COMP9417 Machine Learning Project

CommonLit - Evaluate Student Summaries

Automatically assess summaries written by students in grades 3-12

Competition From Kaggle

Launch date: Jul 13, 2023

Website:
https://www.kaggle.com/competitions/commonlit-evaluate-student-summaries/overview

1. Introduction

Overview:

Using machine learning to evaluate the quality of summary written by students has more advantages than traditional scoring by faculty members. Firstly, machine learning algorithms are able to evaluate students' summaries in an objective manner, avoiding the bias that comes with subjective grading. The same algorithm can be applied to the work of multiple students, ensuring consistency in assessment. Additionally, ability to automate the evaluation of large sums of summaries, which is more efficient than manual evaluation. This is useful for evaluation tasks in large-scale courses or research projects. Thirdly, Machine learning models can assist teachers to more quickly identify students who perform better or poorer among a large number of students, so as to provide targeted assistance or challenges.

However, assessing the quality of student-written summaries is a complex task. Converting student-written summaries into feature representations that machine learning models can process is a critical step. How to effectively extract useful features from text and represent them into numerical vectors required by machine learning algorithms is a challenging problem. And the difference between the evaluation effect of the model and the real result is also an important consideration. Moreover, students' writing styles and expressions may be diverse, which makes it difficult to measure the quality of abstracts with a single standard. Some abstracts may be normative but not creative, while others may be creative but loosely structured, adding to the complexity of the assessment.

The CommonLit Kaggle competition aims to explore and develop efficient machine learning processes that automatically assess summaries written by students in grades 3-12.

Dataset introduction:

In this experiment, we use two csv files as our dataset: prompts_train.csv and summary_train.csv

- **prompts_train.csv:**

prompt_id: unique identifier of each piece of data in the prompts_train.csv file
prompt_question: requirements for students to write a summary
prompt_title：title of article
prompt_text：content of article

- **summary_train.csv:**

student_id: unique identification of each piece of data in the summary_train.csv file
prompt_id：The id of the article corresponding to the summary written by the student
text：The content of the summary written by the students
content：the grade of the summary content
wording：the grade of the summary words and rhetorical devices

- **prompts_test.csv:** format of prompts file
- **summary_test.csv:** format of summary file
- **Sample_submission.csv:** format of the submission

2. Methods

2.1 methods applied:

In order to train better models and predict better target values, we use a variety of fitting methods and make comparisons. After a lot of trial and comparison, we found that the linear fitting method in this experiment is poor, even worse than the natural guess. So in the end, most of the algorithms we choose belong to nonlinear fitting. The following 6 methods are the more effective methods that we have selected.

Kernel Ridge Regression:

Kernel Ridge Regression (KRR) is a machine learning algorithm used for regression analysis. It is an extension of linear ridge regression, suitable for non-linear problems. Kernel Ridge Regression combines the advantages of Ridge Regression and Kernel Methods. It can handle non-linear problems and limit the complexity of the model and avoid overfitting.
The prediction function of Kernel Ridge Regression is as follows:

$$f(x) = \Sigma\alpha_i \mathrm{K}(x, x_i)$$

where x is the new input variable, x_i is the training sample, K is the kernel function, and α_i is the weight of each training sample. The weights α are calculated by the following formula:

$$\alpha = (\mathrm{K} + \lambda I)^{-1} y$$

where K is an n×n matrix, with elements $\mathrm{K}_{ij} = \mathrm{K}(x_i, x_j)$, x_i and x_j are training samples, ***y*** are the target values of the training samples, ***λ*** is the regularization parameter (ridge parameter), ***I*** is the identity matrix, and ***α*** is an n-dimensional vector, with each element corresponding to the weight of a training sample.

hyper-parameters:

kernel function: The kernel function is used to map the original characteristics to the high dimensional eigenspace, thereby realizing the nonlinear regression. In this experiment, we use linear function.

lambda: lambda is a regularization parameter in the return of the ridge, which is used to control the complexity of the model and prevent the coincidence.

K-Nearest Neighbor Regression:

K-Nearest Neighbor Regression (KNN Regression) is an instance-based learning method used for regression problems. Its basic idea is that for a new input instance, it finds the ***k*** closest instances in the training set, and then predicts the target variable of the new instance based on the average or weighted average of the target variables of these ***k*** instances. The K-Nearest Neighbors algorithm (KNN) does not have a fixed mathematical formula because it mainly relies on instance comparison and voting decisions, rather than learning a set of specific parameters or function mappings.

hyper-parameters:

n_neighbors: It represents how many samples of the nearest neighbors are taken into account when making a prediction.

weights: KNN algorithm can assign different weights to the nearest neighbor samples.

Support Vector Regression:

Support Vector Regression (SVR) is the application of Support Vector Machine (SVM) in regression problems. SVM is a classification model. Its basic model is defined as a linear classifier with the largest interval in the feature space. Its learning strategy is to maximize the interval, which can finally be transformed into the solution of a convex quadratic programming problem.

SVR's basic idea is to find a function that maximizes the number of samples whose predicted values differ from their actual values by less than a certain threshold, while also minimizing the complexity of the function. If the difference between the predicted value and the actual value exceeds this threshold, it is necessary to penalize such "out-of-range" cases. The main objective of SVR can be represented by the following optimization problem:

$$minimize: \frac{1}{2}\ ||\omega||^2 + C \sum \xi_i$$

subject to:

$$y_i - \omega^T x_i - b \le \varepsilon + \xi_i, \text{ for } i = 1, \dots, n$$
$$\omega^T x_i + b - y_i \le \varepsilon + \xi_i, \text{ for } i = 1, \dots, n$$
$$\xi_i \ge 0, \text{ for } i = 1, \dots, n$$

where w and b are parameters to be learned, C is a hyper-parameter for controlling the penalty of the error, ε is the tolerance error threshold, and ξ_i is the slack variable used to handle cases where the difference between the predicted value and the actual value exceeds ε.

An important feature of SVR is that it can use the kernel function to map data to high-dimensional space to solve problems that are not linearly separable. In this way, SVR can learn complex non-linear relationships.

hyper-parameters:

C: The regularization parameter C is used to control the complexity of the model. A larger value of C means that the tolerance for error decreases, and the model will better fit the training data.

epsilon: epsilon is a tolerance parameter in support vector regression that controls the allowed range of prediction errors in the loss function.

loss: loss function.

max_iter: maximum number of iterations

XGBoost Regression:

XGBoost Regression is an application of Extreme Gradient Boosting algorithm to regression problems. XGBoost is an optimized Gradient Boosting Machine method, which is an integrated learning method based on Decision Trees. The main benefits of XGBoost are its efficient execution performance and model accuracy.

The core of XGBoost algorithm is to solve the following optimization problems:

$$min \sum l(y_i, \bar{y_i}) + \sum \Omega(f_i)$$

The first $\Sigma\, l(y_i, \bar{y_i})$ is a loss function that measures the difference between the predicted value $\bar{y}_i$ and the actual value y_i. The loss function() depends on the problem. For regression, we can choose the mean square error loss function $l(y_i, \bar{y}_i) = (y_i - \bar{y}_i)^2$.

The second term, $\Sigma\,\Omega(f_i)$, is a regularization term used to prevent overfitting, f_i represents the prediction function of the i th basis learner. In XGBoost, this regularization term consists of two parts: the sum of squares of the number of leaf nodes in the tree and the weight of the leaf nodes, i.e.

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda||\omega||^2$$

where T is the number of leaf nodes in the tree, w is the weight of the leaf nodes, γ and λ are hyper-parameters that control the complexity of the model.

hyper-parameters:

n_estimators: the number of decision trees to be built by this model

max_depth: the maximum depth of each decision tree

eta: learning rate

sub_sample: proportion of samples per tree that the model trains

colsample_bytree: proportion of features per tree that the model trains

gamma: the threshold for loss reduction in tree splitting

Random Forest:

Random Forest is an ensemble method for decision trees that improves accuracy and stability by combining multiple trees. It introduces randomness through data sampling (bootstrap sampling) and feature selection, which increases model diversity, reduces overfitting, and improves generalization. Each tree uses a random subset of features for splitting, reducing feature correlation.

Compared to decision trees, Random Forest addresses issues like overfitting, instability, and high variance. It reduces the impact of noisy data on predictions and stabilizes results by integrating multiple trees. Hyperparameters like max_depth, min_samples_split, min_samples_leaf, and max_features from decision trees are inherited in Random Forest. Limiting the tree depth with max_depth prevents overfitting, and the values can be tuned via methods like cross-validation.

hyper-parameter:

min_samples_split: The minimum number of samples required for internal node subdivision. Increasing this value can prevent the decision tree from performing too many divisions on smaller subsets, which helps reduce overfitting.

n_estimators: the number of decision trees to be included in the Random Forest ensemble.

min_samples_leaf: The minimum number of samples required for leaf nodes. Increasing this value can make leaf nodes contain more samples, which helps reduce overfitting.

max_features: The number of features considered when finding the best split. Decreasing this value can reduce the randomness of feature selection and help reduce the variance of the model.

Neural Networks:

Neural Networks are computational models inspired by biological neural networks that are used to solve various machine learning and artificial intelligence tasks. It consists of multiple neurons and is arranged in a hierarchical structure, forming a network that propagates forward. The main goal of a neural network is to learn a feature representation of the input data and use the learned representation to perform prediction, classification, regression, or other tasks.

Neuron:

Neurons are the basic building blocks of neural networks. It receives input from the neurons in the previous layer and sums the input weighted according to the connections with weights. The output is then generated by a nonlinear transformation through the activation function. The number of neurons used in this experiment is 70.

Layer:

A neural network consists of multiple layers. The input layer receives the raw data as feature input, the output layer produces the final prediction result, and the hidden layer in the middle is used to learn the feature representation. This experiment uses 1 input layer, 1 hidden layer and 1 output layer.

hyper-parameter:

Activation Function: The activation function is a very important component in neural networks. It defines the way neurons output, the process of converting input signals into output signals. Its main role is to introduce nonlinearity, which is crucial for the learning ability of neural networks.

Loss Function: The loss function is the function used to measure the difference between the output of the neural network and the actual target. When training a neural network, the goal of the network is to minimize the value of the loss function by adjusting the network parameters.

Optimizer: The optimizer is an important component in the neural network training process, which is used to automatically tune the parameters of the neural network to minimize the loss function.

batch_size: The size of a batch of samples used for gradient descent during training.

epochs: The entire training data set will be passed through the neural network in its entirety.

2.2 pre-processing and feature extraction:

- Natural language processing is more difficult and requires more steps than direct data processing. It is necessary to clean the text, remove special characters, punctuation marks, HTML tags, etc. in the text, and only keep meaningful text information. Uniformly converts text to uppercase or lowercase to eliminate ambiguity caused by case.
- Delete the emoticons in the text and replace them with the corresponding language vocabulary, for example, replace ":3" with "happy face".
- Also, expand the abbreviations. This may eliminate part of the ambiguity, and after the abbreviation is expanded, the originally abbreviated words can be accurately represented, which helps to improve the expressive ability of features.

- Stemming and lemmatization restore words to their base forms, such as "running" to "run", which can reduce the size of the vocabulary and remove the noise caused by different forms of word forms.
- Perform a second text cleaning, after converting all required information into letters, after removing all non-alphabetic characters.
- Merge the title and content of the dataset for better vectorization later. Because we think that summary whether has words in titles is likely to be a factor that affects the score. So we merge title and content to calculate the words TF-IDF value to emphasize the theme words in summary.
- Calculate the length of each summary and save them. Because we believe that summary length is also a factor affecting the score.
- The number of stop words used in a summary will also affect the possibility of scoring, so we counted the number of stop words and recorded them. After that, we removed the stop words in each summary.
- We think the TD-IDF value is a very good way to deal with natural language. We counted the words in all datasets and then calculated and vectorized the TF-IDF values of the words in each summary.
- The dataset is divided into training and test sets for model training, tuning, and evaluation. Extract the data of different articles(prompt_id) and evenly divide the training set and test set to ensure that each article appears in the same proportion in the data set and test set.
- The final extracted features: words TF-IDF value, the length of summary, the number of stop words, the number of emoji and the number of abbreviation.

2.3 Evaluation metrics:

In this experiment, we only need to predict two results: the content score of summary and the wording score of summary. We believe that the good or bad of a model training depends on the gap between the models scoring and the manual scoring(target value), so we think it is appropriate to use RMSE(Root Mean Squared Error) to judge the good or bad of a models.

Each time, we need to train two models using the same method, and score the content of summary and the wording of summary respectively. Moreover, their importance is the same, so we take the average RMSE of the results predicted by these two models as the final evaluation standard. The smaller the loss, the better the performance of the model.

The content test data set is **X**, the wording test data set is **Y**, and the test set size is **n**.

$$RMSE_{content} = \sqrt{\frac{\sum (x_i - \overline{x_i})^2}{n}}$$

$$RMSE_{wording} = \sqrt{\frac{\sum (y_i - \overline{y_i})^2}{n}}$$

$$\text{loss} = \frac{RMSE_{content} + RMSE_{wording}}{2}$$

3. Result

We rigorously explore six algorithms, optimize their hyper-parameters to obtain the most promising models for each. Subsequently, we compare the prediction loss (the mean of RMSE) generated by these models.

At the same time, we also pay attention to the importance of efficiency. Therefore, after obtaining six ideal models, we recorded the training and computation time of each model while running them.

Kernel Ridge Regression:

When optimizing Kernel Ridge Regression, we focused on choosing different type of kernel functions. As for the type of kernel, we select linear, poly (Polynomial) and sigmoid.

The obtained results are shown in following table:

Table 1. The result of Kernel Ridge Regression

Type of kernel function	Loss (the mean of RMSE)	Time used (train+predict)
Linear	0.653	5.916s
Polynomial	0.678	7.420s
Sigmoid	0.702	94.234s

From the *Table 1*, we could get that linear perform best. Therefore, we select kernel='linear' as the result of Kernel Ridge Regression.

K-Nearest Neighbor Regression:

When optimizing K-Nearest Neighbor Regression, we focused on choosing the number of neighbors and the weights of each neighbor.

As for number of neighbors, we try from 1 to 150. As for weight method, we select uniform (different neighbors have same weight) and distance (closer neighbor has higher weight).

The obtained results are shown in following table:

Table 2. The result of K-Nearest Neighbor Regression

number of neighbors	Loss (the mean of RMSE)		Time used (train+predict)
	weights= 'uniform'	weights='distance'	
1	0.901	0.901	2.650s
2	0.788	0.783	2.591s
3	0.756	0.751	2.586s
⋮	⋮	⋮	⋮
29	0.691	0.681	2.885s
30	0.691	0.681	2.819s
⋮	⋮	⋮	⋮
86	0.689	0.679	2.537s
87	0.687	0.678	2.631s
⋮	⋮	⋮	⋮
148	0.687	0.678	2.608s

149	0.687	0.678	2.671s
150	0.687	0.678	2.542s

From the *Table 2,* we could get that the loss is lower when we use 'distance' (It means that the closer neighbors will have a greater influence) as the weighting scheme for the neighbors. We also can see when number of neighbors increase, the rate of loss change is slowing down. Therefore, we select n_neighbors=87 and weights='distance' as the result of K-Nearest Neighbor Regression.

Support Vector Regression:

When optimizing Support Vector Regression, we focused on choosing the maximum number of iterations, the different loss function and different regularization parameter (C).

As for the maximum number of iterations, we try from 1000 to 50000 (Increase by 1000 each time, because the interval is too small, the difference is not significant). As for loss function, we select epsilon_insensitive and squared_epsilon_insensitive. As for regularization parameter, we select 0.01, 1.0 and 100.

The obtained results are shown in following table:

Table 3. The result of Support Vector Regression

maximum number of iterations	**Loss function**	**Loss (the mean of RMSE)**			**Time used (train+predict)**
		C=0.01	C=1.0	C=100	
1000	epsilon_insensitive	0.767	0.908	0.908	2.203s
	squared_epsilon_insensitive	0.734	0.884	0.907	2.341s
2000	epsilon_insensitive	0.753	0.919	1.463	3.091s
	squared_epsilon_insensitive	0.722	0.830	1.452	3.754s
⋮	⋮	⋮	⋮	⋮	⋮
24000	epsilon_insensitive	0.749	0.672	0.787	27.068s
	squared_epsilon_insensitive	0.718	0.645	0.778	25.375s
25000	epsilon_insensitive	0.749	0.699	0.834	30.776s
	squared_epsilon_insensitive	0.718	0.641	0.823	26.337s
⋮	⋮	⋮	⋮	⋮	⋮
49000	epsilon_insensitive	0.749	0.646	0.725	64.071s
	squared_epsilon_insensitive	0.718	0.637	0.717	50.131s
50000	epsilon_insensitive	0.749	0.644	0.753	63.842s
	squared_epsilon_insensitive	0.718	0.638	0.745	49.389s

From *Table 3*, we could get that higher regularization parameter will lead to significant fluctuations in the loss and lower regularization parameter will make the loss quickly converge to a fixed value. We also find that higher maximum number of iterations will have lower loss, but too heigh may lead to overfit. As for loss function, squared_epsilon_insensitive performs better, because it is more sensitive to larger errors compared to epsilon_insensitive. Therefore, we select max_iter=49000, C=1.0, loss='squared_epsilon_insensitive' as the result of Support Vector Regression.

XGBoost Regression:

When optimizing XGBoost Regression, we focused on the impact of the maximum depth of each decision tree and the number of iterations. We tried to optimize each hyper-parameter, and selected a group of appropriate hyper-parameters for comparison experiments(eta: 0.1 subsample: 0.7 colsample_bytree: 0.8 gamma: 1).

As for number of iterations, we select from 1 to 100. As for the maximum depth of each tree, we select 5, 10 and 20.

The obtained results are shown in following table:

Table 4. The result of XGBoost Regression

number of iterations	**Loss (the mean of RMSE)**			**Time used (train+predict)**
	Max depth = 5	Max depth = 10	Max depth = 20	
1	1.103	1.099	1.098	6.791s
2	1.033	1.026	1.024	7.598s
3	0.973	0.964	0.961	8.275s
⋮	⋮	⋮	⋮	⋮
33	0.598	0.581	0.575	37.620s
34	0.596	0.580	0.575	37.570s
⋮	⋮	⋮	⋮	⋮
60	0.572	0.565	0.565	53.167s
61	0.572	0.565	0.565	52.480s
⋮	⋮	⋮	⋮	⋮
98	0.565	0.565	0.565	77.740s
99	0.565	0.565	0.565	81.691s
100	0.565	0.565	0.565	78.582s

From the *table 4*, we can see that after increasing the number of iterations to 50, the decreasing rate in loss is very low. When the number of iterations reaches 100, the loss value becomes stable. Looking at the results produced by three different sets of models, the larger max_depth is, the faster the loss decreases, but they all stabilizes at 0.565. Therefore, we select max_depth = 10 and iterations = 60 as the result of XGBoost Regression.

Neural Networks:

When designing our neural network, we decided to use mean_square_error as our loss function depending on the dataset type. For optimizer, we use Adam, because it can automatically adjust the learning rate at any time. The number of iterations we set is large enough to ensure that the best results are found. We focused on the impact of the number of neurons,so we choose 20, 60, 100 and compare them.

Since neural networks could suffer from overfitting, we will find the lowest loss for each set of data as our best result.

The obtained results are shown in following table:

Table 5. The result of Neural Network

number of iterations	**Loss (the mean of RMSE)**			**Time used (train+predict)**
	neurons = 20	neurons = 60	neurons = 100	
1	0.8	1.106	1.308	3.791s

2	0.705	0.956	1.06	3.598s
3	0.628	0.867	0.964	4.275s
4	0.602	0.817	0.914	4.873s
5	0.668	0.763	0.829	4.620s
6	0.593	0.717	0.747	5.405s
7	0.67	0.637	0.678	5.637s
8	0.611	0.679	0.643	5.626s
9	0.602	0.633	0.629	4.987s
10	0.665	0.784	0.621	5.345s
11	0.632	0.632	0.617	5.781s
12	0.634	0.624	0.623	5.636s
⋮	⋮	⋮	⋮	⋮

Looking at the table, the best results were obtained with 20 neurons and 6 iterations. Therefore, we select number of neurons = 20 and iterations = 6 as the result of Neural Network.

Random Forest:

When optimizing Random Forest, we focused on choosing the maximum number of iterations, the different loss function and different regularization parameter (C).

As for the maximum number of iterations, we try from 1000 to 50000 (Increase by 1000 each time, because the interval is too small, the difference is not significant). As for loss function, we select epsilon_insensitive and squared_epsilon_insensitive. As for regularization parameter, we select 0.01, 1.0 and 100.

The obtained results are shown in following table:

Table 6. The result of Random Forest

Number of decision trees	Loss (the mean of RMSE)	Time used (train+predict)
1	0.787	3.629s
2	0.699	6.931s
3	0.662	10.092s
⋮	⋮	⋮
28	0.584	89.405s
29	0.583	92.589s
30	0.583	95.649s
⋮	⋮	⋮
48	0.579	153.536s
49	0.579	156.687s
50	0.579	160.228s

From *Table 6*, we could get that more decision trees will lead to lower loss but spend more time. The rate of loss change is slowing down. Therefore, considering both the loss and time cost, we select n_estimators=29 as the result of Support Vector Regression.

Comparison:

Figure 1. The comparison of Loss (the mean of RMSE) between different models

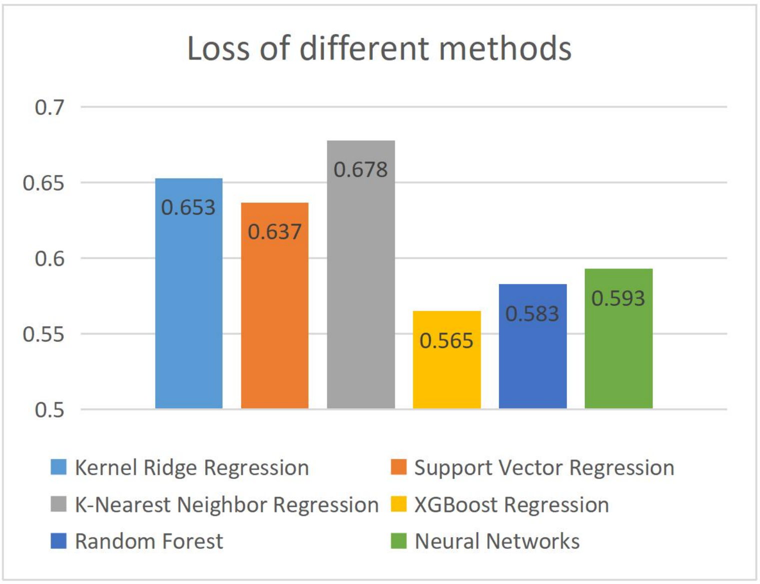
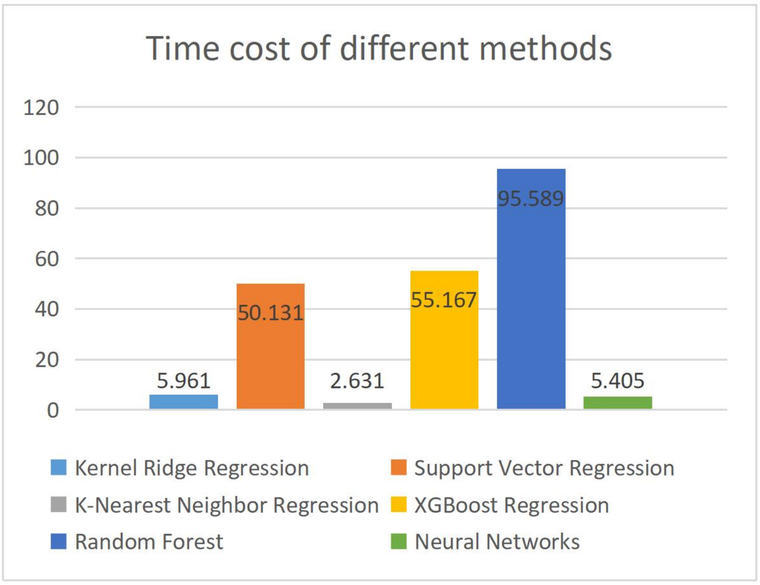


Figure 2. The comparison of time cost between different models



The above results are the best from each model. As we can see from the comparison, XGBoost Regression performs the best in the similarity between the model prediction results and the target. However, it does not perform well in efficiency. Neural networks perform well in terms of both accuracy and speed.

4. Discussion

4.1 Comparison of difference methods:

In this experiment, we used a total of six methods for model training, all of which have their own advantages and disadvantages:

XGBoost Regression:

The advantage of XGBoost Regression is that it has a lot of learning ability, and it has good performance and efficiency when working with large data sets. However, there are many hyper-parameters need to adjust, so it takes more time to adjust if you want to get a better result.

Support Vector Regression:

The advantage of Support Vector regression is that it can handle nonlinear data relationships. Since it only supports using vector to predict, the memory consumption is relatively small. However, when dealing with large-scale samples, the computational complexity is relatively high, which affects the performance of the model.

Kernel Ridge Regression:

The advantage of Kernel Ridge regression is that it can handle nonlinear data relations, and regularization terms can effectively control model complexity and prevent overfitting problems. However, a higher degree of computational complexity occurs when dealing with large-scale samples, which is similar to Support Vector regression.

Neural Networks:

The advantage of Neural Networks is that they have strong learning ability, can handle large-scale high-dimensional data sets, and can cope with various situations by modifying the number of neurons and layers. However, in order to prevent overfitting, the quantity requirement of the trained data set is large.

Random Forest :

Random Forest has the advantage of strong learning ability and good adaptability to high-dimensional data and large-scale data sets. However, there are many hyper-parameters that need to be adjusted and the performance of linear models in dealing with some linear problems is worse than linear models.

K-Nearest Neighbor Regression

The advantage of K-Nearest Neighbor Regression is that it is simple and easy to understand and has fast processing efficiency. K-nearest neighbor regression performs well in processing small samples. However, the whole algorithm is very dependent on the K value, so we needs to find a appropriate K value for getting an ideal result.

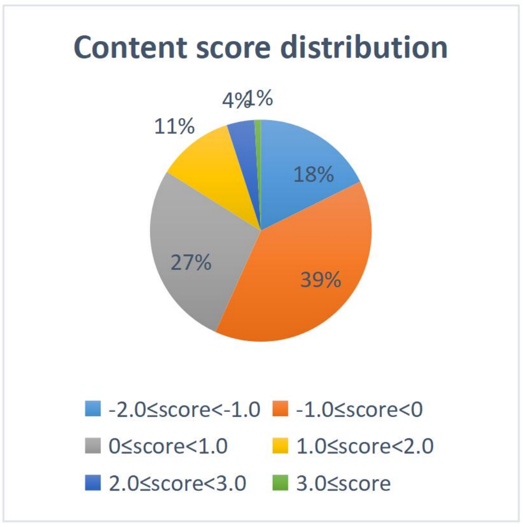
4.2 Which algorithm are more appropriate and why:

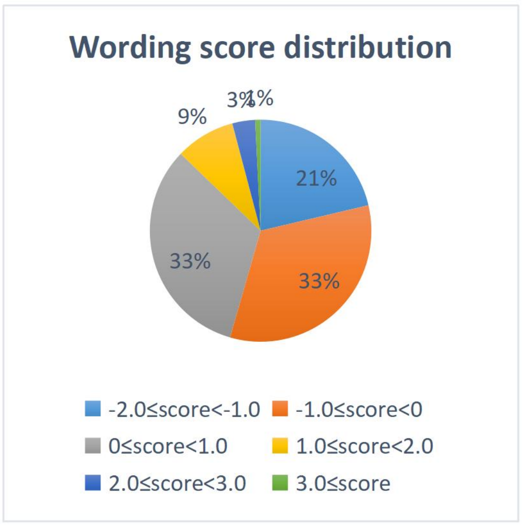
According to the results of this experiment, XGBoost Regression has the best final score. Because the score (RMSE) obtained by XGBoost Regression in the test training is the lowest among the six models. In other words, the combined performance of XGBoost in predicting the content grade and wording grade is the closest to the target value. Not only that, only part of its hyper-parameters are optimized and adjusted in this experiment, and the effect will be better if the optimization is continued.

4.3 Future improvement:

For extracting features from natural language, although we have tried our best to extract the most appropriate and comprehensive features we think, such as the length of summary and the TF-IDF value of each word, we believe that there are still better features that can be extracted.

Figure 3. Distribution of content and wording score in dataset





In addition, by looking at the *Figure 3*, we can know that the distribution of content and wording scores in dataset. There are some extreme data, such as the score higher than 2.0 (5%) and higher than 3.0(less than 1%), which maybe the noise and affect the training of the model. In the future, we could make more attempts in data cleaning in order to reduce the influence of extreme data.

Random Forest is a decision tree based Bagging algorithm. According to the performance of Random Forest, ensemble learning methods such as Bagging can greatly improve the accuracy of data results of machine learning algorithms. Therefore, we believe that other machine learning algorithms can also be combined with ensemble learning methods to further optimize the model training. For example: Bagging + SVM, Bagging + KNN.

Although the six algorithms used in this experiment have obtained relatively best results through our continuous optimization. If we try to optimize more hyper-parameters to make algorithms more suitable, we may get better model.

5. Conclusion

To judge whether an algorithm is good, we should not only look at how well it predicts, but also consider the complexity of the algorithm, its efficiency, and how it performs on larger datasets. As mentioned in the discussion section, the results of XGBoost Regression are the best in this experiment. Not only that, but it is also efficient in processing the dataset. Based on its properties, it will perform well even when dealing with larger datasets. Therefore, we believe that XGBoost Regression is the most appropriate to deal with natural language under the premise of using TF-IDF, article length and other features.

Random Forest Regression provide excellent fitting results, but they are inefficient to train and fit, and take a long time to run on larger datasets, so we do not consider them a suitable method.

It is worth noting that the Neural Network performs well both in terms of fitting results and running efficiency. We believe that the fitting effect will be better if more features and training set samples are used to train the Neural Network model, so Neural Network may be another suitable method.

In the end, we conclude that the two algorithms, XGBoost Regression and neural networks, are more suitable for the situation of this competition.

Reference:

[1] Aoyagi, M and Watanabe, S, 2022, 'Resolution of singularities and the generalization error with Bayesian estimation for layered neural network'
Available at: <https://www.researchgate.net/publication/228878350>
[Accessed 6 July 2023]

[2] Dmitriy Gakh, 2023. *CommonLit (Beginner, Polyfit, Text Size).* [Online]

Available at: <https://www.kaggle.com/code/dmitriygakh/commonlit-beginner-polyfit-text-size>

[Accessed 19 July 2023]

[3] Harsh Priye, 2023. *Summary Evaluation.* [Online]

Available at: <https://www.kaggle.com/code/harshpriye/summary-evaluation>

[Accessed 25 July 2023]

[4] Shah, V and Mehta, M, 2022, 'Emotional state recognition from text data using machine learning and deep learning algorithm'

Avaliable at: <https://www.nstl.gov.cn/paper_detail.html?id=cfd5e98ae6534e9ce1f3979dbf302d7b>

[Accessed 8 July 2023]

[5] Yang Liu and Meng Zhang, 2017, 'Neural Network Methods for Natural Language Processing'

Avaliable at:
<https://direct.mit.edu/coli/article/44/1/193/1587/Neural-Network-Methods-for-Natural-Language>

[Accessed 9 July 2023]

[6] yunsuxiaozi, 2023. *LinearRegression(evaluate-student-summaries).* [Online]

Available at:
<https://www.kaggle.com/code/yunsuxiaozi/linearregression-evaluate-student-summaries>

[Accessed 19 July 2023]

Figures

Comparison of the epochs-loss figure between different number of neurons in the Neural Network:

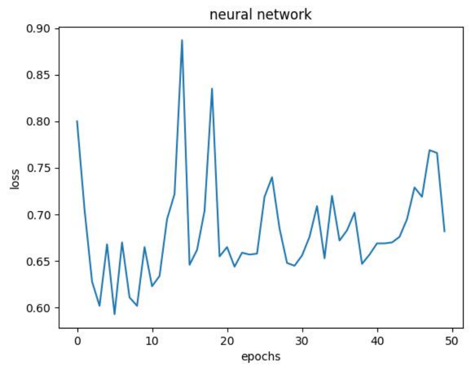


figure 4: 20 neurons

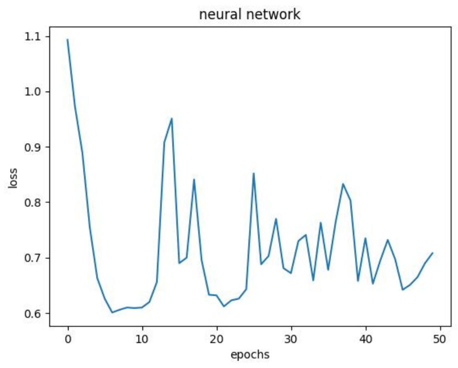


figure 5: 60 neurons

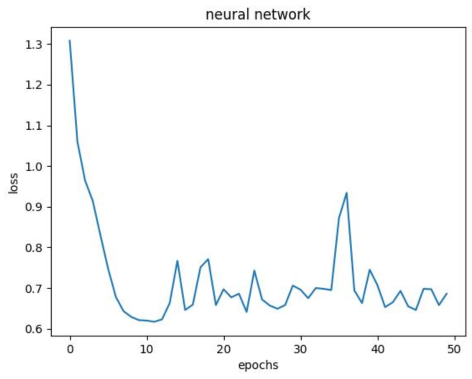


figure 6: 100 neurons

Comparison of the iterations-loss figure between different maximum depth of each tree in the XGBoost Regression:

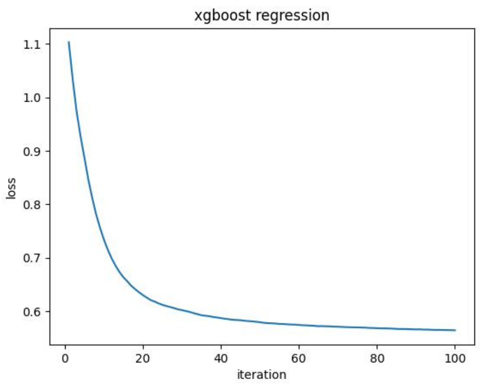


figure 7: maximum depth: 5

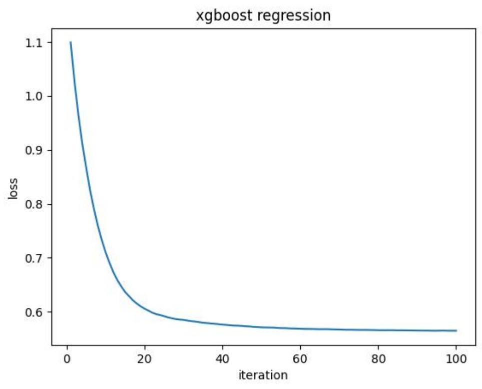


figure 8: maximum depth: 10

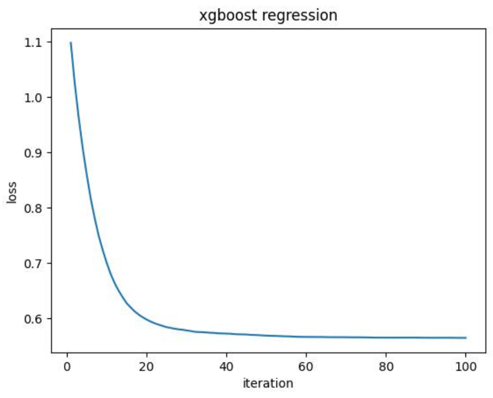


figure 9: maximum depth: 20

Comparison of the number_of_neighbors-loss figure between different weight method in the KNN Regression:

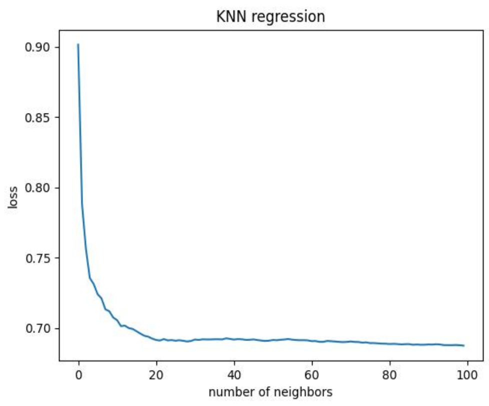


figure 10: distance

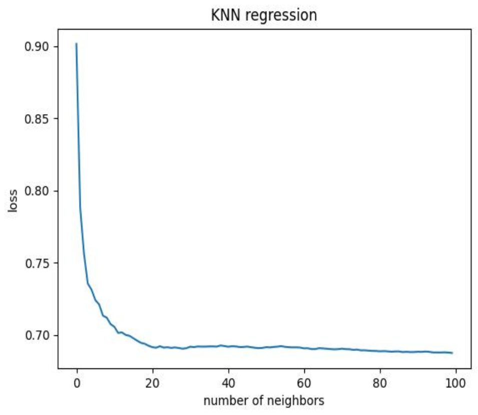


figure 11: uniform

Comparison of the iter-loss figure between different loss function in the Support Vector Regression:

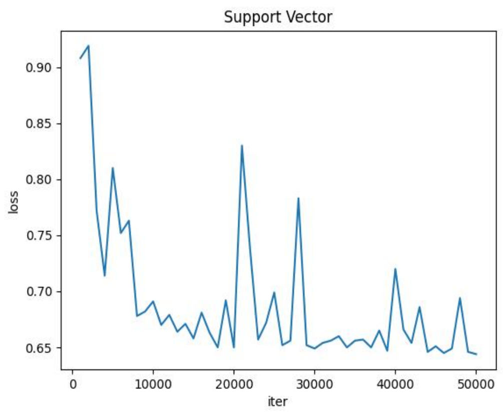


figure 12: c=1.0, epsilon_insensitive

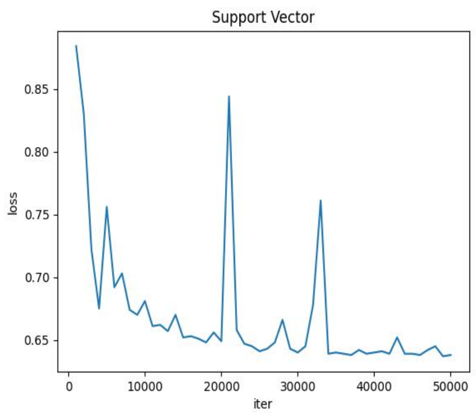


figure 13: sv c=1.0, squared_epsilon_insensitive

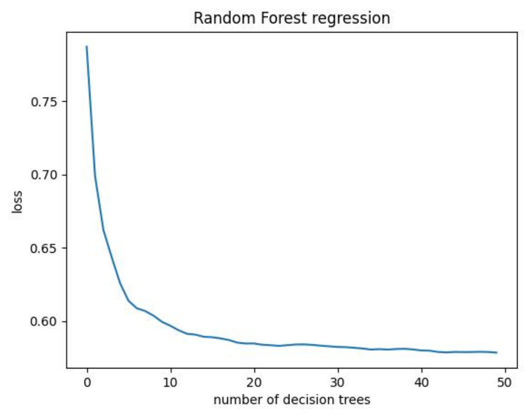


figure 13: number_of_decision_trees-loss figure in Random Forest Regression