Problem Chosen	2023 MCM/ICM Summary Sheet	Team Control Number
C		2306943

Wordle Forecast Based on ETS and Machine Learning Model

Summary

Since The New York Times need an analysis of the reported results to optimize the wordle game and attract more people. We are asked to provide prediction models and give an accurate prediction.

We get an interval prediction for the number of reported results from January 1 to March 1, 2023. The final prediction is a 95% confidence interval with a **lower bound of 5747.029**, a **higher bound of 28448.5**, and the **forecast mean is 17097.8.**

To predict a exact word on a future date, we decide to use **BP neural network** to achieve this goal. We use the whole data set to train the model. And its loss declines and converges. Our prediction of "EERIE" of each tries is **[0.0570, 0.0926, 0.2037, 0.2712, 0.1495, 0.1396, 0.0864]**.

For classifying words by difficulty, We apply EWM to compute weight of each kind of tries. And we classify the words into 3 difficulty levels (easy, normal, difficult) and the word ERRIR is classified as a difficult one. Further,we try to use a model mapping from word to difficulty. We finally achieve **71.2 %** prediction accuracy under **70% training set**.

There are interesting **features** we found in the data set. 1. The curve of numbers of reported results increases dramatically firstly, and declines sharply, Then gradually remains at a relatively stable level. 2. the Proportion curve of the number of people who choose the hard mode fits well to a kind of formula:$y = ae^{bx} + ce^{dx}$, and we calculate the parameter in the formula. 3. words including the letter 'j' or 'z' often have a high proportion of the number of people who choose the hard mode.

Keywords: BP Neural Network;Exponential Smoothing (ETS); Entropy Weight Method(EWM)

Contents

1 Introduction

Wordle is a popular crossword puzzle, founded by Josh Warle. It basic gameplay is that given different reactions to the letters that the player guesses, to help the player speculate the final answer. Since its launch in October 2021, data shows it has had millions of daily active users by January 2022.

According to the statistical display, the total number of players is declining. In order to help its manufacturers arrange contents of the game more reasonably, we have great need to build a model to analyze relevant data and predict the results. Then, we will have a more suitable way to make it better. Our model needs to fulfill the following tasks:

- Develop a model to explain this variation and use your model to create a prediction on a future date
- Figure out the attributes of the word affect the percentage of scores reported that were played in Hard Mode
- develop a model to predict the associated percentages of (1, 2, 3, 4, 5, 6, X) for a given future solution word on a future date
- Develop and summarize a model to classify solution words by difficulty

For these requirements, common ARIMA model calculate autocorrelation and partial autocorrelation warrant that it could not figure it out. As to the ways of difficulty classification, usual approach is not objective. In other words, special and used models are in need.

In this paper, we build a rather helpful model which could provide particular solutions to the related issues. It could give trustworthy the number of people in the future. And evaluate the difficulty and result of a given word. In addition, it shows the effect of word attributes on the results.

1.1 Other Assumptions

- **Time series can be decomposed into addictive components or multiplicative components.** An additive series can be depicted by:$y_t = Trend_t + Seasonal_t + Residual_t$ And a multiplicative can be depicted by:$y_t = Trend_t * Seasonal_t * Residual_t$. Here y is the data, S means seasonal component, T means trend-cycle component, and R means residual component at time t. In fact, we can transform the multiplicative series into the addictive one through logarithmic variation.

- **There are several sources of uncertainty when forecasting a future value of a time series** . The first uncertainty in model choice – maybe another model is correct, or maybe none of the candidate models is correct, and the second is the uncertainty in the future innovations. And the last is iii.the uncertainty in the estimates of the parameters of the equations.

- **The word given on each day is totally random**. In order to focus on the effection (on reported results) of word and date themselves. we assume that the relationship between word to word is extremely low, in a word, it was given randomly.

- **Rate of tries can be divided into two parts, one increases the difficulty level, the other does the opposite.** We define a difficulty level.guessing the right answer in less tries will reduce the difficulty of the word, which means the bigger rate of percentage less tries, the lower the difficulty level of that word. So we assume that tries 1-3 decrease the difficulty level, percentage of tries above 3 augments it.

2 Abbreviations and Symbols

Before starting, we list the symbols we used in our paper, and make a description of each symbol. Wish it can help understand our paper better.

Table 1: Abbreviations and Symbols

Symbol/Abbreviation	Description
P	the output of our neural network model,which is also a matrix
T_x	a target matrix consist of each rate of each tries
$dict$	the map for finding index of each letter
$[a, b]$	the shape of a matrix which rows = a, cols = b
h_i	a hidden state in BP neural network
α	Smoothing parameter for the level of the series
β	discount parameter ,with a value between 0 and 1
ϕ	damping parameter
b_t	slope at time t
ε_t	white noise with a Gaussian distribution
x_n	the state vector at the last time of observation
y_t	the data at time t
h	the number of steps predicted
$y_{n+h\|n}$	h-steps-ahead predicted data
σ^2	variance
μ_{n+h}	h-steps-ahead forecast mean
v_{n+h}	h-steps-ahead forecast variance
θ_h	intermediate variable in formula (8) and (9)
c_j	intermediate variable in formula (8) and (9)
z_q	denotes the q^th quantile of a standard Gaussian distribution
$\mathbf{w}$	weight vector that contains EWM weight of each tries
$\mathbf{W}_i$	the weight matrix of a linear layer
$\mathbf{X}_i$	the input matrix including a batch of input data
F(X)	the ReLU function, input **X** is a matrix
$F_{loss}(\mathbf{X},\mathbf{Y})$	the loss function, inputs **X,Y** are matrices
$F_{ids}(\mathbf{X})$	the function that can convert the letter into its index
$ets()$	the function of Exponential smoothing state space model

3 Predicting the Number of Reported Results

In this section, we employ time series forecasting method to achieve a prediction interval for the number of reported results on March 1, 2023.

3.1 Selecting and Implementing Model

Through the observation of this time series, we found that after 100 days, the recorded numbers decreased rapidly and gradually remained at a relatively stable level. Therefore, we consider using the exponential smoothing model(ETS) to fit the data.

To identify the best exponential smoothing model suitable for the given time series. We implement a common-sense strategy expanded by Chatfield[1]. Here we give a nutshell of the strategy: Here we give a nutshell of the strategy:

- Plot the time series , observe the trend, seasonal variation and other data characteristics. Outliers, and quick changes in structure are important characteristics that indicating ETS is not appropriate.
- Examine the outliers , considering making some adjustment to modify the wrong data.
- Decide on the form of trend and seasonal variation.(We consider make transformation to stationary the time series or minimize the variance through difference .
- Fit an appropriate method. Akaike's Information Criterion (AIC) along with AIC corrected for small sample bias (AICc) and the Bayesian Information Criterion (BIC) are applied to choose an appropriate ETS method.

We employ the ETS statistical framework in the forecast package of R to forecast the interval for the number of reported results on March 1, 2023.First,We implement $ets()$ function to select the method by minimizing AIC_c.

The model selected is ETS(M,Ad,N) , which means nonseasonal damped addictive trend method with multiplicative errors. In this paper, we call this model the DA-N(damped addictive) model. It can be depicted by the following 3 equations[2]:

$$y_t = (l_{t-1} + \phi)(1 + \varepsilon_t) \tag{1}$$

$$l_t = (1 + \alpha\varepsilon_t)(l_{t-1} + \phi_{t-1}b_{y-1}) \tag{2}$$

$$b_t = \phi b_{t-1} + (l_{t-1} + \phi bt - 1)\varepsilon_t \tag{3}$$

The equations are forecast equation, level equation and slope equation. In these equations,ε_t is a white noise process with a variance σ^2. For the parameter estimates above, here are the explanation: the ETS(M,Ad,N) model has the reduced rate ϕb_{t-1} at time t.The output also returns the estimates of initial state: $l_0 = 65494.507, b_0 = 15247.8629$. The parameter estimates are:

Table 2: parameters' value

parameter	α	ϕ	AIC	AIC_c	BIC
value	0.1573	0.9688	8319.0	8319.239	8342.3

The damping exponent ϕ as a value of 0.9688(slightly smaller than 1). In practice, ϕ has the effect of slowing the trend, which means the time series should approach to a constant over time.

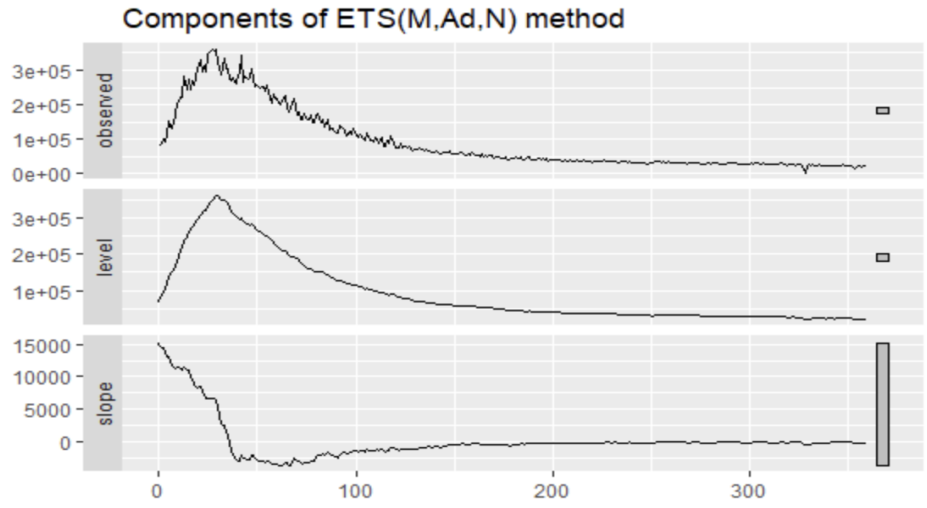


Figure 1: Components of ETS(M,Ad,N) method

3.2 Forecast the internal

3.2.1 Model Principle and Results

To predict an accuracy interval ,it's ideal for taking all the uncertainties into account. However, it's a complex and impossible problem. Most time series analysis just takes the uncertainty in future innovations into account. Rob J. Hyndman simply the prediction model and internally into this short form:

$$y_{n+h|n} \equiv y_{n+h}|x_n \tag{4}$$

The equation(4) means that the prediction distribution in n steps is a distribution of a future value of the series given the model, its estimated parameters and the state vector at the last observation. The forecast mean is denoted by the equation(5), and the equation(6) denotes the forecast variance.

$$\mu_{n+h} = E(y_{n+h}|x_n) \tag{5}$$

$$v_{n+h} = v(y_{n+h}|x_n) \tag{6}$$

In our ETS(M,Ad,N) model forecast variance[3] is given by

$$v_{n+h|n} = (1+\sigma^2)\theta_h - \mu^2_{n+h|n} \tag{7}$$

Where

$$\theta_1 = \mu^2_{n+1|n}, \theta_h = \mu^2_{n+h|n} + \sigma^2 \sum_{j}^{h-1} c_j^2 \theta_{h-j} \tag{8}$$

$$\mu_{n+h|n} = l_n + \phi_h b_n, c_j = \alpha + \beta \sum_{i=1}^{j} \phi_i \tag{9}$$

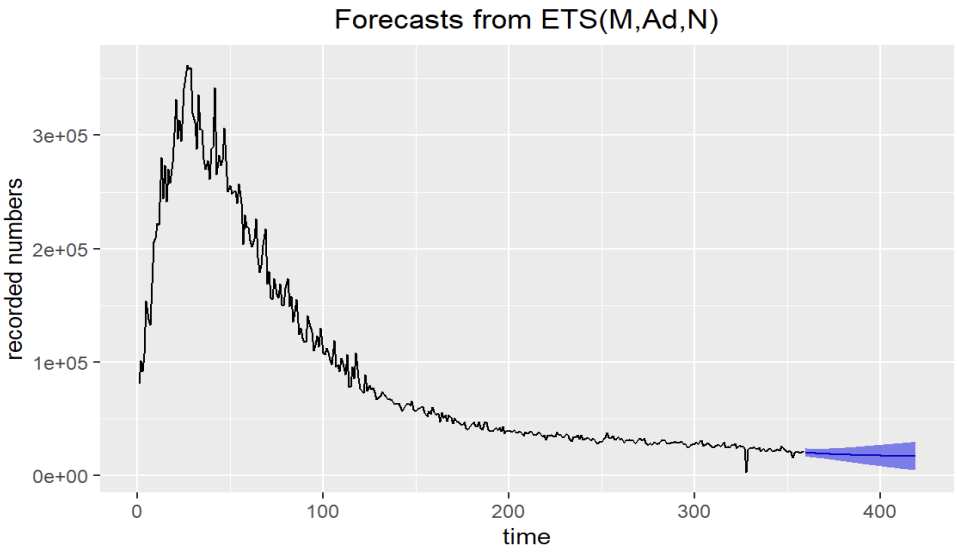


Figure 2: Forecasts from ETS(M,Ad,N)

The prediction distribution above is Gaussian, as ETS(M,Ad,N) model is linear and the errors are Gaussian. Hence, we can calculate η % prediction

internal from the calculated forecast means and variance through $\mu_{n+h|n} \pm z_{\frac{\alpha}{2}}\sqrt{v_{n+h|n}}$, α here is constant.

In practice, we employ the forecast() function in R to simplify the calculation process and get the following results. (The following results are calculated by algebraic rather than simulated data).

The results show that the data decline slowly and downward trend tends to be stable within the predicted coming 60 days. The prediction interval gradually increases, and the diagram shows the 90% prediction interval calculated from the 0.05 and 0.95 quantiles.

3.2.2 Prediction for the example

Table 3: exact value and interval on March 1_{th}

exact value	**interval**
17097.8	(5747.029, 28448.5)

3.3 Words' possible affect on percentage of Hard Mode

We divide the whole words into two parts: one is that words have the same letters, and the other is that words' letters differentiate from each other. And draw them respectively in the figure below. The figure shows words with the same letters have a subtle boosting effect on hard mode percentage since the red line is above the blue line. Thus we can guess the former kind of word is more complicated.

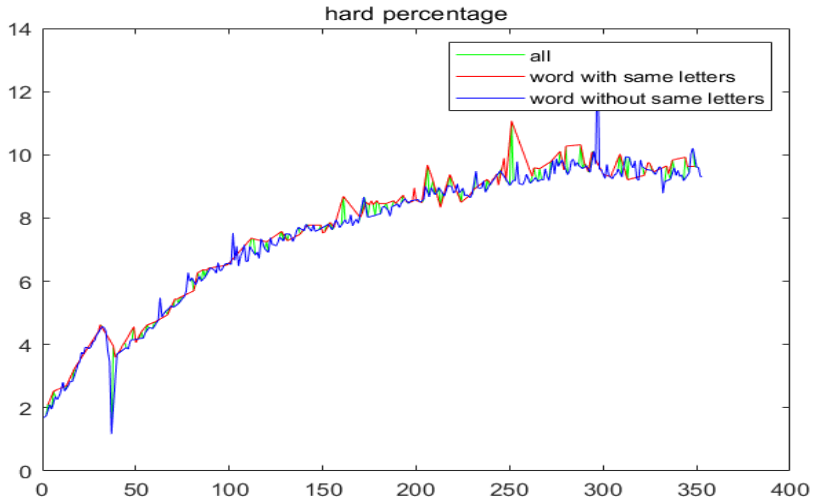


Figure 3: Hard percentage of different class words

4 Applying Machine Learning to predict tries rate

In this task, we need to find a solution that can predict 7 results based on 2 inputs–Date and word. In a word, find a mapping function that can:

$$f[Data, word] \Longrightarrow [1tries, ..., 7tries]$$

So we construct a BP neural network as our mapping function

4.1 processing the data

Since some words in the chart mainly consist of 5 letters, we neglect those words that length less or more than five.

Each row in the data.csv consists of words, reported results and rate of tries.But the data type of the words is string,meaning those string values can't be directly put into our model to compute can predict. The first step is to convert the string type into one that can be calculated.. So we establish a dictionary that can map letters to numbers. here is our $dict$:

Table 4: our dictionary

a	b	c	d	e	f	g	h	i	j	k	l	m
1	2	3	4	5	6	7	8	9	10	11	12	13
n	o	p	q	r	s	t	u	v	w	x	y	z
14	15	16	17	18	19	20	21	22	23	24	25	26

Then we have our dictionary, we use $F_{ids}(X)$ to map letters to numbers (index of the letter), the figure below is an example of how the function works on the word "manly".

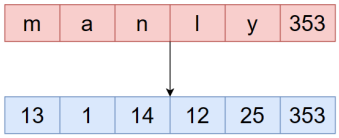


Figure 4: how the F_{ids} works

4.2 Preparing Data Set

After changing words into a list of integer numbers, we can establish our input data. According to the task, we need to predict the rate of each tries for a given word on a future date. Thus we consider each row of data as a sample; its contest numbers are on behalf of its date. Since time is a cumulative value,

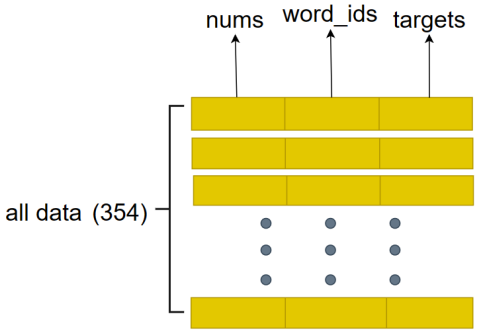


Figure 5: data set structure

the more significant the contest number, the more the cumulative effect of time. We include this value in the input data set as a sample characteristic.

- **Nums** : the contest numbers of a sample
- **word_ids** : a matrix consists of 354 samples, each row includes 6 elements which are 5 indexes and the contest number
- **targets** : a matrix consists of 354 samples, each row includes the rate of each tries.

4.3 Build the predicting model

4.3.1 Structure of the model

After finishing the preparation task above, we used the python package called pytorch to make our model. The model consists of 2 fully connected layer; each layer has a weight matrix(the hidden matrix in Figure 3), the shape of the first matrix is [6,14], and the other one is [14,7]. Also, between the two, we set a ReLU layer as the activating layer.

Here is how the ReLU layer works:

$$F(X) = \begin{cases} 0, & x \leq 0 \\ X & x > 0 \end{cases}$$

4.3.2 How the model works

The whole date set includes 354 samples; since it's too big for a single computation, we divide all the samples into many batches, which size 16. Then our model needs to calculate 16 samples in an iteration. A batch is the input for the model; We set it to **X**, whose shape is [16,6].

Then multiply matrix **X** and W_1(hidden matrix in Figure 3). Get the h_1(hidden state in Figure 3). After h_1 goes through activating layer, multiply h_2 and W_2 then get prediction of the model.

$$h_1 = XW_1 \tag{10}$$

$$h_2 = F(h_1) \tag{11}$$

$$P = h_2W_2 \tag{12}$$

Next step: calculate the Cross Entropy loss of the P and T, x_i and y_i here are elements in matrix P and T.

$$F_{loss}(P,T) = -\sum_{i=1}^{n} P(x_i)log(\frac{\exp T(y_i)}{\sum_{i=1}^{n} \exp T(y_i)}) \tag{13}$$

Then do the back propagation to optimize the weight matrix

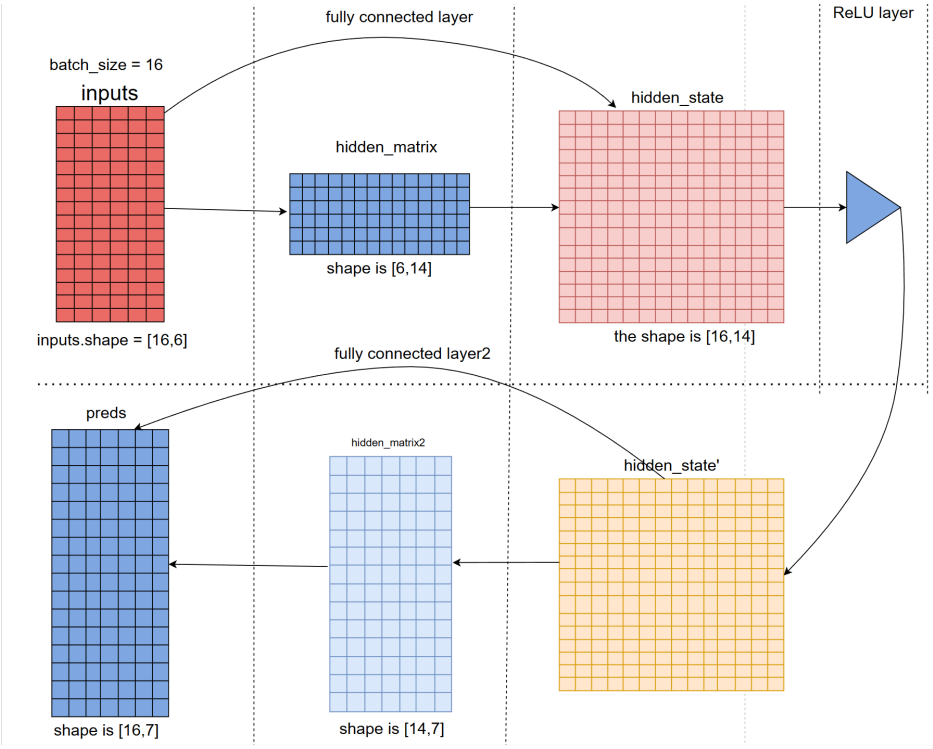


Figure 6: neural network structure

4.4 Train And Model Evaluating

To evaluate the model, we record the average loss of an epoch(a total iteration including all samples), and after 40 epochs, we draw the curve above.

From the curve, we can see that loss is declining overall. From more than 15 to less than 2, and converging to 1.8. which means the prediction is more and more accurate during the iteration.

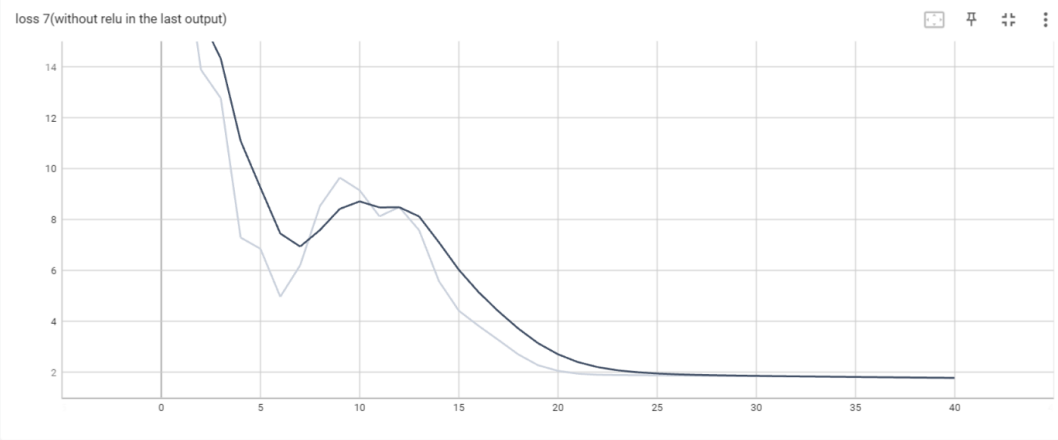


Figure 7: loss curve of each epoch

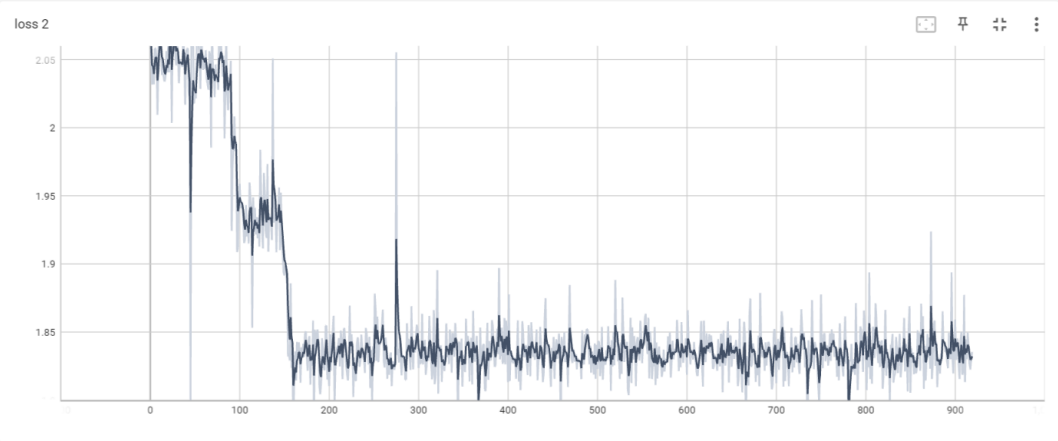


Figure 8: loss curve of each batch

4.5 Prediction Results and Analysis

Applying the model to the given word, our model computes the following result(in Table 3). The rate of 4 tries is the highest, 27.12%, and rate of 1 try is the lowest (5.7%); those results are similar to the previous data, which indicates the model is reliable. But according to the description of the wordle game, its says each word used for guessing is an actual word, which means there are some relationships between letters, all 5 letters are not independent of each other. And we only use fully connected layers when building the model; this kind of structure can't extract the characteristic between the letters. This is the uncertainty associated with our model and predictions.

1 tries	2 tries	3 tries	4 tries	5 tries	6 tries	7 or more tries
0.0570	0.0926	0.2037	0.2712	0.1495	0.1396	0.0864

Table 5: the predict result (percentage)

5 Determinate the Criterion of Difficulty Classification

5.1 Model selection

In the data given, we believe that when the player answered correctly, the distribution of the results of the number of times used reflected, The difficulty of solving the word. In other terms, if the player generally spends more time solving the problem, you can think of this word as more complex. As shown in the figure below, the length of each color piece represents the rate of each kind of tries. From left to right, the deep blue piece stands for the ratio of 1 try, and the deep red piece stands for seven or more tries. Players take more time than' crank' to get the answer–'gorge', so it can be considered that 'gorge' is harder than 'crank'.

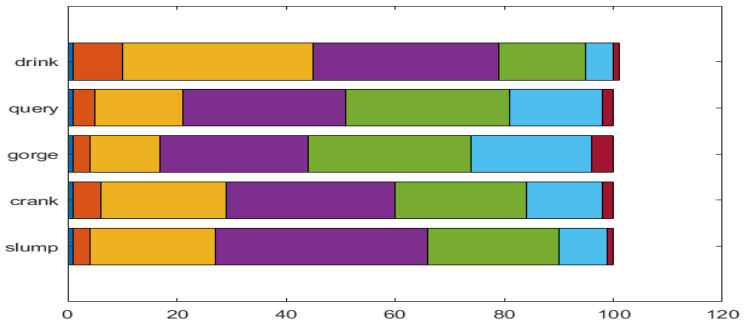


Figure 9: 5 example words,and its tries proportion

We need a more objective method of calculating weights to predict the difficulty. So we introduce the **entropy weight method**. This illustrates the weight impact of each data on the final score. After that, each word is scored and evaluated. Eventually, difficulty categories are divided based on the score results.

5.2 Model Building and Solving

Because the more times it takes to get the result, the more complex the word can be considered, we can put '4', '5', '6', 'x' are set as a positive indicator, '1, '2', '3' times the data is set as positive indicator, and through the following formula, for it separate standardization is required. At this point, all indicators become the higher the score, the greater the difficulty

$$f(X) = \begin{cases} \frac{X_{ij} - min(X_i)}{max(X_i) - min(X_i)}, \text{positive indicator} \\ \frac{max(X_i) - X_{ij}}{max(X_i) - min(X_i)}, \text{negtive indicator} \end{cases} \tag{14}$$

After that, we can get a matrix of standard data – $R = (r_{ij})_{m*n}$. r_i stands for the indicator i of a sample, the entropy is E_j:

$$E_j = -\frac{1}{lnm}\sum_{i=1}^{m} p_{ij} ln p_{ij} \tag{15}$$

And the weight of each indicator is:

$$w_j = \frac{1 - E_j}{\sum_{j=1}^{n}(1 - E_j)} \tag{16}$$

Then, using the formula, for each indicator, calculate its information entropy E_j. Then using the information entropy E_j, according to the formula, the weights of 1 7(x) are respectively [1.412,2.357,5.419,3.408,8.882,20.181,58.34], the table below shows the results.

negative			positive			
1 tries	2 tries	3 tries	4 tries	5 tries	6 tries	7 or more tries
1.4	2.36	5.42	3.41	8.88	20.18	58.34

Table 6: weight results of each tries

Using the weights, linearly multiplication yields the final score, and the formula to compute the scores is $S(p)$; we define the attempt times vector as p,which contains all 7 kinds tries percentage of a sample.

$$S(p) = \sum_{i=1}^{7} w_i p_i \tag{17}$$

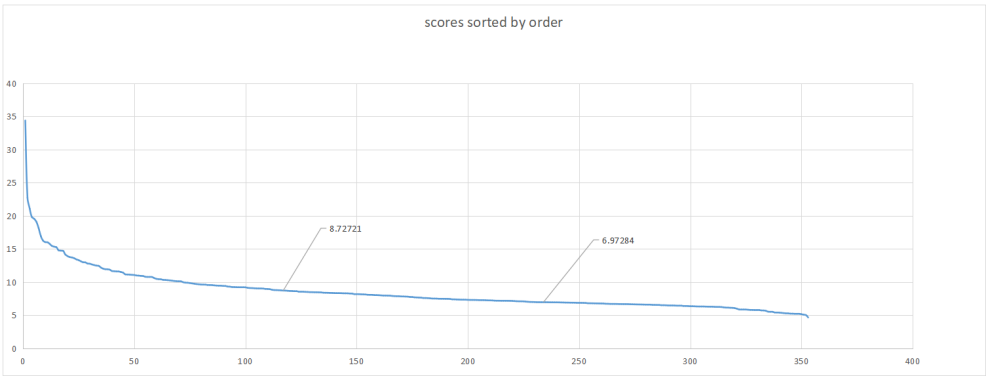


Figure 10: scores sorted by order

After attaining all the scores, we rank all samples by their scores and sort them by order; the figure shows scores of most words stay between 5 and 10.

We can get the difficulty classification criteria for this: scores within [0,6.972] are easy, scores within [6.972,8.727] are normal, and scores above 8.727 are hard.

5.3 Further thinking

The solution offered above gives us a way to judge whether a word is easy, normal or hard. But to get the score, we should acquire the percentage on each kind of tries of a word. However, can we judge a word's difficulty just by the word itself? Here comes our classification model, based on the BP neural network built in the second task. We try to apply it to classify the words.

$$f[word] \rightarrow [easy, normal, hard]$$

5.3.1 Model Modification

The model built in the second question can output 7 results; since in this question we need to classify words into 3 class, we change the model output dimension to 3, still using two fully connected layers and a ReLU layer. Then we record the average loss per epoch and iterate the whole data set 1000 times. Also we use 70% of the data set to train the model and 30% of the data set to calculate the accuracy.

5.3.2 Model Evaluating

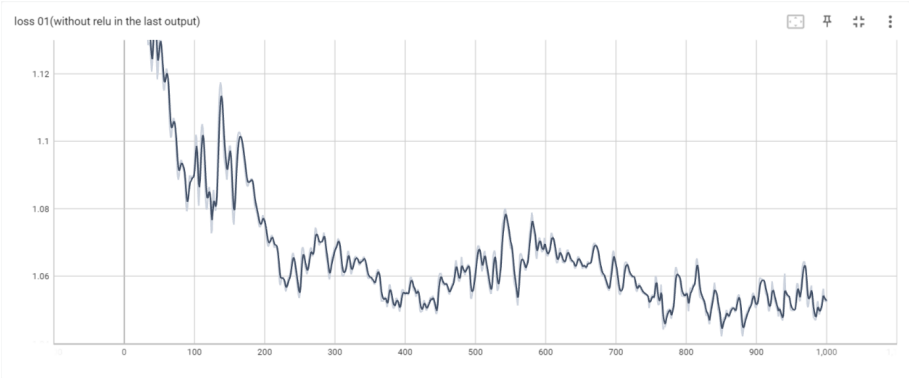


Figure 11: loss of modified model

Figure 8 shows the loss curve of each iteration; the curve fluctuates very much and doesn't converge to a value; nonetheless, the loss decline as a whole. And its accuracy can be up to **71.2%**. Not very high, but it still works.

5.4 examples

5.4.1 predicting result

As predicted by the second question, "EERIE" requires the percentage that guessed the word [0.0570, 0.0926, 0.2037, 0.2712, 0.1495, 0.1396, 0.0864], multiplied by the weights and scored as 11.5125, above 8.727, so we consider it difficult. And we put the data into the modified model, and it classifies the word "EERIE" as "normal." So it is different from the result computed by EWM.

5.4.2 analysis

Prediction of our modified model didn't match the result calculate by EWM, which means the model need to be improve. and we conclude some reasons that may cause the not-good performance.

- The data set is too small, our neural network model can't fully optimize its parameters
- Our range selection is objective. Can't classify and tag each word appropriately.
- The numbers of network layers too small.
- The training strategy is not good enough to make loss converge stably.

6 Features of Data Set

6.1 Reported Results Curve

At the beginning of the Curve shown in figure 1, the reported results keep increasing dramatically. In this stage, we guess that people are interested in the newly appeared game and want to have a try. Then as time keeps moving, the curve decreases sharply, which shows people begin to lose interest in this game. About 200 days later, the gradient reduces, and the curve becomes steady, the tendency to converge to a value.

6.2 Hard Mode Proportion Curve

Once we get the data set, we compute the Proportion of the number of people who choose the hard mode and draw its curve for the raw scatter diagram, the proportion increases as time goes by, but the gradient seemly keeps declining. Through Matlab, we find a formula to fit the curve.

$$y = ae^{bx} + ce^{dx} \tag{18}$$

parameter	value	confidence interval
a	$1.909 * 10^5$	$(-2.561 * 10^{13}, 2.561 * 10^{13})$
b	-0.002659	$(-22.54, 22.54)$
c	$-1.909 * 10^5$	$(-2.561 * 10^{13}, 2.561 * 10^{13})$
d	-0.002659	$(-22.54, 22.54)$

Table 7: parameters' value and confidence interval

To evaluate the formula's accuracy, we calculate each data point's residual. The figure below shows the residual of each data point fluctuates around 0.

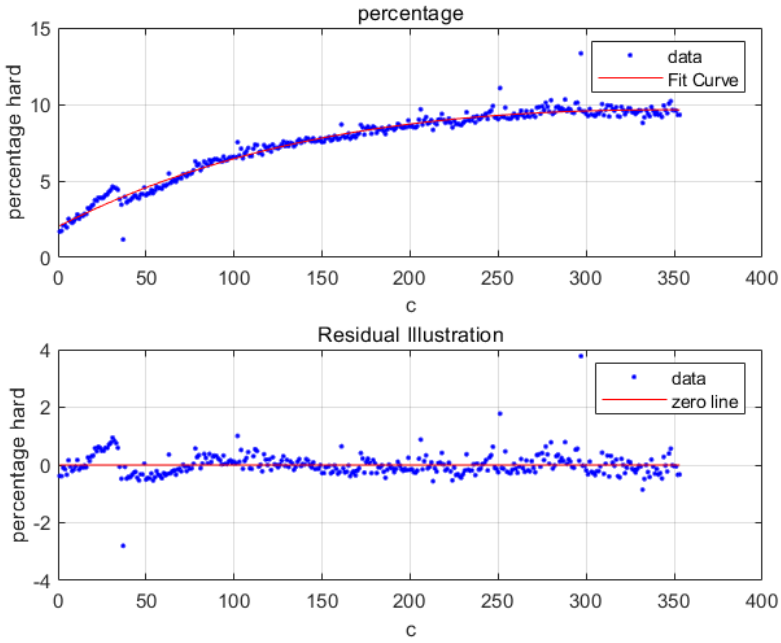


Figure 12: curve figure and residual

6.3 Average EWM Scores for each letter

Firstly, we select those words with a specific letter. and compute the average EWM scores of all the terms, the results shown in the figure below:

On the x-axis, we use numbers 1-26 to represent the letters 'a' to 'b.' The figure shows that words with the letters 'j' and 'z' have the highest average EWM scores. Thus those words may have a higher difficulty level. We guess it may be due to the frequency people use it daily. So we find a chart of letter frequency. Fortunately, it certificates our assumption.

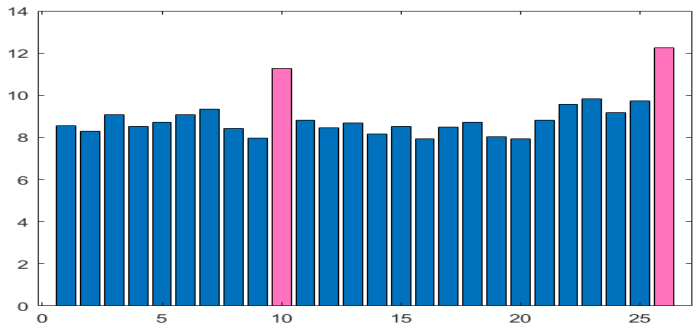


Figure 13: Average EWM Scores

Letter	Relative frequency in the English language[1]			
	Texts		Dictionaries	
Q	0.095%		0.19%	
J	0.15%		0.21%	
X	0.15%		0.27%	
Z	0.074%		0.44%	
W	2.4%		0.91%	
K	0.77%		0.97%	

Figure 14: letter frequency[4]

7 Strengths and weaknesses

7.1 Strengths

- **ETS model can be easily solved by algebra** The model are linear and ε_t is assumed to be Gaussian, $y_{n+h}|x_n$ is also Gaussian. Therefore, prediction intervals are easily obtained from the forecast means and variance.
- **Prediction of ETS is robust** The ETS consider a total message of the given data set, and give a appropriate results.
- **A clear data processing strategy** In the second task,we make a clear and general strategy for data pre-processing. It can be used for every word and replaced with a computable vector.

7.2 Weaknesses

- **Model overfitting** Since the whole data set only have 354 samples, our model may only do an excellent job in this small data set. Doesn't have

a broad application. And this BP neural network encounters a problem in task 3. it's not a mature model for such classification and needs to be improved.

- **Limitations of the entropy weight method** If the change in the indicator value is small or suddenly large or small, the entropy weight method has limitations. In the given data set, the percentage of 1 try doesn't fluctuate very much. So it may have some lousy effect on the classification.
- **Accuracy need to be imporve** in the third task, our modified model only have 71.2 %., not very high.

8 A letter to Puzzle Editor

Dear,editer

Congratulation on the success of your company's game. Undoubtedly, we all wish this great game could get better. To help manage your game better, we've built models to estimate the number of total players for the future and predict the puzzle's difficulty. In a nutshell, our model works in the following three aspects:

- Read the past data and give a prediction of the number of the reported result in one day.
- Predict the distribution of the reported results, and verify their accuracy for a given solution word.
- Use the past data to classify solution words by difficult and describe the difficulty of the word.

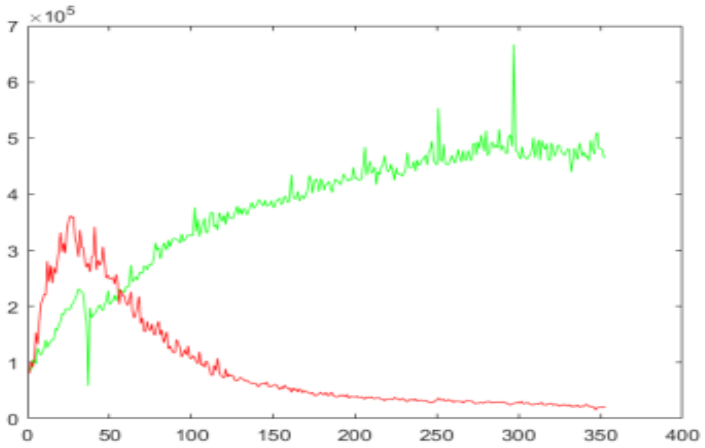
For function 1, it;s evident that there is a clear relationship between the results and the data. Because of this, we adopt the method of temporal sequence to predict the results. In this way, business activity is arranged better, and puzzles can be improved by analyzing anomalies. For example, you can conjecture or verify what types of words are more challenging by those anomalies. Is it more complex if the word has two or more of the same letters ? What about some unique letters? We can use these predicted results to ameliorate the puzzle for player.

As for the second aspect, we choose back propagation neural network to manage it. It is gratifying that we can now predict the distribution of the reported results with some accuracy. According to my model, it will give you an accurate prediction of the puzzle.

Ultimately, we use the entropy weight method to classify solution words by difficulty. According to our calculations, we've divided the existing scores into four difficulty levels. With the help of this classification, we can better consider the difficulty of the word and adjust it to leave the player with a better game experience.

We also found several interesting phenomena in establishing and using our models. First of all, as time goes on, while the number of players has declined, the proportion of players choosing infinite mode has increased. Secondly, the solution word, which contains the letter 'j' or 'z,' would be more difficult for

the player. Lastly, the word with the same letters would affect the percentage of reported scores played in Hard Mode. We sincerely hope these models will



help your company to manage "Wordle." Bless your company can get better and better.

Looking forward to your reply, we'd be glad if we could help you more.

Team: 2306943

References

[1] Chatfield, C. (2004). The analysis of time series: An introduction (6th edition). Boca Raton7 Chapman & Hall/CRC Press.

[2] Gardner, E. S. (2006). Exponential smoothing: The state of the art – Part II. International Journal of Forecasting, 22,637 - 666.

[3] Hyndman, R. J., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). Forecasting with exponential smoothing: The state space approach. Berlin: Springer-Verlag.

[4] https://en.wikipedia.org/wiki/Letter_frequency

Appendices

Appendix A code

Here are simulation programmes we used in our model as follow.

A.1 Main function

Input Python source:

```
import argparse
from data_loader import save_dataset
from trainer import trainer

def main(args):
    data = [5,5,18,9,5]
    Trainer = trainer(args=args)
    #Trainer.train()
    Trainer.load_model()
    Trainer.test(data)
    print('all_done')


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--data_dir", default="./data/saved_train_e",
                        type=str,
                        help="The input data dir")
    parser.add_argument("--batch_size", default=16,
                        type=int,
                        help="Batch size for training.")
    parser.add_argument("--mode", default=0,
                        type=int,
                        help="mode for training.")
```

```
    parser.add_argument("--eval_batch_size",
                        default=64, type=int,
                        help="Batch size for training.")
    parser.add_argument("--learning_rate", default=4e-4,
                        type=float,
                        help="The initial learning rate for Adam.")
    parser.add_argument("--num_train_epochs", default=1000.0,
                        type=float,
                        help="Total number of training epochs to perform.")
    parser.add_argument("--dropout_rate", default=0.5,
                        type=float,
                        help="Dropout for fully-connected layers")
    parser.add_argument("--input_dim", default=5,
                        type=int,
                        help="input_dim for training.")
    parser.add_argument("--hidden_dim", default=14,
                        type=int,
                        help="hidden_dim for training.")
    parser.add_argument("--output_dim", default=7,
                        type=int,
                        help="output_dim for training.")

    args= parser.parse_args()
    main(args)
```

A.2 Dataloader

Input Python source:

```
import torch
from torch.utils.data import TensorDataset
import os
import csv
import numpy as np
abc_map = {
        'a':1,
        'b':2,
        'c':3,
        'd':4,
        'e':5,
        'f':6,
        'g':7,
        'h':8,
        'i':9,
        'j':10,
        'k':11,
        'l':12,
        'm':13,
        'n':14,
        'o':15,
        'p':16,
        'q':17,
        'r':18,
        's':19,
        't':20,
        'u':21,
        'v':22,
```

```
        'w':23,
        'x':24,
        'y':25,
        'z':26
    }
def load_data():
    root_dir = './'
    data_dir = os.path.join(root_dir,'data','data_sq2.csv')

    pos = 6
    nums_id = []
    cat_ids = []
    words = []

    all_trys = []
    targets = []
    with open(data_dir,'r') as fp:
        for i,line in enumerate(fp):
            if i == 0:
                continue
            line = line.strip().split(',')
            nums_id.append(int(line[0]))
            word = []
            target = [0,0,0]
            if float(line[-1]) <= 6.972:
                target[0] = 1
            elif float(line[-1]) <= 8.727:
                target[1] = 1
            else:
                target[2] = 1
            for letter in line[3]:
                assert len(line[3]) == 5,'error with the len of the word!'
                word.append(abc_map[letter])
            targets.append(target)
            cat_ids.append(word+[int(line[0])])
            words.append(word)
            all_trys.append([int(k)/100.0 for k in line[pos:pos+7]])
    all_nums_ids = torch.tensor(nums_id)
    all_trys_rate = torch.tensor(all_trys)
    all_words_ids = torch.tensor(words)
    all_cat_ids = torch.tensor(cat_ids)
    all_target_ids = torch.tensor(targets)
    dataset = TensorDataset(all_nums_ids,
                            all_words_ids,
                            all_trys_rate,
                            all_cat_ids,all_target_ids)
    return dataset

def save_dataset():
    file_name = "saved_{}_{}".format('train','e1')
    save_file_to = os.path.join('./','data',file_name)
    if os.path.exists(save_file_to):
        dataset = torch.load(save_file_to)
    else:
        dataset = load_data()
        torch.save(dataset,save_file_to)
    return dataset
def export_data():
```

```
    data_dir = os.path.join('./','data','data_sq2.csv')
    filename = './data/modify_data.csv'
    fp0 = open(filename,'w',newline='',encoding='utf-8')
    writer = csv.writer(fp0)
    with open(data_dir,'r') as fp:
        for i,line in enumerate(fp):
            if i == 0:
                continue
            word = list(np.zeros(26))
            line = line.strip().split(',')
            for letter in line[3]:
                assert len(line[3]) == 5,'error with the len of the word!'
                word[abc_map[letter]-1] += 1
            word.append(line[5])
            writer.writerow(word)


if __name__ == "__main__":
    #dataset = save_dataset()
    export_data()
    print('over!')
```

A.3 Trainer

Input Python source:

```
from data_loader import save_dataset
from torch.utils.data import DataLoader, SequentialSampler
from model import linear_model,classficer
import torch
from torch import optim
from tqdm import tqdm,trange
from torch.utils.tensorboard import SummaryWriter
import torch.nn as nn
import os
import numpy as np

class trainer(object):
    def __init__(self,args) -> None:
        self.args = args
        self.dataset = save_dataset()
        if args.mode == 1:
            self.model = linear_model(self.args)
        else:
            self.model = classficer(self.args)
        if torch.cuda.is_available():
            self.device = torch.device('cuda:0')
        else:
            self.device = torch.device('cpu')
        self.loss_fn = nn.CrossEntropyLoss().to(self.device)
        self.model.to(self.device)
        self.tb = SummaryWriter('./graph')


    def train(self):
        self.model.train()
        train_sampler = SequentialSampler(self.dataset)
```

```
        train_dataloader = DataLoader(dataset=self.dataset,
                                      sampler = train_sampler,
                                      batch_size=self.args.batch_size)

        t_batchs = len(train_dataloader)* self.args.num_train_epochs
        self.model.zero_grad()
        t_step = 0
        optimizer = optim.Adam(params=self.model.parameters(),
                               lr=self.args.learning_rate)
        scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer=optimizer,
                                                         T_max=t_batchs)
        print("begin training")
        epoch_iterator = trange(int(self.args.num_train_epochs),
                                desc='epoch iterating')
        for _ in epoch_iterator:
            pbar = tqdm(train_dataloader,desc="Iteration")
            avg_loss = 0
            for i, batch in enumerate(pbar):
                if i>15:
                    break
                batch = tuple(t.to(self.device) for t in batch)
                inputs = batch[1].to(torch.float32)
                preds = self.model(inputs)
                loss = self.loss_fn(preds,batch[-1].to(torch.float32))
                avg_loss += loss.item()

                pbar.set_postfix(loss = loss.item())
                loss.backward()
                optimizer.step()
                #scheduler.step()
            t_step += 1
            self.tb.add_scalar('loss 01(without relu in the last output)',
                               scalar_value=(avg_loss/((354*0.7)//self.args.batch_size)),
                               global_step=t_step)
        self.save_model()

    def test(self,data):#the data is a list type with key 'word_ids + date_ids',
        self.model.eval()
        data = torch.tensor(data).to(torch.float32)
        data.to(self.device)
        with torch.no_grad():
            preds = self.model(data)
            preds = nn.functional.softmax(preds)

        print(preds)

    def evaluate(self):
        self.model.eval()
        train_sampler = SequentialSampler(self.dataset)
        train_dataloader = DataLoader(dataset=self.dataset,
                                      sampler = train_sampler,batch_size=32)
        total = 0
        gb_step = 0
        with torch.no_grad():
            pbar = tqdm(train_dataloader,desc="Iteration")
            for i,batch in enumerate(pbar):
                if i<8:
                    continue
```

```
                gb_step += 1
                inputs = batch[1].to(torch.float32)
                preds = self.model(inputs)
                max_preds = np.argmax(preds,axis=1)
                max_targets = np.argmax(batch[-1],axis=1)
                total += np.sum((max_preds == max_targets).numpy())
        acc = total/(gb_step*9)
        print(acc)
        return acc

    def save_model(self):
        torch.save(self.model,'model_for2.pth')
        print('the model is saved in flle')

    def load_model(self):
        filename = 'model_for2.pth'
        self.model = torch.load(filename)
        self.model.to(self.device)
        print('model_loaded')
```

A.4 Model

Input Python source:

```
import torch.nn as nn
class linear_model(nn.Module):
    def __init__(self,args) -> None:
        super(linear_model,self).__init__()
        self.fc1 = nn.Linear(args.input_dim,args.hidden_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(args.dropout_rate)
        self.fc2 = nn.Linear(args.hidden_dim,args.output_dim)


    def forward(self,x):
        output = self.fc1(x)#x.shape = [16,6]
        output = self.relu(output)
        output = self.fc2(output)
        #output = self.relu(output)
        return output

class classficer(nn.Module):
    def __init__(self,args) -> None:
        super().__init__()
        self.fc1 = nn.Linear(args.input_dim,args.hidden_dim)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(args.hidden_dim,3)

    def forward(self,x):
        output = self.fc1(x)#x.shape = [16,6]
        output = self.relu(output)
        output = self.fc2(output)
        return output
```

Appendix B the whole prediction till March 1^{th}

date ids	exact value	lower bound	upper bound
360	20059.95	16477.269	23642.62
361	19945.62	16328.833	23562.41
362	19834.86	16173.77	23495.94
363	19727.54	16011.909	23443.18
364	19623.57	15843.238	23403.9
365	19522.84	15667.883	23377.8
366	19425.24	15486.088	23364.4
367	19330.69	15298.192	23363.19
368	19239.08	15104.601	23373.56
369	19150.33	14905.771	23394.88
370	19064.34	14702.184	23426.49
371	18981.02	14494.335	23467.71
372	18900.31	14282.719	23517.9
373	18822.1	14067.817	23576.39
374	18746.34	13850.094	23642.58
375	18672.93	13629.988	23715.88
376	18601.81	13407.91	23795.72
377	18532.91	13184.243	23881.57
378	18466.15	12959.34	23972.96
379	18401.47	12733.522	24069.42
380	18338.81	12507.085	24170.53
381	18278.1	12280.296	24275.9
382	18219.28	12053.396	24385.16
383	18162.29	11826.602	24497.98
384	18107.08	11600.109	24614.05
385	18053.58	11374.092	24733.08
386	18001.76	11148.707	24854.81
387	17951.55	10924.091	24979
388	17902.9	10700.368	25105.43
389	17855.77	10477.646	25233.88
390	17810.1	10256.021	25364.18
391	17765.86	10035.577	25496.14
392	17723	9816.386	25629.6
393	17681.47	9598.513	25764.42
394	17641.23	9382.011	25900.45
395	17602.25	9166.929	26037.57
396	17564.48	8953.304	26175.66
397	17527.89	8741.172	26314.61
398	17492.44	8530.559	26454.32
399	17458.1	8321.488	26594.7
400	17424.82	8113.976	26735.66
401	17392.58	7908.035	26877.12
402	17361.34	7703.677	27019.01
403	17331.08	7500.906	27161.25
404	17301.76	7299.724	27303.79
405	17273.35	7100.133	27446.57
406	17245.83	6902.129	27589.53
407	17219.17	6705.707	27732.62
408	17193.33	6510.861	27875.8
409	17168.3	6317.581	28019.02
410	17144.05	6125.857	28162.25