Grade Distribution Prediction with Deep Learning

**Group 9**

**Xi Teng**

**Dou Hoon Kwark**

**Hongying Zhu**

**Yixuan Li**

# Overview

We trained models on the UIUC GPA dataset for grade distribution prediction as the grade distribution of a class plays an important role in helping students decide whether to register for the class. It provides valuable insights into the course’s difficulty level, and thus students can make informed decisions about their academic planning and manage their workload effectively, which eventually helps them choose courses that align with their goals and capabilities. We performed EDA first and moved on to model training. We show good improvement from the baseline model and how each feature would play a role in model training throughout feature important analysis. The link to the presentation video is available: <https://drive.google.com/file/d/1JKrgKTq0DWJJ8q98rBhDsJONGfXGApdy/view?usp=sharing> The link to project GitHub repository is also available:

<https://github.com/Dou11112/DL-Project-Group-9>

# Exploratory Data Analysis

## Data pre-processing

From the UIUC GPA dataset, we found that there are 69069 data points with 22 available features:

['Year', 'Term', 'YearTerm', 'Subject', 'Number', 'Course Title', 'Sched Type', 'A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D+', 'D', 'D-', 'F', 'W', 'Primary Instructor'].

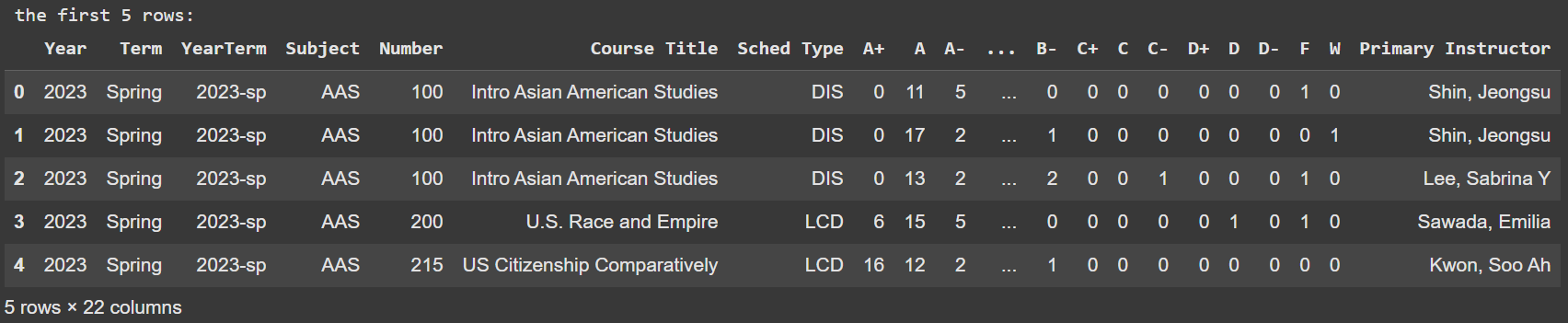


Figure 1

We identified and split the features into two categories: **categorical** features and **numerical** features. For **categorical** features, we have Year, Term, YearTerm, Subject, (Course) Number, Course Title, Schedule Type, and Primary Instructor. For **numerical** features, we have all the grades, including A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F, and W.

We then identified **categorical** features with too many unique values, and found that there are 9313 and 5813 unique values for Primary Instructor and Course Title, respectively. Since we wanted to apply one-hot encoding to convert the categorical features, we decided to drop them as it would lead the model to require too high dimensional complexity to achieve a good performance. Additionally, feature Number can be inferred from the combination of feature Subject and feature Number. On top of that, we also dropped the feature YearTerm as there are separate features, Year and Term. Then, we apply the one-hot encoding to the categorical features. Figure 2 is the expanded dimension of the categorical features with the one-hot encoding applied.

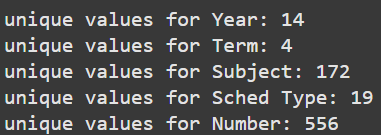


Figure 2

For the **numerical** features (grades), in order to represent the grade distribution, we computed the total number of the grades in the class, divided each grade by the sum, and re-assign them to each value. The steps can be expressed as follows:

where represents the value of the feature.

Therefore, after scaling the grades, the grade distribution data are in the following form:

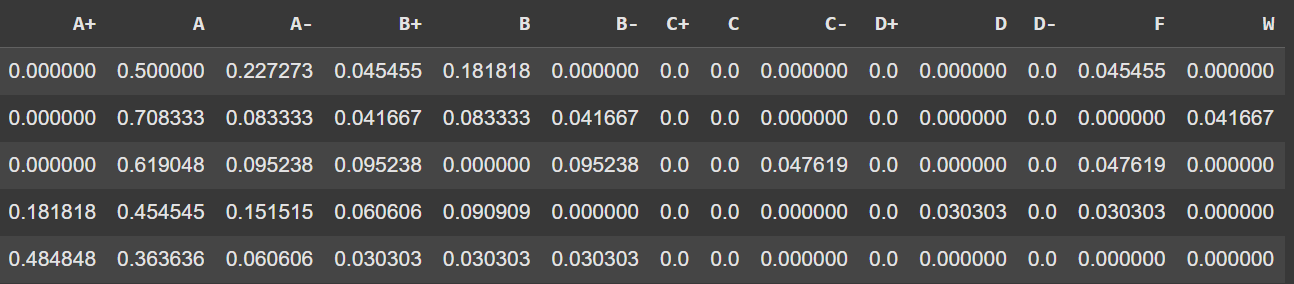


Figure 3

## Exploratory Data Analysis (EDA)

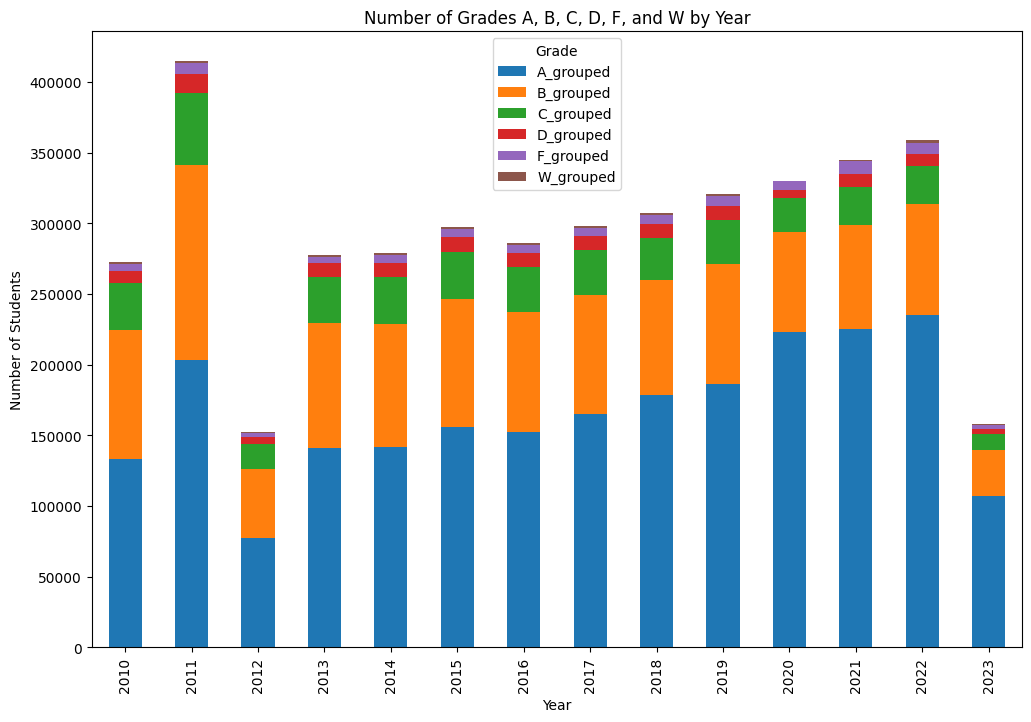
Then, we studied the correlations between features and grade distribution. 

Figure 4

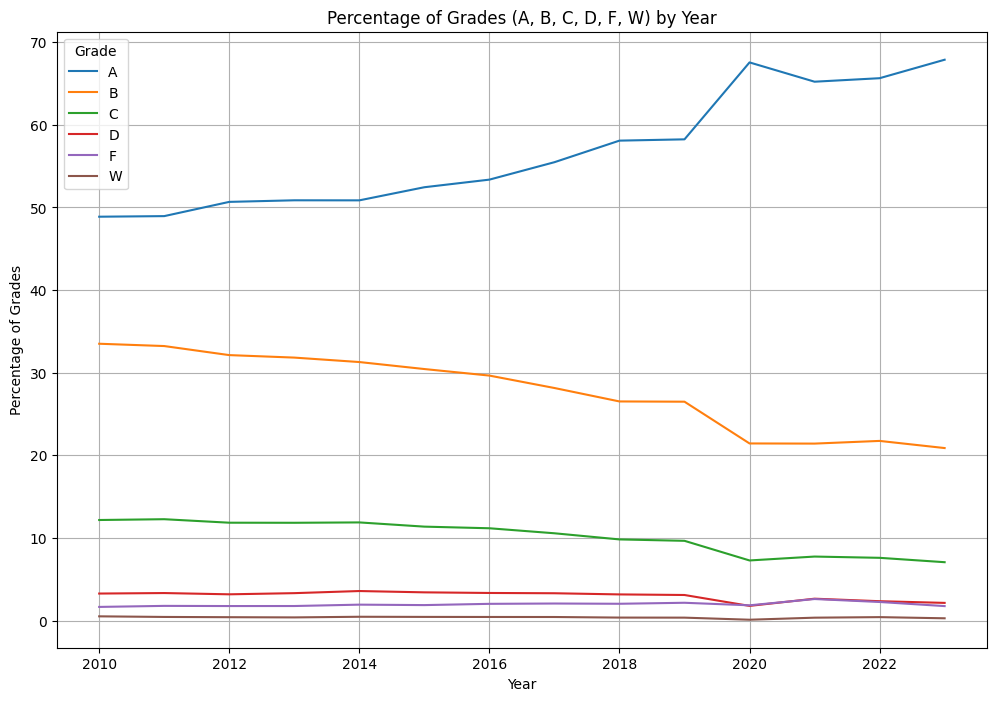
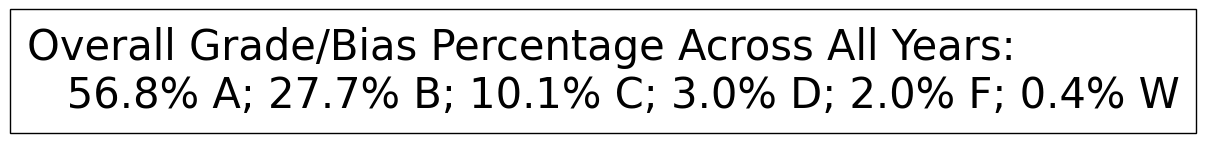


Figure 5

For example, the two plots (Fig. 4 and 5) above show the trend of grouped A (group of A+, A, A-) increases over the year, while it shows the decrease of the trends in other grades. We can make sense of it, as it should be a known fact that many universities including UIUC have experienced “the grade inflation.” Therefore, from the investigation of bias data in the dataset, we can observe grade bias across all years as follows:



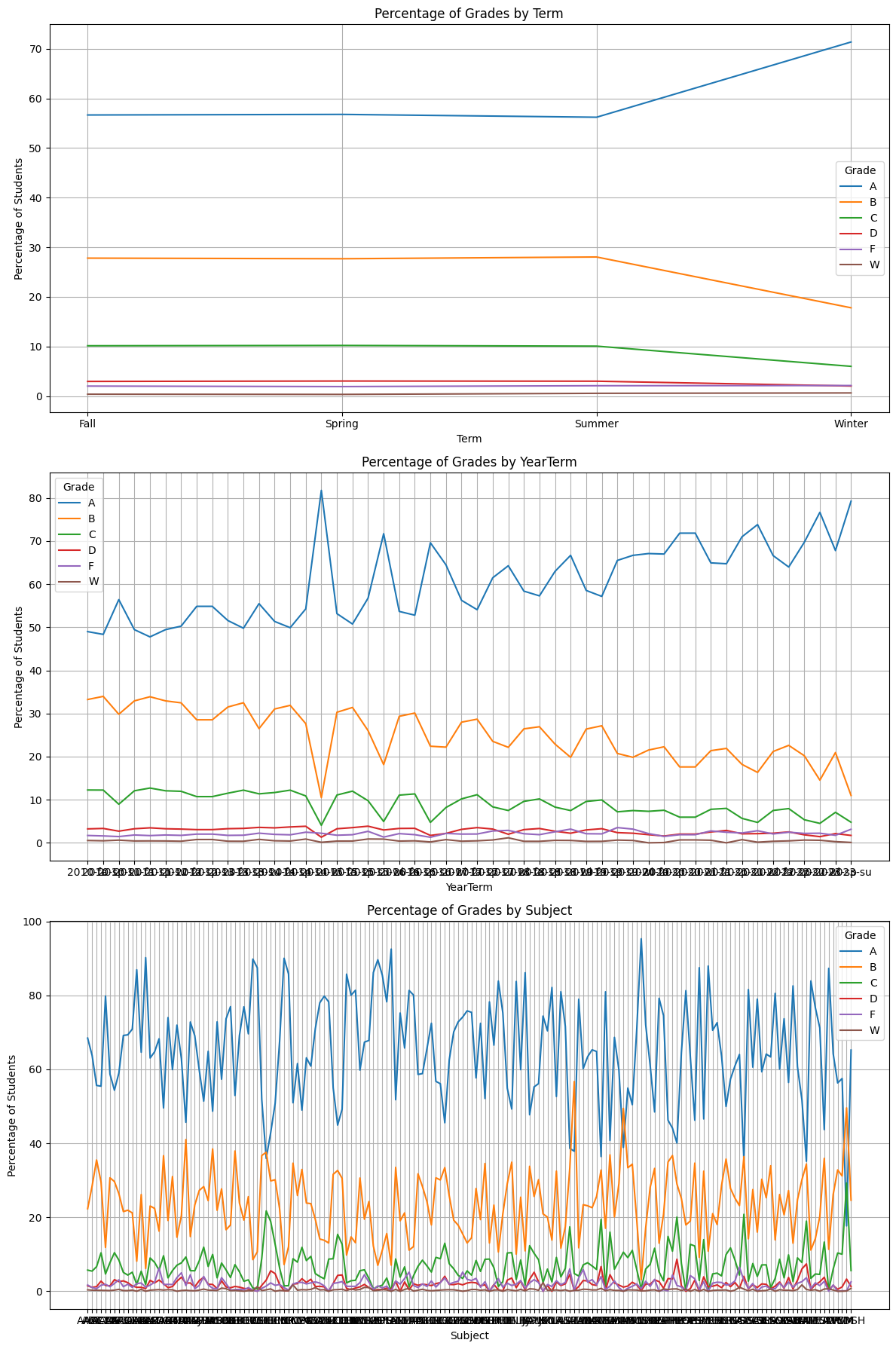


Figure 6

We further performed EDA on other features with the grade distribution. The second plot in Figure 6 is a detailed version of the previous figures on the grade distribution and the year feature. It shows an overall increase of the trend of the grouped grade A and decrease of the other grades with huge fluctuation in winter, which can also be observed in the first plot in the figure above, which helps us understand how knowing term or year features would improve the model performance.

Then, we also studied the relationship between course schedule type and grade distribution. As we can notice in the plot below, each course type also has a different grade distribution, which thus could also help the model perform better.

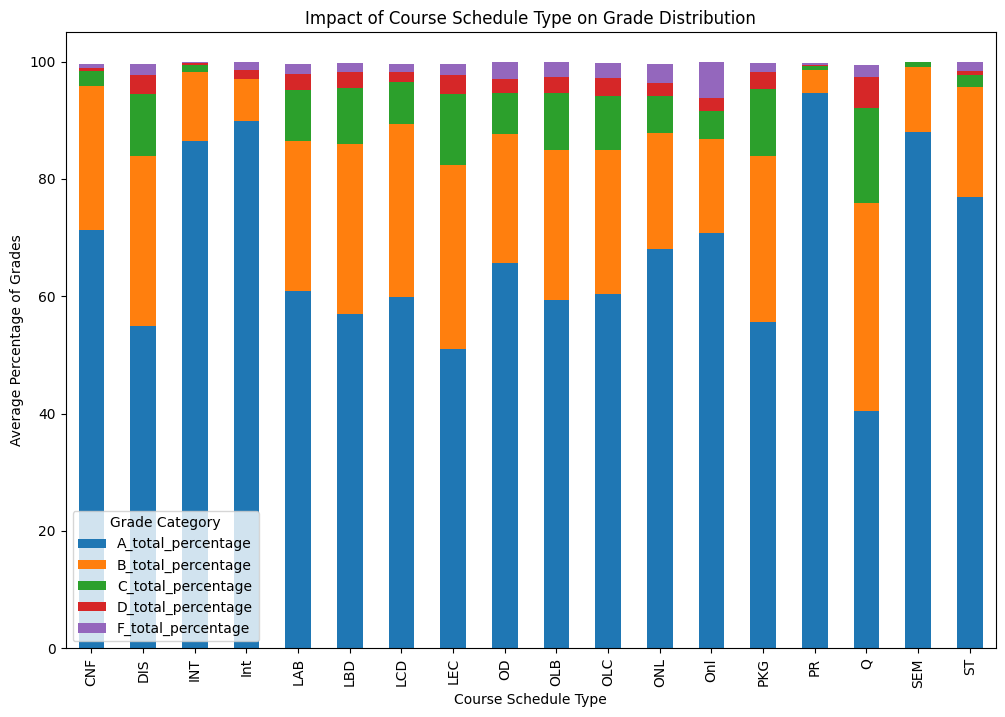


Figure 7

With full consideration of the EDA and careful investigation we performed, we did full pre-processing and data-cleaning, which resulted in a working dataset with 40387 data points.

# Baseline Model

## Linear regression model

With the preprocessed dataset, we first split the dataset in 0.70:0.15:0.15 train/val/test manner. Then, we trained the linear models for grade distribution prediction. We first trained an initial model on debugging dataset (5% of the main working dataset) to make sure that the model properly learned in the given settings, and moved on to train the model on working dataset. To train the simple linear regression (LR) model, we designed each model to predict the percentage of one grade. For example, LR model 1 is trained to predict the percentage of A+ for the given class features, LR model 2 is for A, and so on. We trained two main linear regression models with different regularizers as follows:

'Ridge Regression': 'alpha': [0.1, 1.0, 10.0, 100.0]

'Lasso Regression': 'alpha': [0.001, 0.01, 0.1, 1.0]

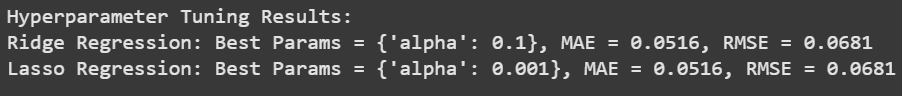


Figure 8

As we can see, we could not find any significant differences between the models and chose Ridge Regression model for the final evaluation on the test set. From the test set, the model prediction and GT comparison is as follows:

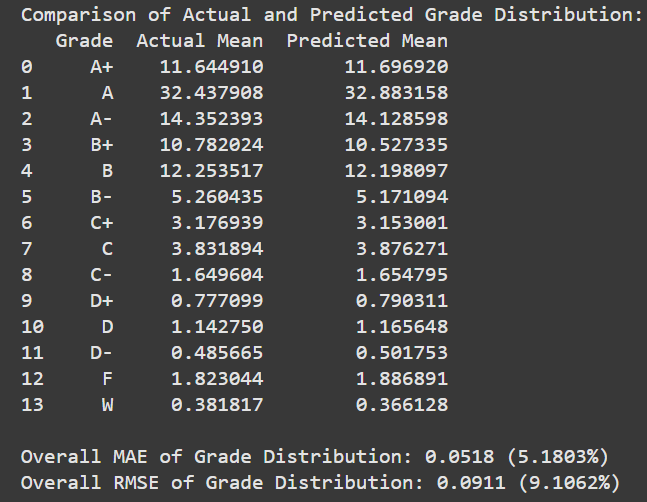


Figure 9

We see on average that the model prediction is 5% (MAE: 0.0518) off from the ground-truth values, which also tells that the grade distribution prediction is 5% on average. Also, the model prediction result means that if we have a class of 100 students, then the model prediction tells that there will be 32.8 students who would receive an A, but in reality, 32.4 students received an A. It is also noticeable that RMSE being 0.0911 (0.1062%) is almost doubled as MAE being 0.0518 (5.1803%). It could be affected by the problem design we made as we train models for each grade percentage prediction.

# Neural Network Model

Similar to Baseline Learning, we start off by training models on debugging dataset in the same manner described in the Baseline Learning Model.

Data Preprocessing involved loading two datasets and transforming categorical features into a numerical format using one-hot encoding. Features like Year, Term, Subject, Sched Type, and Number were converted into fixed-size vectors to ensure consistency across all samples. The dataset was then split into training, validation, and test sets, with a ratio of 70%, 15%, and 15%, respectively. The training set was used to fit the model, the validation set to tune hyperparameters and prevent overfitting, and the test set to evaluate the model's final performance.

The training process began with designing a two-layer neural network model to predict student grades. Each categorical feature was processed independently through fully connected layers with 64 neurons, ReLU activation, and a dropout rate of 0.3 to prevent overfitting. The outputs were concatenated and passed through two fully connected layers, producing predictions for each grade category. The model was trained separately for each grade using the Adam optimizer with a learning rate of 0.0001 and Mean Squared Error (MSE) as the loss function. After achieving an overall train loss of 0.0070 and validation loss of 0.0082, a dry-run on the debugging dataset confirmed that the model effectively minimized errors without significant overfitting. The steady convergence of training and validation losses demonstrated the robustness of the model design and the effectiveness of the training process. (Figure X)

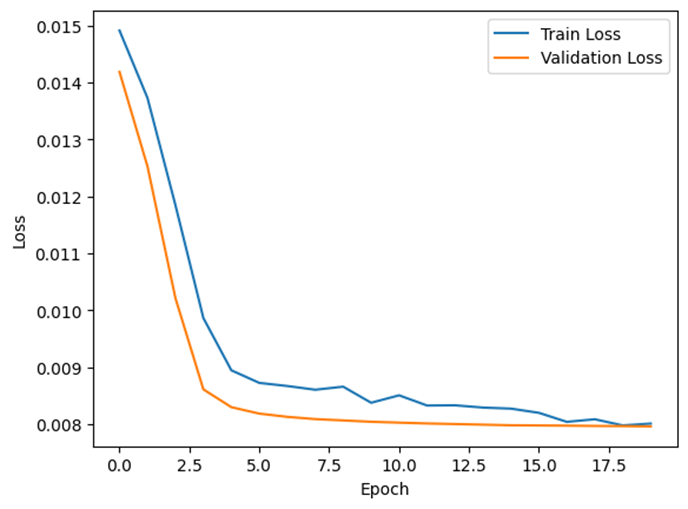


Figure 10

After verifying the model design with a dry-run on the debugging dataset, the model was trained on the main dataset with various batch sizes to analyze their effect on training convergence. The batch sizes tested were 16, 32, 64, 128, 256, and 512, while using the Adam optimizer with a learning rate of 0.00001 and weight decay to improve regularization. The feature "Course Title" was dropped to prevent overfitting. The results indicated that smaller batch sizes, such as 16 and 32, led to faster convergence due to more frequent updates to the model's parameters. In contrast, larger batch sizes, like 256 and 512, required more epochs to converge and showed higher validation loss initially, reflecting slower learning. This behavior highlights that smaller batch sizes are beneficial for datasets where the model tends to overfit quickly, while larger batch sizes offer smoother but slower training dynamics. (Figure 10)

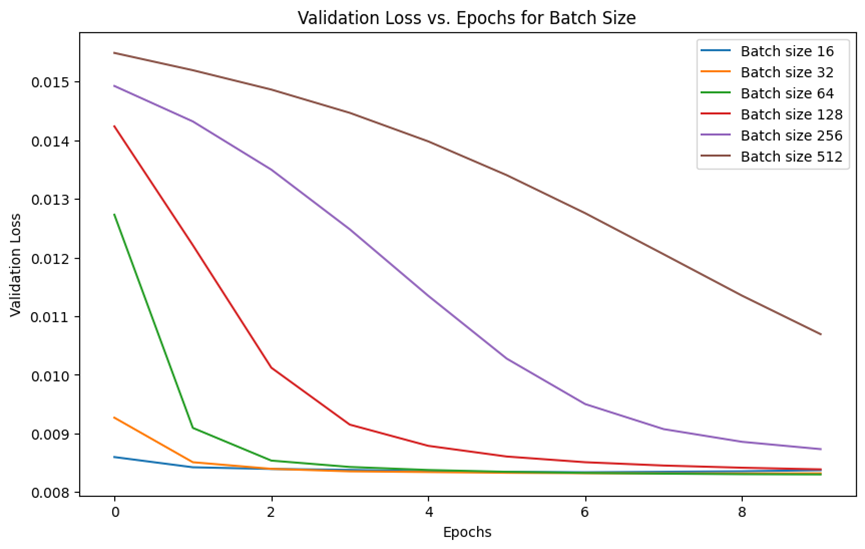


Figure 11

To improve the model's performance, an optimizer comparison was conducted. Following the analysis of batch sizes, we investigated the effects of different optimizers — SGD, Adam, and RMSprop — using a batch size of 64 to determine which optimizer is most effective for our dataset. The models were trained using the same architecture, learning rate, and hyperparameter settings to ensure a fair comparison. Adam's ability to adapt learning rates for each parameter made it particularly effective for this dataset, allowing the model to find optimal parameters efficiently.

An extensive hyperparameter search was conducted to further optimize the neural network’s performance. The search explored various combinations of dropout rates (0.2, 0.3, 0.5), weight decay values (0, 0.0001, 0.001), patience for the learning rate scheduler (2, 3, 5), and hidden layer sizes (64, 128, 256). The Adam optimizer with a learning rate of 0.00001 was used for training, and a scheduler reduced the learning rate when the validation loss plateaued. Models were trained for a maximum of 10 epochs, with early stopping implemented to prevent overfitting.

The plot of validation loss across epochs for different hyperparameter combinations revealed significant variations in model performance. The results demonstrated that the combination of dropout=0.5, weight decay=0, patience=5, and hidden size=256 achieved the lowest validation loss. This optimal configuration effectively balanced regularization and model complexity, allowing the network to generalize well to unseen data.

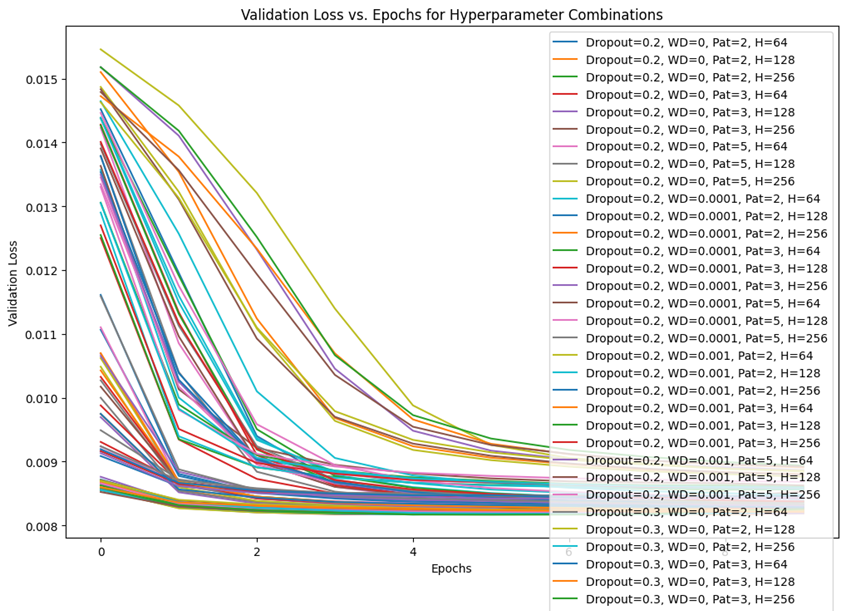
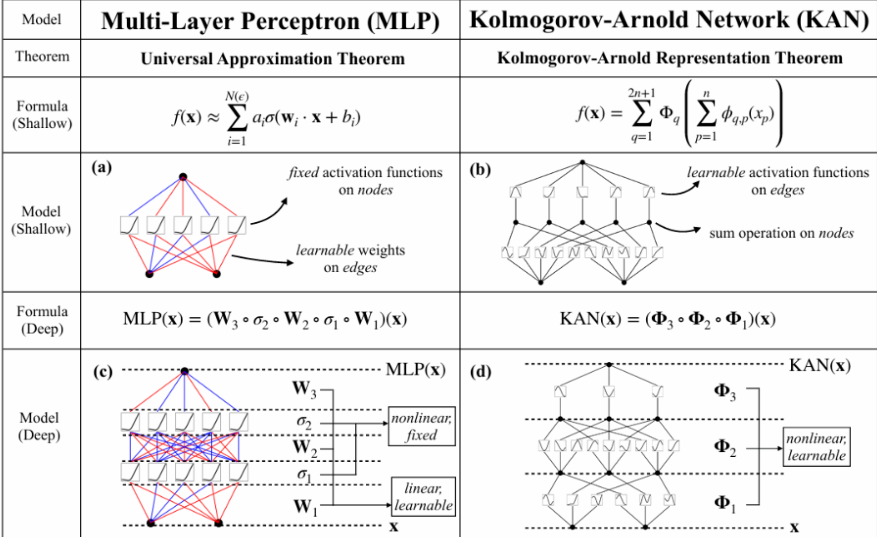


Figure 2

With the best hyperparameters—dropout rate of 0.5, weight decay of 0, patience of 5, and hidden size of 256—the model was re-trained on the main dataset, achieving a final validation loss of 0.0082. When evaluated on the test dataset, the model delivered a Mean Absolute Error (MAE) of 0.0523 and a Root Mean Squared Error (RMSE) of 0.0685, representing a significant improvement over the baseline results of 0.0847 for MAE and 0.1475 for RMSE. As is well known, MAE measures direct numerical differences, while RMSE is more responsive to outliers; the results demonstrated a 65% reduction in outliers, indicating enhanced prediction accuracy and robustness.

The analysis of batch size showed that smaller batch sizes led to faster convergence, suggesting that the dataset is relatively easy to predict. Additionally, the investigation of different optimizers revealed that the Adam optimizer was superior to SGD and RMSprop in finding optimal parameters efficiently. These findings highlight the importance of selecting the right optimizer and performing thorough hyperparameter tuning to achieve a robust and accurate neural network model.

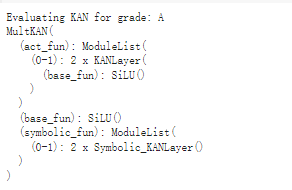
Kolmogorov–Arnold Neural Network (KAN)



We use KAN（Kolmogorov-Arnold Networks） a new neral network structure to build the Grade distribution model.

In traditional deep learning MLP (Multi-Layer Perceptron), the connections between neurons are typically represented by real values, which signify the strength of the connection (weights). Each neuron also has a non-linear "activation function," such as ReLU or Sigmoid. In simple terms, it is "**add first, then transform.**"

In contrast, in KAN (Kolmogorov-Arnold Network), the neurons are "perceptionless." The activation function is moved from the node to the edges, meaning the process is "**transform first, then add.**"

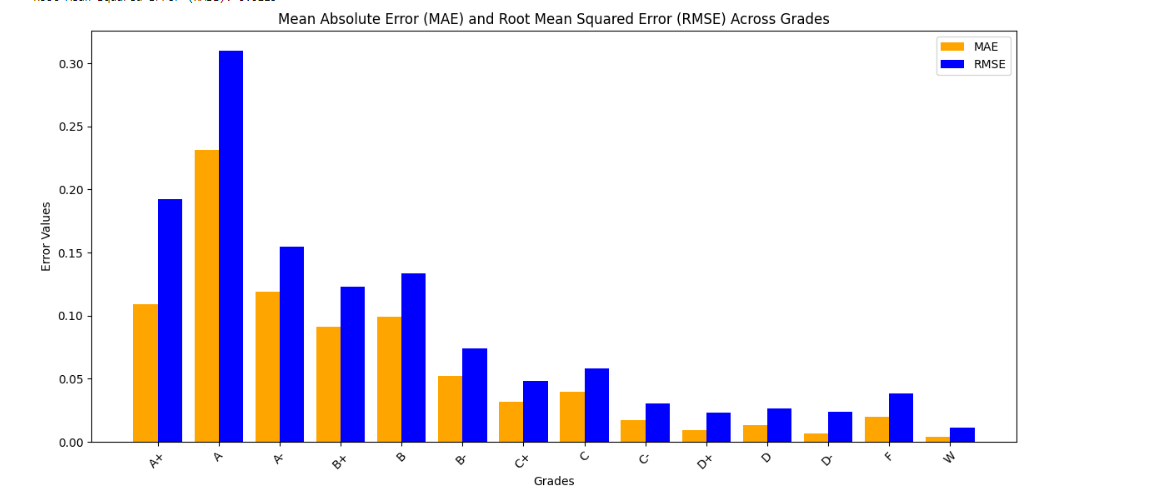


Although the model design only shows a single hidden layer, making the structure relatively simple, its uniqueness lies in the following:

Combining the advantages of symbolic representation and neural networks.

Leveraging KANLayer and Symbolic\_KANLayer to enhance the function approximation capability and interpretability.

Result



.

| Average MAE | Average RMSE |
| --- | --- |
| 0.0499 | 0.0722 |

Using a simpler structure can achieve MAE and Root Mean Squared Error results better than those of an MLP (Multi-Layer Perceptron). However, as seen from the predicted grade distribution in the figure, there are still noticeable differences. This is because even though the K-A (Kolmogorov–Arnold) neural network has a more streamlined structure and fewer parameters. training the univariate functions ψ\psi and ϕ\phi may involve complex optimization challenges, making the optimization process more difficult.

**Advantages**

1. **Based on the Kolmogorov–Arnold Representation Theorem**:
   * Decomposes complex multivariate functions into simple univariate transformations and additive combinations, making the model structured and easy to understand.
2. **Scalability for High-Dimensional Problems**:
   * Effectively handles high-dimensional problems through layered design, reducing computational complexity and improving performance in such scenarios.
3. **Strong Interpretability**:
   * Each layer corresponds to a clear mathematical operation, enabling straightforward analysis and validation of the model’s internal logic.

**Disadvantages**

1. **Sensitive to Hyperparameters**:
   * Requires careful tuning of grid sizes, layer configurations, and regularization parameters, which can make the optimization process challenging and time-consuming.
2. **High Computational Cost**:
   * Training and inference can demand significant computational resources, especially for large datasets or complex models.

# 

# 

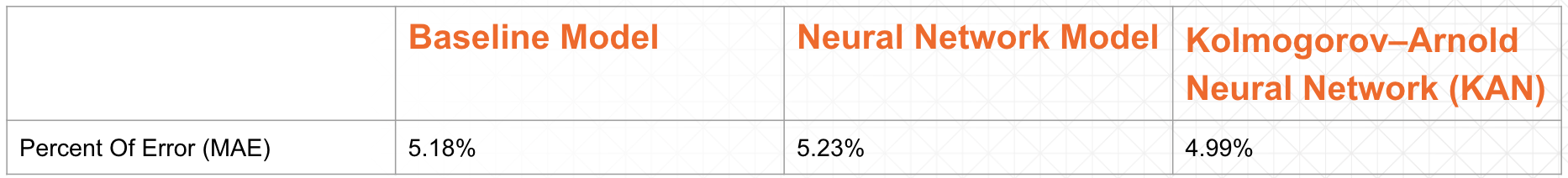
# 

# 

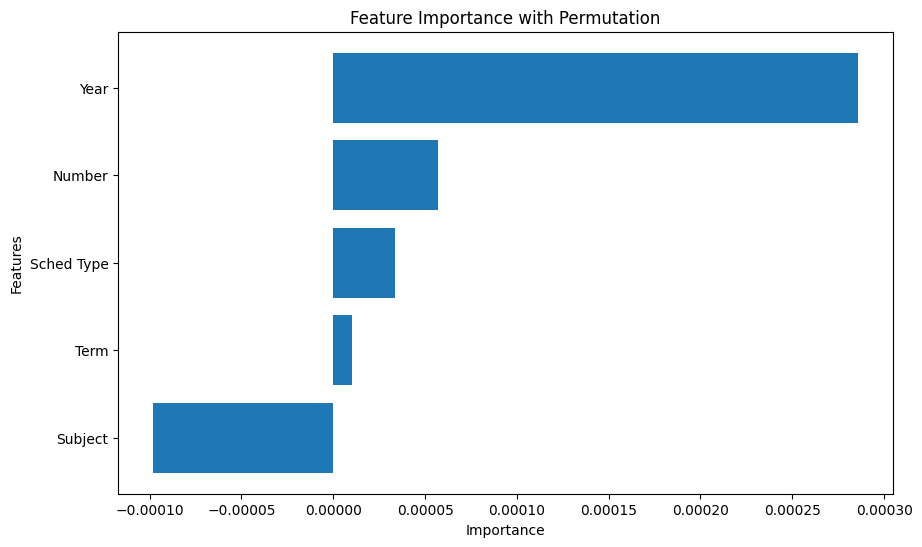
# 

# Conclusion

For this project, we aimed to predict the grades distribution based on easily accessed course information such as year, term, course number, and so on. We started by exploratory data analysis to visually look at the trends of each feature. And then we proceeded to build a simple linear regression model which proved to be a decent model given that the MAE is just 5.18% on average. We then started building two deep learning models, the neural network and the Kolmogorov–Arnold Neural Network (KAN) Model. The neural network model showed a worse MAE result with 5.23% on average and the KAN model showed a slightly better performance of 4.99% MAE on average.



We also tested the feature importance with permutation on those features, and found that year and course number features have more determining value than the rest features. We think this result makes economic sense, as there has been a potential trend that there is grade inflation over years. As for the course number, it is no mystery that the course number level represents the difficulty level of a course. For example, introductory courses are usually 100 and 200 level classes. And advanced courses are usually 300 and 400 classes.



For practical purposes, we recommend the stakeholders to adopt the KAN model as it gave the lowest percent of error among the 3 models. The stakeholders can feed inputs such as year, term, and course number into the KAN model, and then the model will output the grade distribution as result. Stakeholders can predict the proportion of students getting the specific grade for each different class. Given a course, if the actual percentage of students getting an A is 31%, then our model will predict the percentage of students getting an A to be between 29.5% and 32.6% on average.