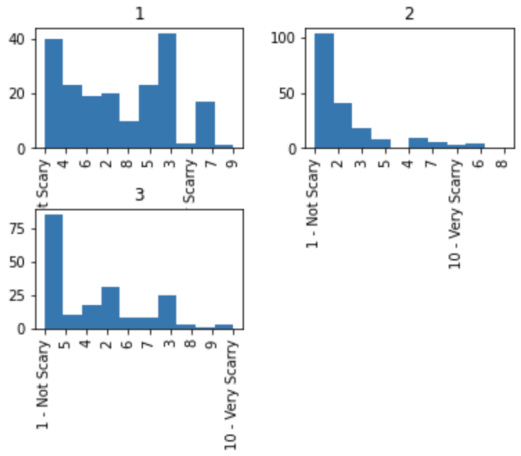
Data:

When we were understanding the data, we performed the following operations on each feature:

q_scary:

The raw data is shown below using histograms.



Some simple interpretation on the correlation to the target, target one has an even distributed input while the other 2 are skewed towards 1 - not scary.

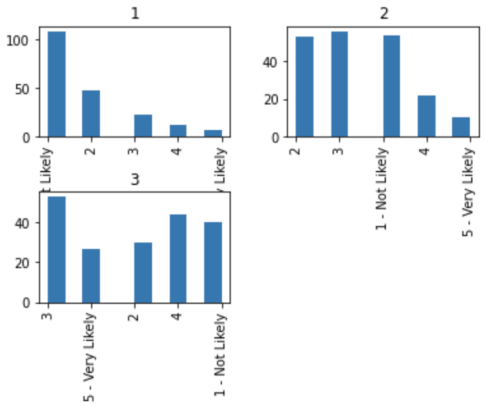
Since the level of scary is different for each person, we decided to encode the feature into onehot layout to see the effect of each input type independently.

Since the input is chosen from 1-10 or empty, so we split q_scary feature into 11 different features starting from q_scary_1 to q_scary_10 (q_scary_-1 means user input is empty). If the user input is 1, then only feature q_scary_1 is 1, and the other 10 q_scary features are 0.

q_scary_-1	q_scary_1	q_scary_2	q_scary_3	q_scary_4	q_scary_5	q_scary_6	q_scary_7	q_scary_8	q_scary_9	q_scary_10
0	1	0	0	0	0	0	0	0	0	0

q_dream:

The raw data is shown below using histograms.



Some simple interpretation on the correlation to the target, target one is skewed towards 1 - not likely while the other 2 are more evenly distributed.

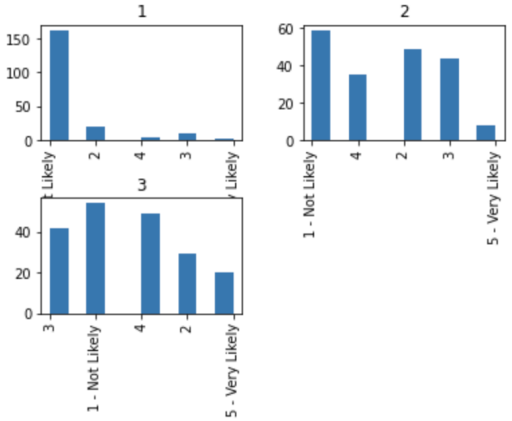
Since people would interpret dream differently, we decided to encode the feature into onehot layout to see the effect of each input type independently.

Since the input is chosen from 1-5 or empty, so we split q_dream feature into 6 different features starting from q_dream_1 to q_dream_5 (q_dream_-1 means user input is empty). If the user input is 1, then only feature q_dream_1 is 1, and the other 5 q_dream features are 0.

q_dream_-1	q_dream_1	q_dream_2	q_dream_3	q_dream_4	q_dream_5
0	1	0	0	0	0

q_desktop:

The raw data is shown below using histograms.



Some simple interpretation on the correlation to the target, target one is skewed towards 1 - not likely while the other 2 are more evenly distributed.

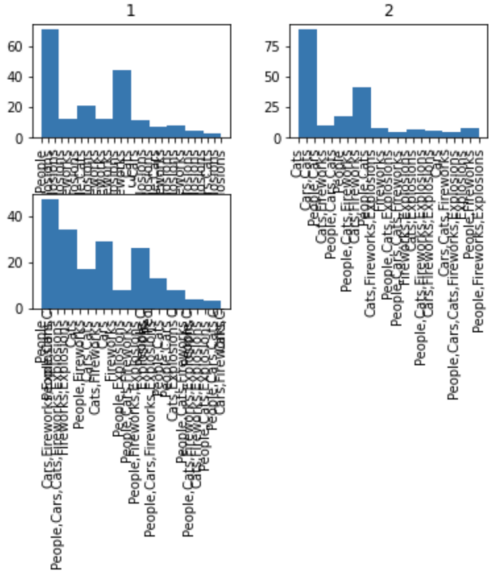
Since people would put stuff on their desktop differently, we decided to encode the feature into onehot layout to see the effect of each input type independently.

Since the input is chosen from 1-5 or empty, so we split q_desktop feature into 6 different features starting from q_desktop_1 to q_desktop_5 (q_desktop_-1 means user input is empty). If the user input is 1, then only feature q_desktop_1 is 1, and the other 5 q_desktop features are 0.

q_desktop_-1	q_desktop_1	q_desktop_2	q_desktop_3	q_desktop_4	q_desktop_5
0	1	0	0	0	0

q_better:

The raw data is shown below using histograms.



Some simple interpretation on the correlation to the target, target one and three is skewed towards people and two is skewed towards cat.

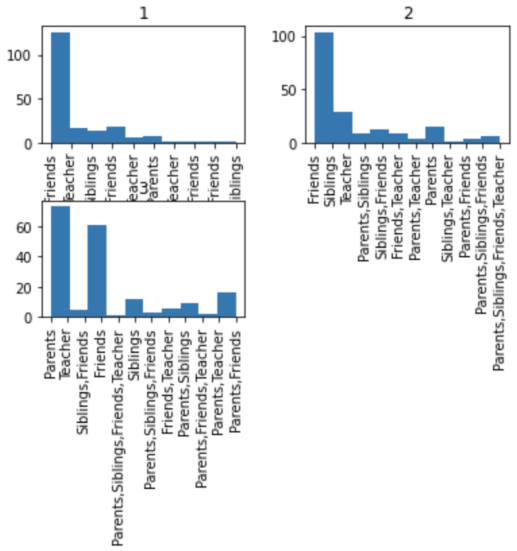
Since each type does not correlate to each other and can have multiple choice, we decided to encode the feature into onehot layout to see the effect of each input type independently and the possible interaction.

Since the input is multiple chosen from “People”, “Cars”, “Cats”, “Fireworks”, and “Explosions”, we split q_better into 5 different features. The new q_better features are q_better_People, q_better_Cars, q_better_Cats, q_better_Fireworks and q_better_Explosions. If the user did not choose anything, then all 5 q_better features are 0. If the user chooses “Fireworks” and “Explosions”, then only q_better_Fireworks and q_better_Explosions are 1, and the rest q_better features are 0.

q_better_People	q_better_Cars	q_better_Cats	q_better_Fireworks	q_better_Explosions
0	0	0	1	1

q_remind:

The raw data is shown below using histograms.



Some simple interpretation on the correlation to the target, target one and two is skewed towards friends and two is skewed towards parents then friends.

Since each type does not correlate to each other and can have multiple choice, we decided to encode the feature into onehot layout to see the effect of each input type independently and the possible interaction.

Similarly to q_better but the q_reminder input is multiply chosen from “Parent”, “Siblings”, “Friends” and “Teacher”, so we split q_reminder into 4 different features. The new q_reminder features are q_reminder_Parents, q_remind_Siblings, q_remind_Friends and q_remind_Teacher. If the user did not choose anything, then all 4 q_remind features are 0. If the user chooses “Parent”, “Siblings” and “Friends”, then only “q_reminder_Parent”, “q_reminder_Siblings” and “q_reminder_Friends” are 1, the q_reminder_Teacher is 0.

q_remind_Parents	q_remind_Siblings	q_remind_Friends	q_remind_Teacher
1	1	1	0

q_quote:

This feature asks the user to rank 5 different quotes or leave some rank of quotes as blank, so we split each quote into 6 different features. If the user thinks the first quote is rank 4, then only rank_1_4 is 1, and rank_1_-1, rank_1_1, rank_1_2, rank_1_3 and rank_1_5 are 0. Therefore, we have 5 * 6 = 30 new features in total. Those new quotes are from rank_1_-1 to rank_5_5. If a user’s rank is quote 1 is rank 4, quote 2 is rank 5, quote 3

is rank 1, quote 4 is rank 3, and quote 5 is rank 2, then only rank_1_4, rank_2_5, rank_3_1, rank_4_3 and rank_5_2 are 1, the rest quotes features are 0.

rank_1_-1	rank_1_1	rank_1_2	rank_1_3	rank_1_4	rank_1_5
0	0	0	0	1	0

rank_2_-1	rank_2_1	rank_2_2	rank_2_3	rank_2_4	rank_2_5
0	0	0	0	0	1

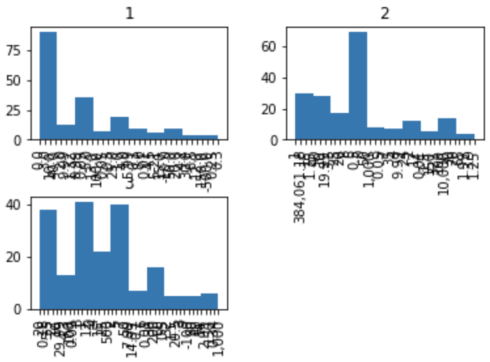
rank_3_-1	rank_3_1	rank_3_2	rank_3_3	rank_3_4	rank_3_5
0	1	0	0	0	0

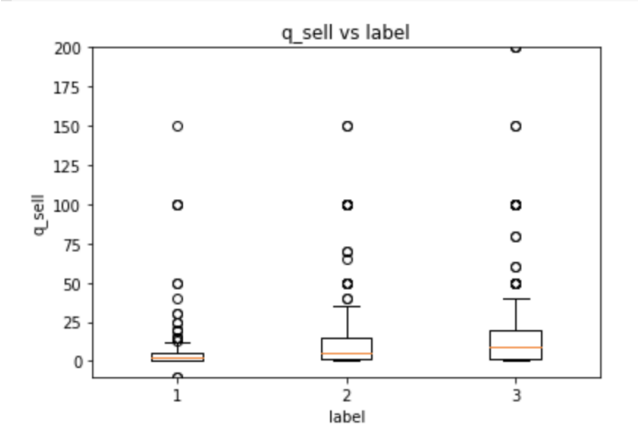
rank_4_-1	rank_4_1	rank_4_2	rank_4_3	rank_4_4	rank_4_5
0	0	0	1	0	0

rank_5_-1	rank_5_1	rank_5_2	rank_5_3	rank_5_4	rank_5_5
0	0	1	0	0	0

q_sell:

The raw data is shown below using histograms.

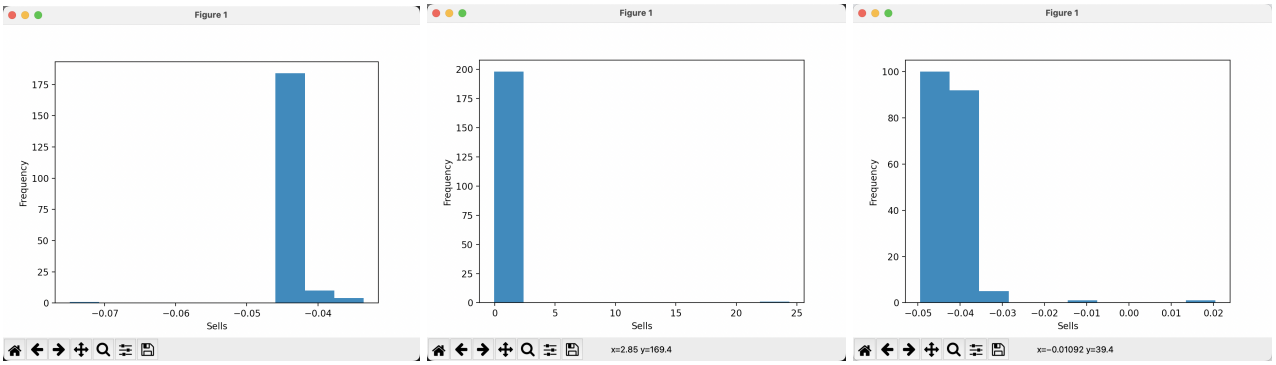




As shown in the boxplot, there are a lot of outliers inside. This feature asks a number to represent how much they think this picture should be. Since the number can be any number within the ranges (384061.18 to -500), in order to improve the performance and training stability and avoid the impact of outlier data, we need to normalize this feature first. We normalize this features by “x’ = (x - mean)/standard deviation” (x’ = (x - μ)/σ). Also, if the user didn’t give a price, we set it to 0.

q_sell	user_id	q_sell	user_id
384,061.18	415138	24.384170543999300	415138

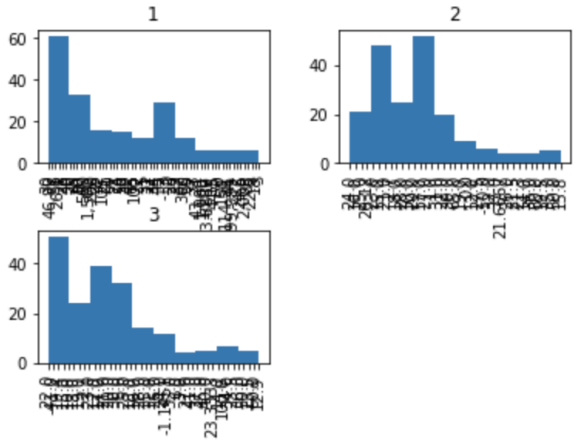
Below are three graphs showing q_sells’ frequency versus money with label 1, 2, 3 respectively after normalization.

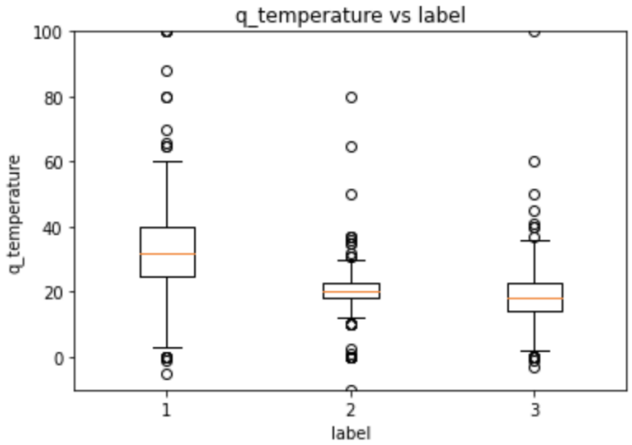


From the graph we can see some clear mean value that falls in to the correspond target.

q_temperature:

The raw data is shown below using histograms.



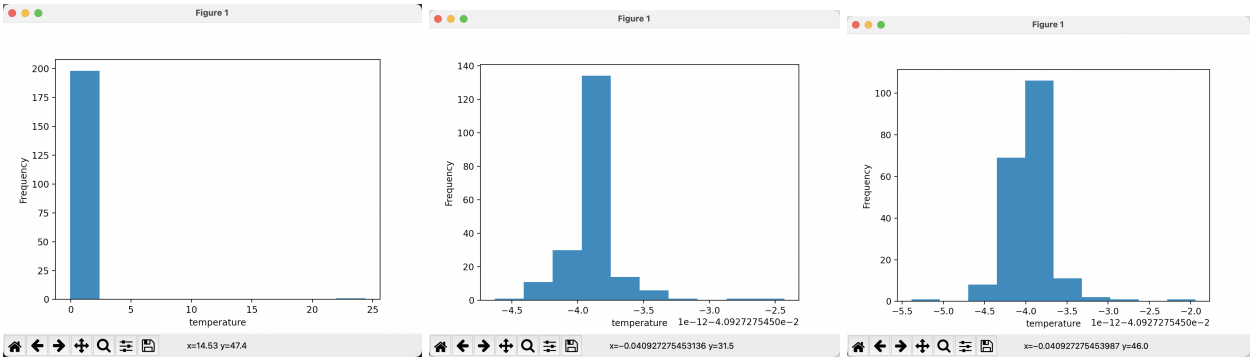


Similarly to q_sell, this feature asks for a number to represent the temperature that this picture feels. Since the number can be any number within the range (999,999,999,999,999 to -41), in order to improve the performance and training stability and avoid the impact of outlier data, we need to normalize this feature first to avoid outliers' influence as shown in the boxplot. We normalize this features by "x' = (x - mean)/standard deviation" (x' = (x - μ)/σ). Also, if the user didn't give a temperature, we set it to 0.

q_temperature	user_id
999,999,999,999	209501

q_temperature	user_id
24.392656170287700	209501

Below are three graphs showing q_temperatures' frequency versus temperature degrees with label 1, 2, 3 respectively after normalization.



From the graph we can see the obvious range of different values in different target.

q_story:

This feature should not be processed since it simply asks the user to write a story for it.

user_id:

We believe user_id should not be used as a feature in our model. The reason is in our survey each user evaluated 3 different pictures, there should be 3 different sample data for each user for each picture. So a user_id should have no relation with our prediction. Instead, we believe user_id should divide the whole data set into the training set, valid set or test_set based on our user_id. For example, for one user_id, all 3 of his sample data should in the same set. Cases where for the same user_id, his sample data for picture 1 is in training ser, picture 2 is in validation set will never happen. The reason we want to put 3 sample data with the same user_id into 1 set is we believe each user has their preference, habit or comprehension of grading level. These hidden relations should not impact our prediction of our model. Since 1 user's measurements are so close together based on those hidden relations, we are essentially "cheating": we have something in our training set that is unreasonably close to the test set. In other words, when we actually deploy the model in practice, the data we wish to make predictions for will not be as close to our training data. Thus, the test performance overestimates the model performance.

user_id	label	label	user_id
310448	1	1	286732
415138	1	2	286732
204814	1	3	286732
410821	1	1	3679
412195	1	2	3679
398257	1	3	3679

Label:

We do not need to modify the label since the label simply tells us the concrete result of this sample set.

Model:

Stage 1:

Since the data type varies, we decided to use different models on different parameters of the data. Train each model individually, and produce a final accuracy from our sample test set. We will then let each model take in the actual test input and found the most occurrence output (e.g. the models output [1,1,3], then we use 1). And after that combine our models together and see the final accuracy performance. In the case of all models producing different outputs, we take the prediction from the most accurate model. In general, we have 2 combined models to train and test our data and compare their accuracy.

Combined Model 1:
We use naive Bayes on q_story and Multilayer Perceptron on everything else. Combine them and test the combined model to see the accuracy.

Combined Model 2:
We use naive Bayes on q_story, Decision Tree on q_quotes and Multilayer Perceptron on everything else. Combine them and test the combined model to see the accuracy.

Stage 2:

We encountered many difficulties when implementing satge1. We are not sure whether the method of stage 1 is feasible or whether the effect is good enough. Therefore, we want to use the sklearn to test different models to see which model should we use:

Random Forest for only Type 2 quote from the sklearn:
After testing with sklearn's random forest function, with input quotes, the output accuracy is around 0.3, which is no better than a random generator. So this route will be inapplicable and the model will not be considered to be a part of the model combination.

MLP for everything except story with the sklearn:
With Multilayer Perceptron, we obtained the highest sample test accuracy of around 0.70 with the ReLu activation function out of all other activation functions. So we will include MLP as part of the model combination.

Random forest for everything except story:

We test both random forest and decision tree with data set of everything except the story. With our sample test set, we obtain higher test accuracy with decision tree of around 0.6 and random forest has an accuracy of 0.56. However, this could be because the number of data in the test set is very little which causes the decision tree to overfit the test set.

Naive Bayes for the story:
Since there are 3 possible outputs for each input, we tried Naive Bayes in 2 ways. The first way is to simply add another k to our prior to make the Bayes Classifier takes in the count of the 3rd case. With this approach, the sample test accuracy is around 0.3, so this model will not be considered.

Double Naive Bayes for the story:
Since there are 3 predictions, and from the lecture we only learned to deal with 2 with Beta distribution, we group the first possible output as 1 and the rest 2 as 2 and use Naive Bayes to compute the probability that the output belongs to which group. If it's the second group of the last 2 predictions, the model will run Naive Bayes again to compute which one of the two outputs it is. With this approach, we still only obtained around 0.3 accuracy so this model will not be considered.

Therefore, our final combined model is gonna be:
we decided to use the multilayer perceptron for every feature except the story and Naive bayes for every feature except q_sell and q_tempurature, and then combine them to form our final model. By combine together, we are referencing to the method in Stage 1 where we take the results and find the most occurrence one.

After implementing and combining our model, we have a really good model with accuracy:

```
accuracy:  0.8676716917922948
```

Stage 3:

We have a really good model with accuracy:

But we also want to try out Logistic Regression, we tried a lot of different gsd and implementation methods, but in the end, it seemed to have little effect on the results, so we gave up this method.
Below is the accuracy of combine model after added Logistic Regression, which is lower then before.

```
accuracy:  0.7989949748743719
```

Model Choice and Hyperparameters:

How we use our data for training and validating:

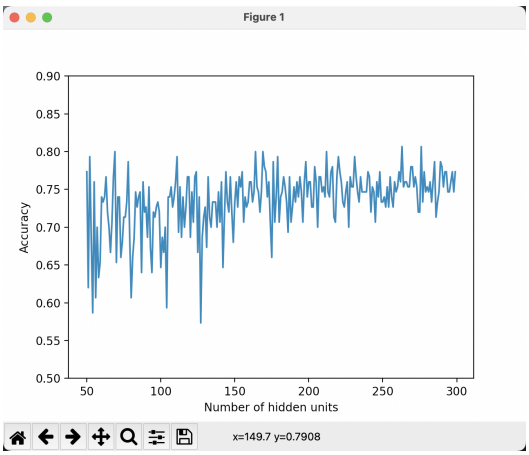
Generating Training and Validation data set as well as the sample test set.
Firstly, we need to make sure that the training and validation set contains an equal amount of output categories.(e.g. Training set can not contain 80% of output with label 1 and the validation set can not contain 80% of output with label 3)
Secondly, we need to make sure that the inputs of the same user_id are in the same data set, this is to prevent user bias, in which a user might perform a bias in the input they give. That is, for a single user, if one of the inputs is in the training set while the others are in the validation set, then the model will mismatch the prediction because a single user will have preference towards some specific answers when making choices.
For the second concern, we would have a list of user_id and randomize it, giving each data set a number of user_id and their corresponding input for output with label 1, 2, 3.
Given the amount of data(597), we split the data set into three parts. 300 for the training set, 150 for the validation set and the rest(147) for our sample test set.

Evaluation metric(s) we used to evaluate our models

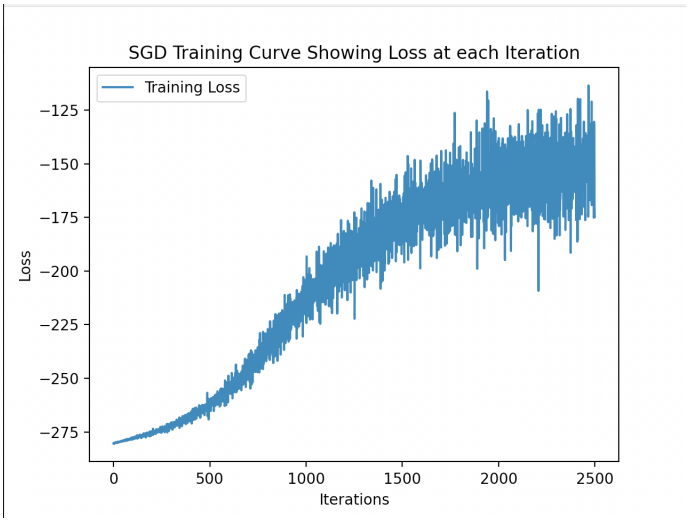
The evaluation metric we use is our sample test accuracy. For any model that has higher than 0.6 accuracy we will consider it into our combination model. This simmulates the actual test and we expect our model to have close accuracy when apply to the actual test.

Multilayer perceptron for every feature except the story:

Referring to the description in the data part, I shuffle the data by user_id randomly and split the data sets into three: 300 data for training set, 150 for validation set, and 147 for test set.
For data column like q_sell and q_tempertuare, because there are some obvious outliers inside, we decided to normalize the data in order to reduce bias. For other data collecting from multichoice questions, I just processed the data to be dummy variables like what is done in the example.
I use a validation set containing 150 data to tune my hyperparameter: the num_hidden size. Since I have testes mlp model using sklearn, so I can fix the range of my num_hidden size to be within 50 and 300. So I use a for loop to store different test accuracies calculated from different num_hidden sizes and find the max one to be my hyperparameter.

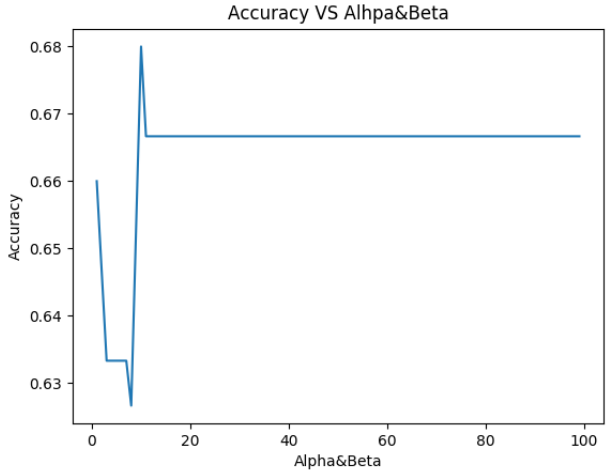


Using the tuned model, we get a training curve showing the number of iterations on the x-axis, and the loss on the y-axis.



Naive Bayes for every feature except q_sell and q_tempurature:

Referring to the description in the data part, I shuffle the data in user_id randomly and split 300 data points to training set, 150 data points to validation set and 147 data points to test set.
Then, I compute every feature into dummy variables except q_story, q_sell and q_tempurature. For q_stroy, we compute the sentences in the form of bag of word. For q_sell and q_tempurature, since it is hard to explain them in 0s and 1s, we decided to not count on them. Now, we have dummy variables and a bag of word metric with data_points of rows. Therefore, we simply concatenate them on axis=1 to be my input metric.
For the validation set, we use it to check how good the different estimation methods are when every word appears is included in the vocabulary. Similar to lab8, we try two approaches which are MLE and MAP. For MAP, we tried Alpha and Beta from 1 to 100 and get the accuracy plot below.



The result shows that the best value for alpha and beta is alpha=beta=10 which gives us validation accuracy of 0.68. Since the MLE estimator has only 0.56 validation accuracy, we decided to use the MAP estimator with alpha=beta=10.

Combining:

For combining the models together, we simply use trained models with the method from **stage 1.** So there is no tuning required.

Prediction:

We expect our model to have an accuracy around 0.75-0.80. Since we dont know how the test set will be, we cant guarantee our accuracy to be an exact number. However, in worst case, our combined model takes the model with highest prediction accuracy on our sample test set which is the Multilayer Perceptron model. This model has an sample test accuracy of 0.76 which is where the 0.75 in our lower bond came from.

Workload Distribution:

Robin:

Deal with the data and code Naive Bayes model. Look for if the logistic regression can explain the data well. Complete the Naive Bayes and logistic regression parts on the report.

William:

Coded and adjust the algorithm for combine model to suit all model provided. Recorded model behaviors to the report. Implemented random forest for quotes.

Chen:

Use sklearn to test different models' accuracy, tune hyperparameters to decide which model and which hyperparamaters should we use for better result.
Process data(everything except "q_story") for MLP and design a MLP model.
Process data again(everything except "q_story") for logistic regression model and work on a logistic regression model to try to find out if it can improve accuracy.
Complete the data, model, model choices and hyperparameters part of the challenge report.

Haike:

Processing data, splitting data based on user_id and recording how we process data into reports. Exploring different activation functions for MLP. Implement validation for MLP to tune hyperparameters.