CS 536 : Final Project - Data Completion and Interpolation

Team member:

Name: Ningyuan Zhang, RUID: 186002871
Name: Lemin Wu, RUID: 171008940
Name: Yunqi Li, RUID: 188006563

Data: Many Labs 3

Data Preprocessing and Selecting：

In this project, according to Prof.Cowan's suggestion and considering the time limitation and that different features may have different importance for data analysis, for example, predicting the name, ID and the time to fill out questionnaires of users are meaningless and inefficient, we only consider to do the data completion and interpolation of the features in the variable codebook which are more important for analysis and more possible to predict with the information we have.

Note that there is an attention check in the ML3 dataset, i.e., if the answer of this question is not "I read the instructions", it represents that the user gave the answers to the questionnaire without reading the questions first, which means that their data is just the noise and will definitely provide bad influence to our model. Thus, we firstly check the content of this feature "attention correct" and drop out all the noise in the dataset.

In the variable codebook, there are 106 features with type "*effect*" (shown as blue in our attached variable codebook) or "*individual difference*" (shown as red). And only 23 of them has "open response" (shown as green) which means most of them are natural language responses with a lot of variability in how people were able to answer questions, so we just leave these features in the bonus. For the remaining part, there are 35 features with type "*effect*" (the features "mcdv1" and "mcdv2" have only five categories in the questions but correspond to seven values so we think the data may have some mistakes and we just ignore them here.) and 46 features with type "*individual difference*". In our project, we choose to use the features with type "*effect*" as our features which need to be predicted or reconstructed from the features that are present and the features with type "*individual difference*" as our latent relevant features. The reasons why we make such decision are as the following: Firstly, after checking the dataset, we find that the "*effect*" features have more missing data and the "*individual difference*" has very few blanks which are not necessarily needed to be predicted. Secondly, after reading the questionnaire, we find that the features with type "*individual difference*" represent the users' evaluation and identification of themselves, such as personality, etc, and the features with type "*effect*" reflect

the users' opinion to other people or events. Thus, from a psychological perspective, we believe we can infer some information of how people consider their surroundings and friends from how they look like themselves, for example, an optimist with high confidence may love his life more than a pessimist. We believe it makes sense to assume that there exists causal relationship between "*Individual Difference*" variables and "*Effect*" features. Thirdly, if we want to predict the value of an "*individual difference*" feature, we can also use the same method we will talk about in the following part to train a new model, so there is no need to train so many models under the same method here which are wasting time and not so meaningful for our aim of our project. There are approximate 1400 records after we preprocess the data and we pick 10% of them as the testing data set, 10% of them as the validation dataset and 80% of them as our training data.

Part 1: Before getting into the specifics of the data, here are some problems we have to address when coming up with a solution:

How to represent or process the data. Data features may contain a number of diverse data types (real values, integer values, categorial or binary values, ordered categorial values, open/natural language responses). How can you represent these for processing and prediction?

We consider doing the data completion and interpolation of the features in the codebook which are more important for analysis. There are real values, binary values, ordered categorial values and natural language responses. When processing real-value data and binary-value data, we could use the original values in the file directly. When it comes to ordered categorial values, the dataset has already transformed each answer with an integer. For instance, in question mood_01, 1 represents Very Happy, 2 represents Happy, 3 represents Somewhat Happy, 4 represents Neutral, 5 represents Somewhat Unhappy, 6 represents Unhappy, 7 represents Very Unhappy and 8 represents Decline to Answer, so we can simply consider that the bigger the number, the unhappier the attitude is and there is no need to encode the answer into a vector, hence we can still use the original values without any preprocess. The most difficult part is how to represent natural language responses which we will talk about in the bonus part. So now, we can see that all the data we have is just a number which can be processed and predicted well in our proposed model.

How to model the problem of interpolation. What are the inputs, what are the outputs? An important if subtle question to consider here - what does it mean to predictor or interpolate a missing feature?

According to the codebook, we could divide all variables into two parts based on whether they belong to "*Effect*" variables or "*Individual Difference*" variables. As we have analyzed above, it makes sense to assume that there exists causal relationship between "*Individual Difference*"

variables and "*Effect*" variables. Here, we need to predict each missing "*Effect*" variables using "*Individual Difference*" variables. In this case, in each model, the input is the values of all "*Individual Difference*" variables while the output is the value of each "*Effect*" variable. More specifically, the input is a vector with the dimension of 46 and the output is a number represents the value of the correspond "*Effect*" variable. And in our models, we not only output a number represents the predicted value of the correspond "*Effect*" variable, but also provide a probability to show how much likely the true value of the correspond "*Effect*" variable is what we predicted here.

And in the same way, if we want to interpolate missing "*Individual Difference*" variables. in each model, the input is the values of all "*Individual Difference*" variables while the output is the value and probability of each "*Individual Difference*" variable. To predict or interpolate a missing feature is just calculate the value of it according to data we have under the model we design and train for it.

Model selection. What kind of model or models do you want to consider?

In this project, for improving the accuracy, we consider using two different models to make the prediction: decision tree model and cross entropy model with one norm regularizer. After we train both of them we can make a comparison of them and choose a better one. The reason why we consider the two models is because that there are too many latent relevant features in the dataset which we can't judge accurately by ourselves whether it is relevant to a specific feature, so our model must have the good ability to recognize the relevant and irrelevant features of a given specific feature which need to be predicted. And both of the two models have the good ability to eliminate irrelevant features: decision tree could prune by depth or sample size, and cross entropy model could apply some one-norm regularizers which can make the weights of irrelevant features more likely to be zero.

Quantifying loss or error. How can you quantify how good a model is, how to measure its loss/error? This is important not only in terms of evaluating your model, but in terms of training as well - how can you refine and improve your model without a way of comparing them?

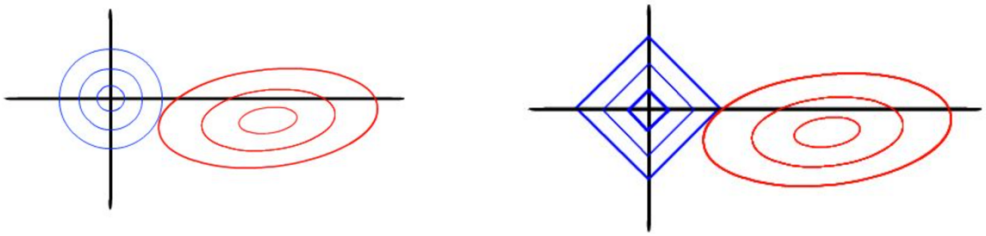
There are approximate 1400 records after we preprocess the data and we pick 10% of them as the testing data set, 10% of them as the validation dataset and 80% of them as our training data. We use the validation dataset to pick the best model trained from the training dataset and then we use the testing dataset to test our model. In our project, we use the test error to quantify how good the model is: if the data type is real value, we use the square mean error: $\text{test\_err} = \| \hat{y} - y \|^2$; If the data type is ordered categorial value, we use the accuracy: $\text{test\_err} = \frac{1}{n}\sum_{i=1}^{n} I(\hat{y}_i \neq y_i)$. And for the cross entropy model, we define a loss function which will be explained in the next part.

Training. What kind of training algorithm can you apply to your model(s)? What design choices do you have to make here?

For the decision tree model, we consider the ID3 Algorithm: For each variable Xi, compute IG(Xi); Select the variable with maximum information gain; Split the data based on that variable; Recursively apply this algorithm on each part of the data. And prune according to the size of the leaf nodes or the depth of the trees finally to get the model. For the cross entropy model, we consider to use SGD to optimize the loss function and get the parameters finally. And We will explain both of the training algorithm in detail in the part 2.

Feature selection. It is frequently useful in learning problems to focus on specific features and exclude others, to try to eliminate spurious features and focus on what matters. How can that be applied here?

Decision tree could realize it by pruning by depth or sample size. Pruning is a technique that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting. We could limit the depth of the tree or limit the minimum size of the leaf nodes to only keep the most relevant features. We could prune by sample size because the less data a split is performed on, the less 'accurate' we expect the result of that split to be. Cross entropy model eliminates spurious features by applying one-norm regularizers. The pictures below explain this property, comparing with the two-norm regularizer shown as the left picture, the one-norm regularizer shown as in the right can more easily to make the weights of the irrelevant features to be zero which can help us to eliminate the spurious features.



Validation. How can you prevent or avoid over-fitting? Can you apply the usual training/testing/validation paradigm to this problem? How do you choose the training or testing data? Note that a record won't need to be complete to still be useful, potentially, in interpolation. Can cross-validation be applied here? This can be especially important when the data set is not overwhelmingly large and data must be used carefully.

Firstly, we use the testing data and validation data in both of our models. And to guarantee the test and validation data have the same distribution, we choose the data randomly. What's more, for the models themselves, we also consider how to avoid overfitting: In decision tree model, we prune the tree by depth and leaf nodes size to avoid over-fitting. And in cross entropy model, we consider to use regularizer to avoid overfitting which will be explained in detail in part 2. We think cross-validation can be applied here since the data set we have is not overwhelmingly large but we must use the data carefully.

Evaluation. How good is your final model? How can you evaluate this? What are the limits and strengths of your model - how many features does a new record actually need to be able to interpolate well?

After we get the final model which has the best loss or error on the dataset, we generate 500 new data by our model and we calculate the proportion of each category of each feature in the new data and we compare the statistics with those calculated from the given dataset to show how good our final models are. The reason we do it is because we believe we can get the similar statistics from the original data and simulated data if the models are good enough. The strengths of our model are that we can recognize irrelevant features better than other models and our models are very simple and really practical which mean that the models can be trained and used to predict the new values easily. But the limits may be that our model is too simple and may get worse results than some complex models such as some multiple layer neural networks. Different feature may have different relevant features, for making our lives easier, we consider all the "*individual difference*" features as our input, so for a new record to be predicted well, we will only need no more than 46 features which depends on its unique model.

Part 2:

In machine learning area, different learning algorithm has different power. For finding the better model to predict the given feature, we consider two reasonable but different models in this part, the first one is the decision tree model and the second one is the cross entropy model. We will show our two different models separately here.

Decision Tree Model:

Describe your Model: What approach did you take? What design choices did you make, and why? How did you represent the data? How can you evaluate your model for goodness of fit? Did you make an effort to identify and exclude irrelevant variables? How did you handle missing data?

We design a decision tree model here as our first model because decision tree model has the power to pick out some real relevant features by pruning and here we want both prediction result

and corresponding probability. We use ID3 to build our tree. For each variable, we need to compute the maximum information gain and select the variable with the maximum information gain; then we can split the data based on that variable and we can build the tree by recursively apply this algorithm on each part of the data. Finally, we can prune the tree according to the size of the leaf nodes or the depth of the trees to get the best model.

Information Content:

$$H(Y) = -\sum_{y} P(Y = y) log P(Y = y)$$

$$H(Y|X = x) = -\sum P(Y = y|X = x) log P(Y = y|X = x)$$

We evaluate the model by calculating training and validating error.

$$err(f) = \frac{1}{m}\sum_{i=1}^{m} 1\{f(x^i \neq y^i)\}$$

We prune the decision tree to identify and exclude irrelevant variables:

- By Depth: The depth of the decision tree generally reflects the overall complexity of the model. Set the maximum depth and once the depth is reached, even if the remaining data set down that branch is not 'pure', terminate and decide Y for that terminal node by the majority decision of the remaining data down that branch.

- By Sample Size: Set a threshold value and once the number of remaining data points falls below that point, terminate growth for that branch and determine a value of Y for that terminal node again by the majority of remaining data.

Describe your Training Algorithm: Given your model, how did you fit it to the data? What design choices did you make, and why? Were you able to train over all features? What kinds of computational tradeoffs did you face, and how did you settle them?

Here we use ID3, which a classical and effective way, to build our decision tree model. After preprocessing the data, we can train over all the features we selected now by our model.

ID3 Decision Tree Algorithm:

- Select variable Xi with most information gain

- Partition the data into n sets

- Recursively construct a decision tree on Xi data

- If, at any point, the remaining data down one branch has the same Y value, set that node to return that Y value

- Recursively until all data is captured by the terminal node of some branch on the resulting decision tree

Describe your Model Validation: How did you try to avoid overfitting the data? How did you handle the modest (in ML terms) size of the data set?

On one hand, we prune the decision tree to identify and exclude irrelevant variables to avoid over-fitting. On the other hand, we consider the behavior of our model on both the training and testing dataset to avoid overfitting. For handle the modest size of the data set, while the raw data set provided is small, it can actually be blown up to quite a considerable data set. Given a complete record, we can create lots of training data by knocking out various subsets of features and trying to match this `knocked out' feature set with the original full feature set to get more data for training the models.

Evaluate your Model: Where is your model particularly successful, where does it lack? Does it need a certain amount of features in order to interpolate well? Are there some features it is really good at predicting and some it is really poor at predicting? Why do you think that is?

After training all the models, we find that we can get good results for some features, but the models don't work good all the time. We find that features with less options perform better. For example, mcmost1-5 (2 options) have quite smaller training and testing errors while bestgrade3-5 (7 options and more) have larger errors. The reason bestgrade5 performs okay is just because most people choose 1 without considering any features. Our whole results can be checked in the folder "output of decision tree model" and we just show some examples here.

Effect	Training Error	Validating Error
mcmost1	0.1252174	0.1958042
mcmost2	0.2173913	0.2027972
mcmost3	0.1041667	0.08333333
mcmost4	0.2118056	0.2986111
mcmost5	0.02083333	0.02083333

Effect	Training Error	Validating Error
bestgrade3	0.7350272	0.8832117
bestgrade4	0.6309091	0.810219
bestgrade5	0.3266788	0.3138686

Analyze the Data: What features were particularly valuable in predicting/interpolating? What features weren't particularly useful at all? Were there any features about the researcher/experimental environment that were particularly relevant to predicting answers - and if so, what conclusions can you draw about the replicability of those effects?

We only take features in the codebook into account due to the limit of time.
Firstly, we try to predict “Effect” features.
The following is relevant features of some effect variables. (Please check results of decision tree models.xlsx for the full table.)

Effect Features	**Valuable Features**
kposition	big5_04,big5_05,big5_10,intrinsic_07,intrinsic_11,intrinsic_12,intrinsic_13nfc_05,pate_01,pate_02,pate_05,selfesteem_01,stress_01,stress_02,stress_04
lposition	big5_02,big5_03,big5_04,big5_05,big5_06,big5_09,intrinsic_06,intrinsic_11,nfc_06,pate_02,pate_04,pate_05,year
nposition	pate_02
rposition	big5_08,gender,intrinsic_02,nfc_04,nfc_06,selfesteem_01,stress_01,stress_02

Actually, we have picked out the features that are more valuable in predicting in data preprocessing. As we have talked about, some features are not necessary and possible to be predicted, for example, predicting the name, ID and the time to fill out questionnaires of users are meaningless and inefficient, so we only consider to do the data completion and interpolation of the features in the variable codebook which are more important for analysis and more possible to predict with the information we have. And in our results, we can see that the feature with less categories may get better accuracy than those have many categories. And some answers are actually come from the same question, so we can predict such features better with the information we have than those almost independent to other features.
In our model, we try to use the “*individual difference*” to predict “*effect*” features, so we just find that how people consider themselves may influence how they consider their surroundings. And we think the researcher’s behaviors and the experimental environment will definitely influence the psychological characteristics of the respondents. For example, if the respondents are tested in a spacious and bright environment by a friendly researcher, they may give more positive answers. Under such case, for decreasing the negative noise caused by the surroundings, we can consider the replicability of those effects in other experiments.

Generate Data: Use your system to try to generate realistic data, and compare your generated data to the real data. How good does it look? What does it mean for it to ‘look good’?

We consider to use decision tree to generate new data. Firstly, we generate “Individual Difference” variables based on their probability distribution in the original dataset. Next, we predict each “Effect” variable, using the decision tree model which is trained previously and taking those “Individual Difference” variables as input.
We evaluate the new data by comparing the distribution of new “Effect” variables with the original data.
Here’s part of the result:
(Please check new data analysis of tree model.xlsx for the full table.)

Old Data		**New Data**	
Effect Features	**Distribution**	**Effect Features**	**Distribution**
kposition	2 0.432117 1 0.567883	kposition	2 0.29 1 0.71
lposition	2 0.465268 1 0.534732	lposition	2 0.47 1 0.53
nposition	1 0.426396 2 0.573604	nposition	1 0.01 2 0.99
rposition	1 0.471056 2 0.528944	rposition	1 0.334 2 0.666
vposition	1 0.490962 2 0.509038	vposition	1 0.48 2 0.52

Features whose data type is binary value, such as lposition, mcmost1-mcmost5 and mcsome1-mcsome5, look good. The probability distribution of the new data is close to the original one. At least the majority choice keeps same. And when it comes to questions with more than two options, the new data eliminates some choices and only keeps the most possible ones.

Cross Entropy Model:

Describe your Model: What approach did you take? What design choices did you make, and why? How did you represent the data? How can you evaluate your model for goodness of fit? Did you make an effort to identify and exclude irrelevant variables? How did you handle missing data?

We design a cross entropy model here as our second model. We consider the cross entropy is because we not only want our system to give us a number as the value we predict for the given feature as the output, but also want to get the probability to tell us how much likely our model

believes it should be such value. For example, if the feature Y we need to predict has 3 different categories: 1, 2 and 3. We want our system to compare firstly the probability Y=1, Y=2 and Y=3 given the data we input. If we know that Y=1 with the probability 0.2, Y=2 with the probability 0.3 and Y=3 with the probability 0.5, then our system will predict Y=3 here and tell us that the probability of Y=3 given the data we input is 0.5. Such output can be more convincing than just taking the value as the output, because if we get a low probability, for example, Y=1 with the probability 0.3, Y=2 with the probability 0.3 and Y=3 with the probability 0.4, our system will predict Y=3 as the result, but we will not trust this result that much just like we get that Y=3 with the probability 0.99. What's more, we add the one-norm regularizer to our lass function here to help us to drop out the irrelevant features. The reason why one-norm regularizer has such property has been explained in the part 1. Next, we will show our model specifically.

Loss function: $min_{\underline{w}} \ \sum_{i=1}^{m}[-\sum_{c=1}^{k} y_c^i \ln(f_c(\underline{x}^i))] + \sum_{i=1}^{k} \lambda \parallel \underline{w_i} \parallel_1$

Where: $f_i(\underline{x}) = \frac{e^{q_i(\underline{x})}}{\sum_{j=1}^{k} e^{q_j(\underline{x})}}$; $q_i(\underline{x}) = \underline{w_i} * \underline{x} + b_i$.

Note: Here m is the number of data, k is the number of categories of the feature we need to predict. $q_i(\underline{x})$ is the "probability" of the feature belongs to the i-th category and $f_i(\underline{x})$ is the softmax of $q_i(\underline{x})$ which is the "true probability" of the feature belongs to the i-th category. $\underline{w_i}$ is the weights we need to learn and we just let b_i as the last dimension of $\underline{w_i}$ in the loss function. $y_c^i = 1$ if y_i belongs to category c, else, $y_c^i = 0$.

The meaning of this loss function is to let the probability of the dataset being what it looks like now get the maximum value. And we need to find the $\underline{w}$ to make the loss function be minimum.

In our project, we choose to use the features with type "*effect*" as our features which need to be predicted or reconstructed from the features that are present and the features with type "*individual difference*" as our latent relevant features. We represent the input as a 46-dimension vector with each position is a real and positive number. And our output is a number and its probability to show how likely the feature is such value given our input. We need to train 35 different models for 35 different "*effect*" features here. What's more, since there are only few of blank of "*individual difference*" features, we can solve the missing data problem naturally, there are approximate 1400 records after we preprocess the data and we pick 10% of them as the testing data set, 10% of them as the validation dataset and 80% of them as our training data. We use the validation dataset to pick the best model trained from the training dataset and then we use the testing dataset to test our model. In our project, we use the test error to quantify how good the model is: if the data type is real value, we use the square mean error: $\text{test\_err} = \parallel \hat{y} - y \parallel^2$; If the data type is ordered categorial value, we use the accuracy: $\text{test\_err} = \frac{1}{n}\sum_{i=1}^{n} I(\hat{y}_i \neq y_i)$.

Describe your Training Algorithm: Given your model, how did you fit it to the data? What design choices did you make, and why? Were you able to train over all features? What kinds of computational tradeoffs did you face, and how did you settle them?

Firstly, we need to generate a matrix1 with the shape of m*(#features+1) (the last column is 1 which is designed for including bias into weights) ,i.e. m*47. Then we need a 47*(#category of Y) matrix2 to put the parameters. Then we do matrix1* matrix2 to get matrix Q to put $q_i(\underline{x})$, then we map it through softmax function to get matrix F to put $f_i(\underline{x})$. Then we need a matrix3 to put the target, which shows the category of y_i and it is used to get y_c^i. We use stochastic gradient descent to update the value of weights for optimizing the loss function: $w_{t+1} = w_t - \alpha\Delta w_t$. We give an initial value to weights and we need to calculate the gradient of weights every steps we update it.

Firstly, we consider the cross entropy part of the loss function:

$$L1 = -\sum_{c=1}^{k} y_c^i \ln\left(f_c(\underline{x}^i)\right)$$

$$\frac{\partial f_i(\underline{x})}{\partial \underline{w_i}} = \frac{\underline{x}e^{\underline{w_i}*\underline{x}}\left(\sum_{j=1}^{k} e^{\underline{w_j}*\underline{x}}\right) - e^{\underline{w_i}*\underline{x}}\underline{x}e^{\underline{w_i}*\underline{x}}}{\left(\sum_{j=1}^{k} e^{\underline{w_j}*\underline{x}}\right)^2} = \frac{\underline{x}e^{\underline{w_i}*\underline{x}}\left[\left(\sum_{j=1}^{k} e^{\underline{w_j}*\underline{x}}\right) - e^{\underline{w_i}*\underline{x}}\right]}{\left(\sum_{j=1}^{k} e^{\underline{w_j}*\underline{x}}\right)^2}$$

$$= \underline{x}f_i(\underline{x}) * (1 - f_i(\underline{x}))$$

$$\frac{\partial f_i(\underline{x})}{\partial \underline{w_j}} = \frac{0 - e^{\underline{w_i}*\underline{x}}\underline{x}e^{\underline{w_j}*\underline{x}}}{\left(\sum_{j=1}^{k} e^{\underline{w_j}*\underline{x}}\right)^2} = -\underline{x}f_i(\underline{x}) * f_j(\underline{x})$$

$$\frac{\partial L1}{\partial \underline{w_j}} = -\sum_{c=1}^{k} y_c^i \ln\left(f_c(\underline{x}^i)\right) = -\sum_{c=1}^{k} y_c^i \frac{\partial \ln\left(f_c(\underline{x})\right)}{\partial \underline{w_j}}$$

$$= -\sum_{c=1}^{k} y_c^i \frac{\partial \ln\left(f_c(\underline{x})\right)}{\partial f_c(\underline{x})} \frac{\partial f_c(\underline{x})}{\partial \underline{w_j}} = -\sum_{c=1}^{k} y_c^i \frac{1}{f_c(\underline{x})} \frac{\partial f_c(\underline{x})}{\partial \underline{w_j}}$$

$$= -y_j^i(\underline{x} * \left(1 - f_j(\underline{x})\right) + \sum_{c\neq i}^{k} y_c^i \underline{x} * f_j(\underline{x}) = f_j(\underline{x})\underline{x}\left(y_j^i + \sum_{c\neq j}^{k} y_c^i\right) - y_j^i\underline{x}$$

$$= \underline{x}(f_j(\underline{x}) - y_j^i)$$

The value of above equation can be got very easily from the data we have now.

Then, we consider the one-norm part of the loss function:

Since $\| \underline{w_i} \|_1 = \sum_j |{w_i}^j|$, so we just check the value of w_t if ${w_t}^j > 0$, $|{w_t}^j| = {w_t}^j$, else, $|{w_t}^j| = -{w_t}^j$, so the derivative is 1 if ${w_t}^j > 0$, else it will be -1.

So now, adding the two parts, we can get the gradient of weight easily every step. We can just give an initial value to weights and input one record each time to update the value of weights for optimizing the loss function by $w_{t+1} = w_t - \alpha \Delta w_t$.

For every model we get on training dataset, we run it on validation data and keep the one with the minimum validation error, then we run this model on the testing dataset to get our accuracy finally.

As the results, for every given "*effect*" feature, we have one unique model. We have write down 35 weights matrix to represent our model and for selecting the relevant features, we calculate the average of each weight and we can drop out those features whose weights are much less than the average weight. Our results can be checked in our zip-folder "weight" and here I put part of weights of the feature "bestgrade3" as one example (the columns are the relevant features and the last column is the bias and the rows are the weights correspond to each category):

	big5_01	big5_02	big5_03	big5_04	big5_05	big5_06	big5_07	big5_08	big5_09
class 1	0.039652735	5.3954078E-05	-0.037211895	0.055357598	0.11201878	-0.18217158	-0.035205144	-0.026789749	-0.021865703
class 2	-0.026321258	-0.17939538	0.046418216	0.07383074	0.03530177	0.03290638	0.080761164	-0.05108834	0.08834754
class 3	0.05161481	-0.02588438	0.053964455	0.06385788	0.022742629	-0.08714253	0.0319621	-0.015391442	0.10670081
class 4	-0.03769811	0.0063403477	0.14177944	-0.038676336	-0.07745773	0.0016828052	0.092783265	-0.04909634	-0.08804405
class 5	-0.007158026	-0.00066886	-0.058182165	-0.16764876	0.065056644	-0.04704678	0.025241414	-0.009296141	0.022204617
class 6	-0.018190073	-0.025328815	0.035283096	-0.006594456	-0.05036621	-0.11595549	-0.016761603	0.05649844	0.08408843
class 7	-0.056819405	0.11907671	0.063983575	-0.05121858	0.06025349	-0.027095465	-0.04415303	0.05515776	-0.090033464
class 8	-0.06377201	-0.06324502	0.04424834	0.049112983	-0.09121866	0.013714375	-0.08554805	0.14074187	0.08522162
class 9	0.0032238327	0.04238891	-0.0838492	-0.007339818	-0.014030564	0.047795653	0.050959215	0.089453526	-0.07422736
class 10	-0.07838742	0.031699263	0.06359963	0.0027607714	-0.015229292	-0.03168829	-0.031430252	0.07220533	0.10525891
Average	-0.019385492	-0.009496326	0.027003348	-0.0026557983	0.004707086	-0.039500095	0.006860907	0.02623949	0.021765133

……

pate_02	pate_03	pate_04	pate_05	selfesteem_01	stress_01	stress_02	stress_03	stress_04	Bias
0.044146657	-0.0037405214	-0.032271348	0.08696163	0.037934884	-0.14368378	-0.0062348754	0.0936232	0.09842746	-0.09089702
-0.090143725	0.07494039	-0.03626762	-0.11176487	0.13219772	-0.10213839	-0.08580334	0.056667056	0.12702842	0.10285405
0.11759253	0.059772223	-0.091507375	-0.08842308	-0.04017522	-0.050056376	0.08186577	0.03634958	0.18397495	-0.117907986
-0.07871207	-0.05000974	0.0035725383	-0.126596	0.06727152	0.04410046	0.073040396	-0.07052866	-0.16566302	-0.11369984
0.037921876	0.01914236	0.055774882	-0.035914488	-0.015460222	0.054291062	0.10741231	-0.09725693	0.04596647	0.07312911
9.4066745E-05	-0.07019048	0.02317902	0.033181388	0.08506217	-0.10889727	-0.077464476	0.0066133356	-0.05308377	-0.0039476175
0.035937868	-0.10901516	-0.010054457	9.1615046E-05	0.009736027	-0.10252245	-0.049347285	0.12266112	-0.1248511	0.0008347813
-0.11526959	-0.0016569857	0.08127094	0.004461481	0.09877562	0.06621768	0.11167104	0.048904486	-0.021181826	0.039013796
0.009667548	-0.12672246	0.04378746	0.044047747	0.098526835	-0.07392512	0.079276234	0.028923528	-0.12484722	-0.15133458
-0.09998713	0.120928146	-0.028664941	0.0638531	0.030910177	0.09923638	-0.14169627	-0.09442578	-0.09628804	-0.09799584
-0.013875196	-0.008655222	0.00088191126	-0.013010149	0.05047795	-0.031737775	0.0092719495	0.013153091	-0.013051769	-0.03599512

Actually, all the features with the ordered category value can be predicted by the same method, and even for the real value feature, we can still consider it as an ordered category feature, for example, if the value of a real value feature is 0-100, we can consider it have 100 categories and use the same method to predict it.

We use the same codes but different parameters to train different models. After getting 35 different models, we use these models to do the predictions. For filling out the blank of all the "*effect*" features, we just input the full dataset of the "*individual difference*" which has 1400 rows and get all the values of the "*effect*" features. Our results can be checked in our zip-folder "result"-document "predicted_data_all.csv" and here I put part of it as an example (the columns are the "*effect*" features and the rows are predicted values and the corresponding probabilities):

	kposition	kposition	lposition	lposition	nposition	nposition	rposition	rposition
	predicted_class	probability	predicted_class	probability	predicted_class	probability	predicted_class	probability
0	1	0.8108503222465515	2	0.7715947031974792	1	0.5325911641120911	1	0.8178765177726746
1	1	0.6824100017547607	2	0.8976356387138367	2	0.7379558682441711	1	0.8536559343338013
2	2	0.6786436438560486	2	0.5683311820030212	2	0.6628215312957764	1	0.8749818801879883
3	1	0.6315901279449463	2	0.8601129651069641	1	0.7723014950752258	1	0.7841419577598572
4	1	0.8738604784011841	2	0.8770291209220886	2	0.7619215250015259	1	0.7302187085151672
5	1	0.7981171011924744	2	0.7782427072525024	2	0.7331482768058777	1	0.8890008926391602
6	1	0.7714205384254456	2	0.6328662633895874	2	0.5994004011154175	1	0.6834749579429626
7	1	0.8060348033905029	2	0.6722903251647949	1	0.5746402144432068	1	0.9263318181037903
8	1	0.5269376635551453	2	0.549077033996582	2	0.6897966861724854	1	0.9427793025970459
9	2	0.5669162273406982	2	0.7803880572319031	1	0.7866184711456299	1	0.8607258796691895
10	1	0.8346427083015442	2	0.8462015986442566	2	0.690218448638916	1	0.5146244168281555
11	1	0.8865802884101868	2	0.7526243925094604	1	0.6863186359405518	1	0.5903964042663574
12	1	0.5485775470733643	2	0.733930230140686	1	0.5914600491523743	1	0.8506489992141724
13	2	0.852742075920105	2	0.6663267016410828	1	0.5680985450744629	1	0.8618625402450562
14	2	0.7559506297111511	2	0.5875723958015442	1	0.5263628363609314	1	0.8893190026283264
15	1	0.65817791223526	2	0.5200364589691162	2	0.536467432975769	1	0.6874875426292419
16	2	0.7492137551307678	2	0.7948413491249084	2	0.6273735761642456	1	0.789155900478363
17	2	0.7951325178146362	2	0.9555124044418335	1	0.5722659826278687	1	0.884945273399353
18	2	0.526511013507843	2	0.8745261430740356	2	0.5536641478538513	1	0.6391515731811523
19	2	0.9847338795661926	2	0.7804118990898132	2	0.7261204123497009	1	0.9295305609703064
20	1	0.6031985282897949	2	0.8135512471199036	2	0.5699849128723145	1	0.6596235632896423

Describe your Model Validation: How did you try to avoid overfitting the data? How did you handle the modest (in ML terms) size of the data set?

For avoiding overfitting the data, we add the regularizer to the loss function and also consider the behavior of our model on the validation and testing data as we have talked about above. For handle the modest size of the data set, while the raw data set provided is small, it can actually be blown up to quite a considerable data set. Given a complete record, we can create lots of training data by knocking out various subsets of features and trying to match this `knocked out' feature set with the original full feature set to get more data for training the models.

Evaluate your Model: Where is your model particularly successful, where does it lack? Does it need a certain amount of features in order to interpolate well? Are there some features it is really good at predicting and some it is really poor at predicting? Why do you think that is?

We calculate the accuracy, i.e., the test error for each model which can be checked in our zip-folder "result"-document "predicted_accuracy.csv" and here I put part of it as an example (the columns are the "*effect*" features and the rows are correponding accuracy):

	kposition	lposition	nposition	rposition	vposition	bestgrade3
accuracy	0.5185185185185185	0.5109489051094891	0.5328467153284672	0.5109489051094891	0.5036496350364964	0.2058823529411765
num of classes	2	2	2	2	2	10

We find that under the same cross entropy model (but different parameters) and the input, the accuracy of some features are pretty good, we can see that the accuracy of "mcmost5" is almost 99% as below:

mcfiller2	mcfiller3	mcmost1	mcmost2	mcmost3	mcmost4	mcmost5
0.6857142857142857	0.7518248175182481	0.8028169014084507	0.7746478873239436	0.8873239436619719	0.6338028169014085	0.9859154929577464
5	4	2	2	2	2	2

But the accuracy of some features are pretty bad, we can see that the accuracy of "elm_04" is almost 16% as below:

elm_04	elm_05	sarcasm	mcfiller1
0.15827338129496402	0.17985611510791366	0.28776978417266186	0.3525179856115108
9	9	6	4

What's more, we also calculate the proportion of each category of each "*effect*" feature in the original data set and compared it with the proportion of each category of each "*effect*" feature in our predicted data. The result can be checked in our zip-folder "result"-document

"statistic_data_all.csv" and here I put part of it as an example (the columns are the "*effect*" features and the rows are the proportion of the corresponding feature from the source dataset and the predicted dataset, the blanks are zeros or to represents that the feature has no such category):

	kposition	kposition	lposition	lposition	nposition	nposition	rposition
	source	predicted	source	predicted	source	predicted	source
1	56.78832116788322	48.42632331902718	53.47322720694645	4.649499284692418	42.63959390862944	38.698140200286126	47.10564399421129
2	43.21167883211679	51.57367668097281	46.526772793053546	95.35050071530759	57.360406091370564	61.301859799713874	52.89435600578871

We can see that our model do good on some features:

mcsome3	mcsome3	sarcasm	sarcasm	elm_05	elm_05	elm_01	elm_01
source	predicted	source	predicted	source	predicted	source	predicted
34.68795355587808	35.908440629470675	9.290780141843971	5.793991416309012	1.3532763532763532	1.2875536480686696	0.3561253561253561	0.35765379113018597
65.31204644412192	64.09155937052932	13.546099290780141	11.587982832618025	2.564102564102564	2.503576537911302	1.282051282051282	1.144492131616595
		15.319148936170212	12.231759656652361	7.264957264957266	13.662374821173104	4.202279202279202	1.859799713876967
		12.056737588652481	5.793991416309012	12.82051282051282	3.5765379113018603	9.401709401709402	25.393419170243202
		35.46099290780142	53.00429184549357	21.43874643874644	16.80972818311874	14.245014245014245	6.080114449213162
		14.326241134751774	11.587982832618025	23.575498575498578	13.23319027181688	20.299145299145298	21.244635193133046
				19.017094017094017	30.32904148783977	25.85470085470086	30.54363376251788
				6.98005698005698	11.087267525035765	13.603988603988604	11.301859799713878
				4.985754985754986	7.510729613733906	10.754985754985755	2.074391988555079

But our model do bad on some features:

bestgrade4	bestgrade4	vposition	vposition	lposition	lposition
source	predicted	source	predicted	source	predicted
20.203488372093023	3.5765379113018603	49.09616775126536	100	53.47322720694645	4.649499284692418
18.168604651162788	12.088698140200286	50.90383224873464		46.526772793053546	95.35050071530759
18.313953488372093	42.70386266094421				
26.01744186046512	30.972818311874107				
11.918604651162791	10.228898426323319				
4.941860465116279	0.4291845493562232				
0.436046511627907					

The reasons of this behavior may be that:

1, The parameters can be tuned better for getting a good result;

2, The data we have is too small for getting a good result;

3, Since the category of these features are bigger, for example, "elm_04" has 9 categories, we need more complex model to train such feature;

4, Such features are more irrelevant to the input features that we selected so we can't get the good model with these features. Actually, there are some features we think hard to predict because the questions of these answers are independent to any other questions.

5, Since usually people do not like to do such questionnaires, they may just give the answers without even thinking about it, so the data we obtained may not have a true distribution to predict at all.

Analyze the Data: What features were particularly valuable in predicting/interpolating? What features weren't particularly useful at all? Were there any features about the researcher/experimental environment that were particularly relevant to predicting answers - and if so, what conclusions can you draw about the replicability of those effects?

Actually, we have picked out the features that are more valuable in predicting in data preprocessing. As we have talked about, some features are not necessary and possible to be predicted, for example, predicting the name, ID and the time to fill out questionnaires of users are meaningless and inefficient, so we only consider to do the data completion and interpolation of the features in the variable codebook which are more important for analysis and more possible to predict with the information we have. And in our results, we can see that the feature with less categories may get better accuracy than those have many categories. And some answers are actually come from the same question, so we can predict such features better with the information we have than those almost independent to other features.
In our model, we try to use the "*individual difference*" to predict "*effect*" features, so we just find that how people consider themselves may influence how they consider their surroundings. And we think the researcher's behaviors and the experimental environment will definitely influence the psychological characteristics of the respondents. For example, if the respondents are tested in a spacious and bright environment by a friendly researcher, they may give more positive answers. Under such case, for decreasing the negative noise caused by the surroundings, we can consider the replicability of those effects in other experiments.

Generate Data: Use your system to try to generate realistic data, and compare your generated data to the real data. How good does it look? What does it mean for it to 'look good'?

Firstly, we calculate the proportion of each category of each "*individual difference*" feature in the original data set and use these statistics to generate a dataset with 500 rows which can be checked in our zip-folder "data"-document "new_features.csv". Then we use this dataset and our models to predict the values of the remain "*effect*" features, i.e., we generate Y according to P(Y|X), the result can be checked in our zip-folder "result"-document "predicted_simulate_data.csv".

	kposition	kposition	lposition	lposition	nposition	nposition	rposition	rposition
	predicted_class	probability	predicted_class	probability	predicted_class	probability	predicted_class	probability
0	1	0.6670247912406921	2	0.8255488276481628	1	0.7428255677223206	1	0.9149802923202515
1	1	0.5100256204605103	2	0.848031222820282	2	0.691831648349762	1	0.9035362601280212
2	2	0.8155136108398438	2	0.6950541138648987	2	0.7073256969451904	1	0.8159722089767456
3	2	0.5537821650505066	2	0.9228917360305786	1	0.5964411497116089	1	0.5379949808120728
4	1	0.651670515537262	2	0.761235773563385	1	0.7901426553726196	1	0.9015927910804749
5	2	0.9262356758117676	2	0.5883728265762329	2	0.6304897665977478	1	0.7615376114845276
6	1	0.8656064867973328	2	0.6612139344215393	2	0.5513502955436707	1	0.8257940411567688
7	2	0.8384344577789307	2	0.81841641664505	2	0.5649401545524597	1	0.7249894142150879
8	2	0.668674111366272	2	0.851194441318512	1	0.5333775281906128	1	0.7310436367988586
9	2	0.6498396396636963	2	0.9079025387763977	1	0.5683093070983887	1	0.5795122981071472
10	2	0.8879927396774292	2	0.5162975192070007	2	0.7813905477523804	1	0.900888204574585
11	1	0.9542505741119385	2	0.7394999861717224	2	0.6818077564239502	1	0.9173287153244019
12	1	0.5345555543899536	2	0.7915383577346802	1	0.567439615726471	1	0.7720364332199097
13	2	0.6083289384841919	2	0.922155499458313	2	0.5308924317359924	1	0.9001678228378296
14	1	0.9213956594467163	2	0.866810142993927	2	0.6026454567909241	1	0.7960014343261719
15	2	0.8341971635818481	2	0.8593428730964661	2	0.5072439908981323	1	0.8956641554832458
16	2	0.7252914905548096	2	0.7450329661369324	1	0.5237290859222412	1	0.776282787322998
17	2	0.8316800594329834	2	0.9655025005340576	1	0.6489747166633606	1	0.5795525908470154
18	2	0.5427748560905457	2	0.8505455851554871	1	0.6348791718482971	1	0.8325870037078857
19	2	0.5681858658790588	2	0.9425067901611328	1	0.7488163709640503	1	0.9265558123588562
20	1	0.6826180219650269	2	0.9648531675338745	1	0.5765531063079834	1	0.7184996604919434
21	1	0.9122455716133118	2	0.7992504239082336	2	0.5415118932723999	1	0.6835739612579346
22	1	0.8000768423080444	2	0.5500382781028748	2	0.6838840246200562	1	0.8556732535362244
23	1	0.8157827854156494	2	0.7269489169120789	1	0.5676677823066711	1	0.7264890074729919

Then we calculate the proportion of each category of each "*effect*" feature in the original data set and compared it with the proportion of each category of each "*effect*" feature in our predicted data. The result can be checked in our zip-folder "result"-document "predicted_simulate_data.csv" and here I put part of it as an example (the columns are the "*effect*" features and the rows are the proportion of the corresponding feature from the source dataset and the predicted dataset, the blanks are zeros or to represents that the feature has no such category):

	kposition	kposition	lposition	lposition	nposition	nposition
	source	predicted	source	predicted	source	predicted
1	56.78832116788322	50	53.47322720694645	7.6	42.63959390862944	40.8
2	43.21167883211679	50	46.526772793053546	92.4	57.360406091370564	59.199999999999996
3						
4						
5						
6						
7						
8						
9						
10						

We can see that our model do good on some features:

	kposition	kposition	nposition	nposition	mcsome3	mcsome3
	source	predicted	source	predicted	source	predicted
1	56.78832116788322	50	42.63959390862944	40.8	34.68795355587808	32.4
2	43.21167883211679	50	57.360406091370564	59.199999999999996	65.31204644412192	67.60000000000001
3						
4						
5						
6						
7						
8						
9						
10						

But our model do bad on some features:

lposition	lposition	vposition	vposition	elm_01	elm_01
source	predicted	source	predicted	source	predicted
53.47322720694645	7.6	49.09616775126536	100	0.3561253561253561	0.2
46.526772793053546	92.4	50.90383224873464		1.282051282051282	1.4000000000000001
				4.202279202279202	3.2
				9.401709401709402	22
				14.245014245014245	4.8
				20.299145299145298	20.4
				25.85470085470086	29.2
				13.603988603988604	15.6
				10.754985754985755	3.2

We can consider that if the results look good, it may show that our models for predicting these data are good. Since we believe there exist some distributions of these features in the natural world and our model capture these distributions well. But we can see that some of our models do not work well and the reasons may be the same with we have talked about above in the “Evaluate your Model” part.

Compare two models and analysis:

Comparing the results of the decision tree model with the cross entropy model, we can find that:
Firstly, from the perspective of models’ accuracy, in general, the cross entropy model performs better than the decision tree model because it has smaller testing errors in most cases. The reason may be that the cross entropy model has stronger ability of telling relevant features from

irrelevant ones under the ways we train the models. For example, in the decision tree model, we just prune the tree according to the size of the leaf nodes which can be tuned greatly and our parameters may not be the best one to build the tree. What's more, in the cross entropy model, we design a loss function which can describe the probability of each feature's category and try to maximize the sum of each probability which makes more sense for finding the relevant features from all the input features we give to the model.

Then, from the perspective of the proportion of the new data we generate from our models, it's hard to say which model produces data that are closer to the original data since we can see that both of the two models can do well on some features but bad on some other features and the two models perform different on different features. Theoretically, the decision tree model has the ability to fit any function well through the data, but there is the overfitting problem, and the prune can avoid it somehow, but we will face the loss of accuracy too if we can not choose the best parameter. However, we design a very reasonable loss function here as our cross entropy model and get better results than decision tree model here, which somehow tells us that we must choose and design different and suitable algorithm for different problems in our life. All in all, we think that the reasons why the model perform bad on some features are that: the data we have for train is not enough and has too much noise; the parameters we tune in each model may not be the best one since the time is limited and there are too many models to train. On some degree, the features with smaller categories will have smaller testing error and follow the original distribution better.

Bonus

Bonus 1: Build your system to include processing and analysis of natural language answers, to try to use them to inform prediction of other features. How can you represent natural language answers in a useful way? What language processing is necessary to be able to use them? Do they actually provide useful information for the prediction problem?

We can represent the natural language answers in vectors. And there are many ways we can get the embedding of words, for example, a naïve way is that, for a given feature with the natural language answers, we can first find all the words in our dataset and put them into a words bag. For example, if in our dataset, we have 5 words: I; love; machine; learning; class. Then we put all these 5 words into a words bag. And we encode each record into a 5-dimision vector. For the first record, if its answer has "I", then we will let the first place be 1 in our vector, if its answer does not have "love", then we will let the second place be 0 in our vector, etc. then we can get the vector of every record. We can also use neural network to get the embedding of the words, for example, we can consider the auto encoder to do that. Anyway, for processing and analysis of natural language answers, to try to use them to inform prediction of other features, we must firstly get their embeddings. After that, we can just add the vector to our input vector X to train the model or we can choose to do not use the embedding to do the training, but calculate some similarity of each pair of the embeddings and use these numbers as the input which depends on the unique questions and models and need to be tested to see the effect.

Bonus 2: Build your system to include prediction about the natural language answers. Based on available features, what can you predict about a person's answers to natural language questions (if not the actual answers themselves)? How can you assess the quality of these predictions?

Based on available features, I think we can predict the emotion of the answers to natural language questions. For example, we can pick some words which have positive emotion, for example, happy, agree, etc; some words which have negative emotion, for example, unhappy, disagree, etc; some words which have neutral emotion, for example, indifferent, etc. And we can label such bag of words as numbers, for example:1,2,3. Then we can rewrite the dataset by using the numbers to replace the natural language answers and use the new dataset to do the predictions. We can use the same way as we talked about to train the model and get the emotion of the answers after training. And we can also use the test error, i.e., the accuracy to assess the quality of the predictions.

Bonus 3: Build your system to include generation of natural language answers to questions, based on available features. How can you model this generation problem, and how can you evaluate the quality of the answers?

To provide machine generated answers, we design a sequence to sequence model to achieve this goal. The overview of the model is given in the following figure.

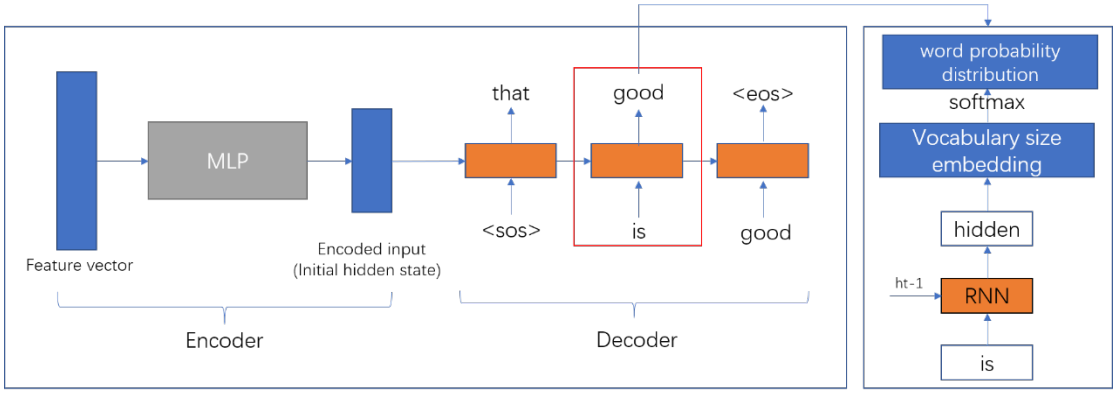


Figure 1 Overview of the answer generation model

Encoder: We use the features in the given dataset to assemble the input feature vector. Then we feed the feature vector into a multi-layer perceptron and encode it into a lower dimension latent representation factor, which is served as the initial hidden state of the decoder.

Decoder: Except for the feature vector from given data, we need to build a set of word vectors for each of the words in the vocabulary ('vocabulary' means the dictionary which contains all the words in the entire dataset). These word vectors can be randomly initialized and learned during the process. You can also use some pretrained word embeddings such as GloVe. In our design,

we randomly initialize the word embeddings (vector) and the embeddings are learnt during the whole training process.

The decoder is a recurrent neural network (RNN) which takes encoded input feature vector as initial hidden state and word embedding at each time as input. Take the right part of the figure as an example. When we want to predict the next word “good” by given current word “is” and the hidden state from previous step, the RNN module would generate a hidden vector as the output. This output hidden vector would be fed into a linear layer to map to a vocabulary size vector (here vocabulary size is the total number of words in the dataset. For example, if we have totally 1000 different words, the vocabulary size is 1000 and we assign a unique index to each of these 1000 words). Then the vocabulary size vector would be processed by softmax to generate the words probability distribution. Each dimension is representing the probability of the corresponding word under this index in the vocabulary for the next predicted word. For training process, we minimize the cross entropy loss to make sure the model would generate the expected words. During testing process, we can simply select the word with the maximum probability as the current output word. Although this is easier to implement, the generated sequence may not be the optimal one. To make the model more robust, we can do beam search on the generated sequence to select the best sequence. In this way, the selected sequence (or say the generated answers) has a better chance to get an optimal solution.

Summary: this is a sequence-to-sequence model which contains an MLP as encoder and RNN as decoder. The encoder takes original feature data as input and produce an encoded hidden state for decoder use. The decoder takes the initial hidden state from encoder and current word vector input to produce a word probability distribution. Then we can minimize the cross entropy loss to train the model. To generate the sequence by given the trained model, we can do beam search to obtain the sub-optimal solution.