factors, such as data quality and problem complexity, can also affect the model's performance. Therefore, further experiments are needed to ensure that our model can perform well on new data.

3 Reference

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI) (Vol. 14, No. 2, pp. 1137-1145).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.