CS536 Project

Bharat Manikanta (nmv54), Sandeep Panigrahy(sp1673), Shubham Sinha (ss3076)

May 2019

1 Data Representation

For this project, we go ahead with the dataset provided by " The Original Many Labs Project" in the file "Tab.delimited.Cleaned.data set.WITH.variable.labels.csv". The given file is a tab delimited data set. To represent diverse data in this file, we first start with reading and storing the file in dataframe object using following steps:

1. Parse the file "CleanedDataset.sav" using pyreadstat package to read its data and metadeta information.
2. The data part is then read and stored in pandas dataframe. This is saved as a separate 'csv' file, which is then directly used for data access.

After successfully loading dataset in pandas dataframe, we remove unwanted and noise values from our dataset using following steps:

1. Use a replace function to find few regex values in dataframe and assign them 'empty'.
2. While searching for number of missing values, we found that there are few cells, which are null but are not detected by pandas library as empty cells.

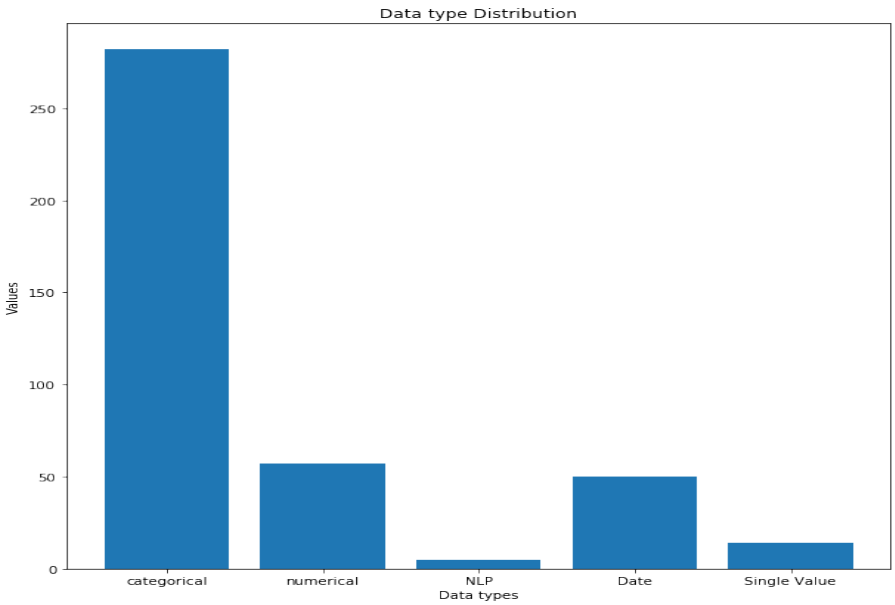


Figure 1: Different data types of data

From the above bar graph we can see that our dataset has more than 250 columns of categorical data and almost 50 columns has numerical data. There are quiet a few NLP columns and nearly 50 columns have values in format of date. The last bar represents the number of columns that have only a single value for the entire column which are approximately 20.

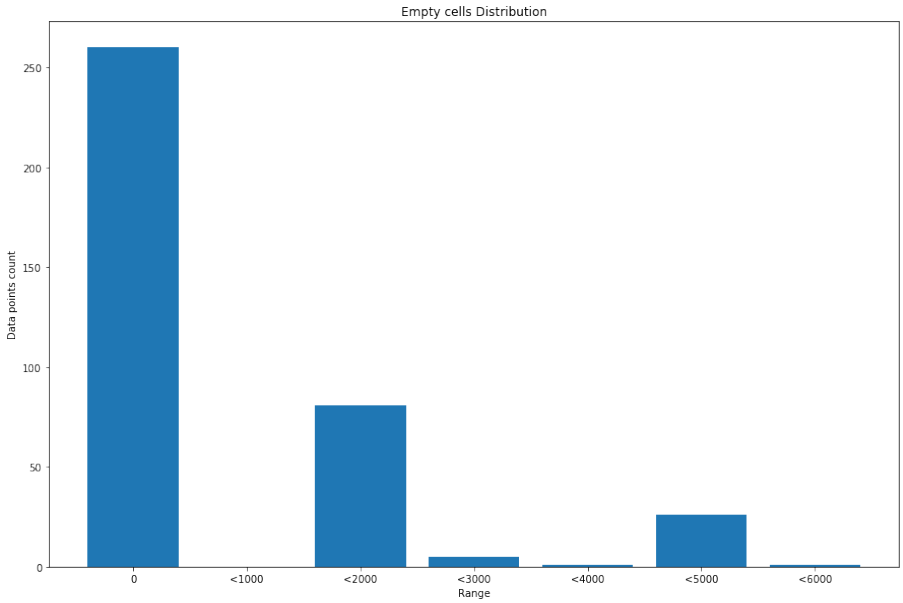


Figure 2: Empty cells distribution

The above bar graph shows that we have almost 250 columns that don't have missing values. Almost 80 columns have 2000 missing values. Nearly 35 columns have 5000 missing values. 10 columns fall in each of the categories <1000, <4000 and <6000.

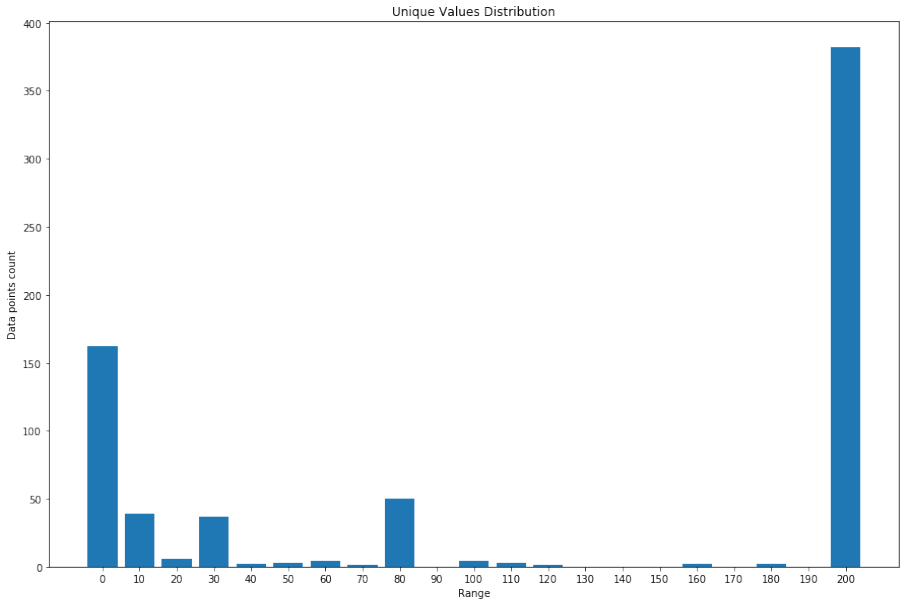


Figure 3: Unique values in data

The above bar graph gives us an idea about the unique values for each column in our dataset. Here X-axis represents the number of unique values and Y-axis represents the number of columns.

We have cleaned dataset, which can be used for representation and processing for learning algorithms.

2 Data Processing

In cleaned dataset, we find the datatypes of all features and separate them into multiple dataframe objects based on their datatype. In our dataset, we separated features based on 4 different datatypes : categorical, numerical, datetime and natural language responses. We standardize and represent each dataframe file separately:

1. Categorical data: For every feature, we represent categorical data as unique integers. For this, we first find all unique values of features and assign them different integer values. This information is stored in dictionary. Then we replace all categorical data as values from the label dictionary.

2. Numerical Data: We represent numerical data as standardized values. To standardize these values, we use following formula for every column:

$$x = (x - x_{min})/(x_{max} - x_{min})$$

3. DateTime data: To represent datetime values in our dataset, we first convert all values to a single date format. We parse all values as string and using datetime function, we convert these values to the format "dd–mm–yyyy-hh:mm:ss". Then, these datetime values are converted as type integer, and are then standardized in the similar way of numerical data. We have assumed that the earliest date of a column is minimum and the most recent date value is maximum.

4. Natural Language Responses: To represent natural language responses, we use few representation methods. For the feature "feedback", the responses were in multiple languages. We translated all responses in different languages to a common language - "English".

After all data processing, we combine the dataframes to a single dataframe object which can then be trained using our learning models.

3 Problem of interpolation

At first, we separated the columns into 2 sets. One set contains all the columns with full values (no need to predict these columns further) and the other set with columns that have missing values. Mathematically, in simple terms, interpolation means to find the unknown values using the known values. Generally, we use an algorithm trained on available values and use it to predict/generate the outputs for unknown values. All the columns in the full values set are used as input all the time. Additionally, along with these, we use the other available values of every row to predict the null values. These inputs are passed to imputation function to output the predicted values (in place of missing values). In our case the number of missing values for each row varies and we use the available values of each row to guess the unknown or null values in that particular row (which is interpolation in our case).

4 Model Selection

1. DECISION TREES:
   At first, we thought of going with decision trees as they are easy to build and simple to code. In our dataset there are many columns/features with almost 90 percent of values missing. Here the problem with decision trees is that they need to learn that feature using the other 10 percent of available values. Now as they can't learn the feature properly, they make a lot of errors while predicting the missing values. Also, we thought of building multiple decision trees for a subset of features. Even now these columns with most values as missing is a hurdle for good accuracy of the model. As accuracy is the key for us, we decided to drop this idea.

2. SMOTE:
   Next, we thought to create synthetic data points to increase the dataset and came to know an algorithm called SMOTE. SMOTE stands for 'Synthetic Minority Over-sampling Technique'. It is a popular over sampling technique. We can create artificial data points from our available datapoints using this approach. The best part is that these artificial data points are not the duplicates of the available data points. These are created by the following approach:

   - Represent the data points in their respective space.
   - Find the line joining any 2 different points (the difference of the points).
   - Multiply that value with a random number between 0 and 1 to get the artificial point.

   We can create as many data points as we want using this approach and easily explode our dataset. But, to do this we need some datapoints which have complete features i.e. the data points where there is no need to predict any feature. In our dataset we don't have any data point of this kind. So, we dropped this approach.

3. MICE:
   Next, we thought to come up with an approach that has the power to predict missing values with out duplicating the dataset and used an approach called MICE. MICE stands for 'Multiple Imputation by Chained Equations'. It has the ability to predict the missing features after learning the available features. The approach is as follows:

   - Collect all the features that are completely available (has no missing values).
   - Feed them as the input to the algorithm.
   - Arrange the columns that are to be predicted in the order of increasing number of missing values.
   - Predict one feature (the one that comes at the beginning of the ordered columns) by using the features that are with no missing values as input to the algorithm.
   - Append this feature to the input features and recursively predict all the features until convergence.

   This could have been a successful approach for our complete data. But, the problem is that we were unable to find out proper reasoning to use both numerical and categorical data in this model. So, we have implemented MICE as an imputation approach for numerical values in our dataset.

4. AUTOENCODERS:
   Next, we thought of implementing Autoencoders. These are neural nets that aims to mimic their inputs. They compress the input into a latent-space and then reconstruct this into an output. This consists of two parts:

   - ENCODER: This is the part of network that converts input into latent space.
   - DECODER: This is the part which converts latent space into output.

   Thus, the autoencoder tries to produce the output that is very similar to the input.

   This is not just the copying of input as output. But the main part lies in producing some useful latent space representation. This is just like dimensionality reduction. Autoencoders can learn data projections that are more interesting than PCA or any other basic techniques.

   These autoencoders learn a distribution in that latent space of the data. Thus, it can generate samples that are very similar to our dataset and inputs. Suppose that our input is X and is passed to encoder to obtain E(X). Now decoder takes E(X) as input and generates D(E(X)) or Y as output. The autoencoder tries to generate Y as closely as possible to input X. In other words, it produces an output that minimizes the squared loss (of X and Y). This squared loss is called as reconstruction error in the context of autoencoders. There are some further advanced versions like β-VAE, regularized autoencoders, denoising autoencoders etc. They give best results but are hard to code. So, we settled with autoencoders.

5 Training Algorithm

5.1 Categorical Data

1. 21 columns test:
   We were in a situation where we need to make sure whether autoencoders work for our problem or not. Then we did this 21 columns test to give a try for autoencoders. This is how we did: There are almost 200 full columns in our dataset. We took 20 columns randomly out of these and the 21st column is a numerical column. Then we used these 20 columns to predict the missing values of 21st column. This sanity kind of test provided good results for us. So, we decided to go with autoencoders.

   Then we tried this approach for the entire dataset. Surprisingly it didn't give better results when we used autoencoders for the prediction of missing values in the entire datasets. But we didn't want to drop the idea of autoencoders. So, we tried splitting dataset into 2 separate datasets: one containing categorical data and other containing numerical data. We used these subsets of dataset to solve our problem. We trained our autoencoder

to predict missing values in numerical dataset and it gave bad results. Our only option is to try for categorical data. We tried and it worked.

2. Our Autoencoder:
   It consists of 3 layers that are fully connected. We used the following activation functions for the layers: For layer one we used RELU. This is the one that worked better for our problem. RELU is the function that selects MAX (0, X). Simple but effective. For the next layer Sigmoid suited better. The optimizer we used is Adam optimizer. This gave us better results than gradient descent. We coded a custom Adam optimizer function and used it for training purpose.

3. Training:
   We took a batch size of 256 rows and ran the algorithm for 100 iterations. We masked X% (X is 10, 20 etc) of known data. Masking in the sense hiding the known data from the algorithm and make it use in future. This is for the purpose of evaluating our model. Our epoch count is also 100. Here the training error is nothing but the error that we got for imputation.

4. Imputation:
   For categorical data we imputed the Nan values by the known values. Here we need to decide which value to use among the available values of the column to impute. For this we pivoted the columns based on the distinct available values and passed it to the function to get the weights of the pivoted columns and used the column with heavy weight to impute the Nan values in the column. This provided good results for imputed data.

5.2 Numerical Data

To start with we have 57 columns. We have taken the column that has no missing value (previous session id) and assigned it to the variable Z. We arranged the remaining 56 columns in the decreasing order of their number of missing values.

1. Now for column in i = 2 to 57, we assign a column to variable Y. We assign Z to X. Using X, Y we fit our model on observed Y values and predict the missing Y values. We now add this column to Z and repeat the process until we get a completely filled Z.

2. For column in i = 2 to 57, we pick a column and assign it to Y and assign all the remaining columns to X. Using X, Y we fit the model on observed Y and predict the missing values of Y.

3. We performed the step b for 10 times (we found this as optimal).

4. We repeated the steps a, b, c for 3 times and generated 3 sets of imputed data.

6 Feature Selection

The main challenge we faced while exploring our dataset, was to how to select features for imputation and training our model. We tried to find similarities between questions, responses and features. Though there were similarities between certain features like:

1. All features with task id creation time and task id creation dates were same.
2. There was a direct relation between the features involving multiple columns of "allowed forbidden Group", "flag Group", "gender", etc

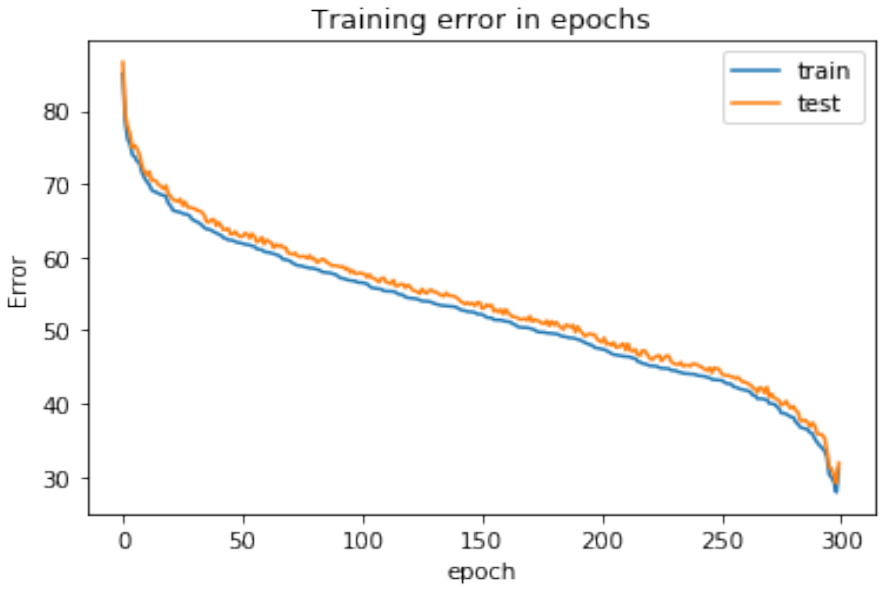
there was no conclusive similarity for a large set of feature which can have a noticeable impact on our data imputation. Hence, the most reasonable feature selection we thought, was that "all features contribute to the prediction of all features". To learn about which feature contributes how much to the prediction of certain columns, we use dataset as input to our Auto Encoder model.

7 Model Validation

7.1 Categorical Data

Our training algorithm for categorical data depends on number of iterations and interdependence of features in our dataset. We know from given data that there is not a concrete way to determine if features are strongly correlated. To ensure that our training algorithm learns to a certain point to predict properly and not overfit as well. To counter this challenge, we iterate our model to a certain number of iterations with certain batch size. These parameters are determined by repeated experimentation using different pair values. Initially we select a random 10 % of training data as validation data. For every iteration, we train our model on training data and then determining mean absolute error of our model for validation dataset. We iterate our model till error continues to decrease and does not increase as well. We summarize our training and validation error for each epoch in the following graph. Here we observe that though the initial training error was closer to 80% it comes down till 30% at around 296 epochs, and then it increases again. As soon as the training error ceases to decrease, the model returns the final weights, which is then used for imputation.

Figure 4: Figure of training error for epochs



The above graph shows us the relation between the number of epochs and error for both training and test data. We can see that the test error is always a little bit more than training error through out the graph. Here the main thing is that the error decreases with increase in number of epochs. The error is very bad when the number of epochs are nearly zero and continued to decrease with the increase in number of epochs. For 300 epochs we have the least amount of error in the graph.

7.2 Numerical Data

As MICE uses decision trees to model the data we can say that MICE is very prone to overfitting the data. Because decision trees easily overfit on the given data if they are grown to a more depth. So, to avoid overfitting in the model we need to do some kind of regularization to our model. As MICE uses decision trees, we decided to use the pruning technique on the decision trees in our model. Here we decided to prune the decision trees to a certain depth before they start overfitting.
We fix the depth of the decision trees by fitting and validating the decision trees on the observed values in the initial seeding part of our algorithm. In the initial part of the algorithm we fix Z as the column that has all observed values. Then we take the first column that has the least number of missing values and fit a decision tree on 80% of the observed values and validate on the rest 20% and plot a graph of the depth of the decision tree vs validation error. After that we fit a unconstrained decision tree on all the observed data and predict the

missing values, and get a completed column. We add this completed column to Z and do the same for the column that has 2nd least number of missing values, and then for the 3rd and so on. We get the plot and see that once depth increases over approximately 10, the validation error starts increasing which is an indication of overfitting. Hence, we keep a maximum depth of 10 for each of the decision trees in all of the steps that we use.

We plotted the results of depth vs RMSE values for 5 different columns and figured out that a depth of 10 is better to avoid overfitting in our model. The plots below show the results for our experiment.

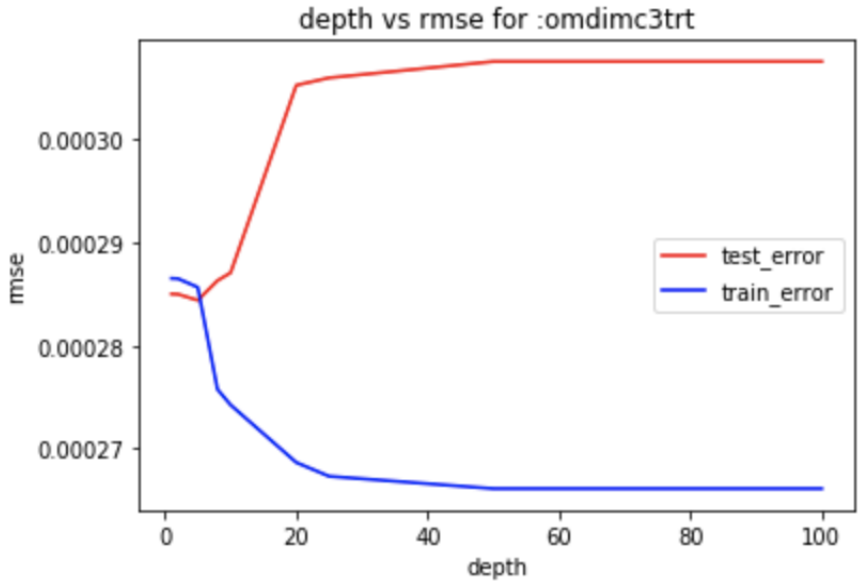


Figure 5: Dept vs rmse - 1

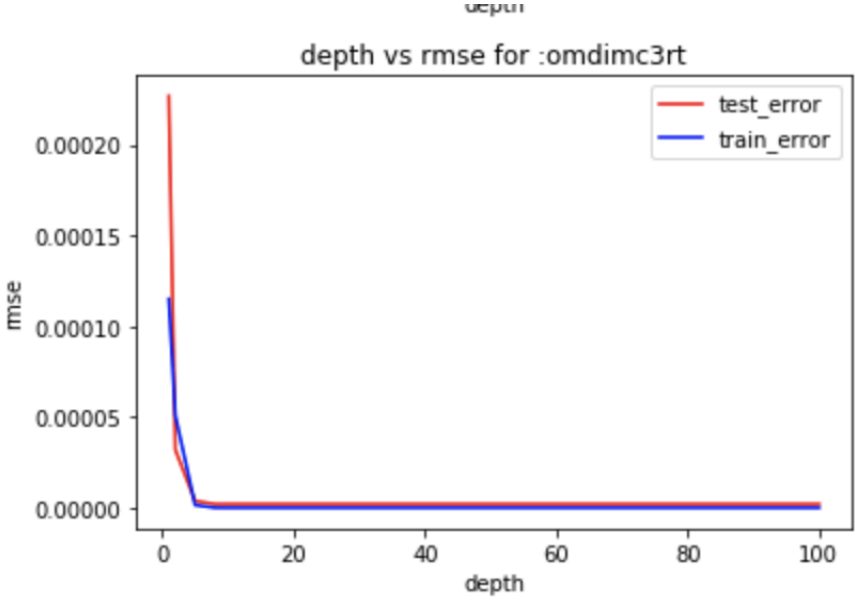


Figure 6: Dept vs rmse - 2

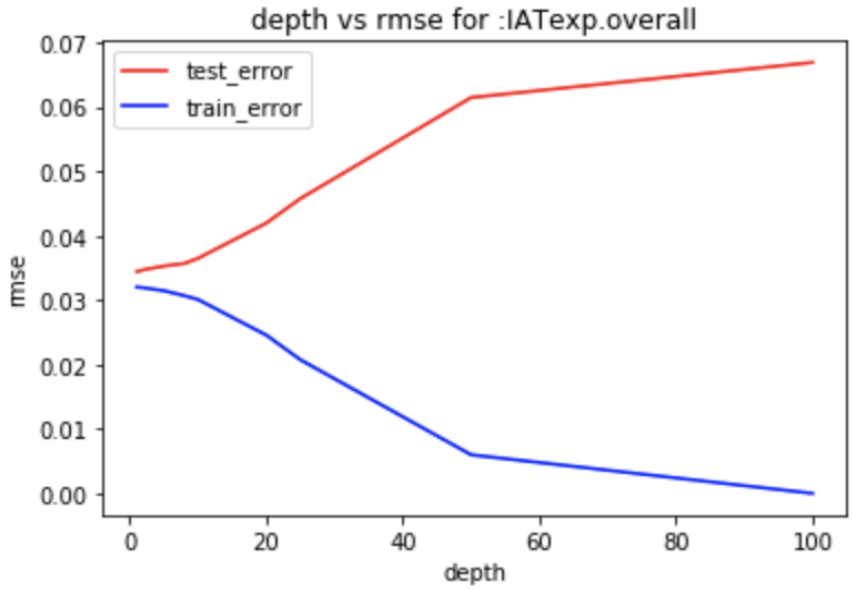


Figure 7: Dept vs rmse - 3

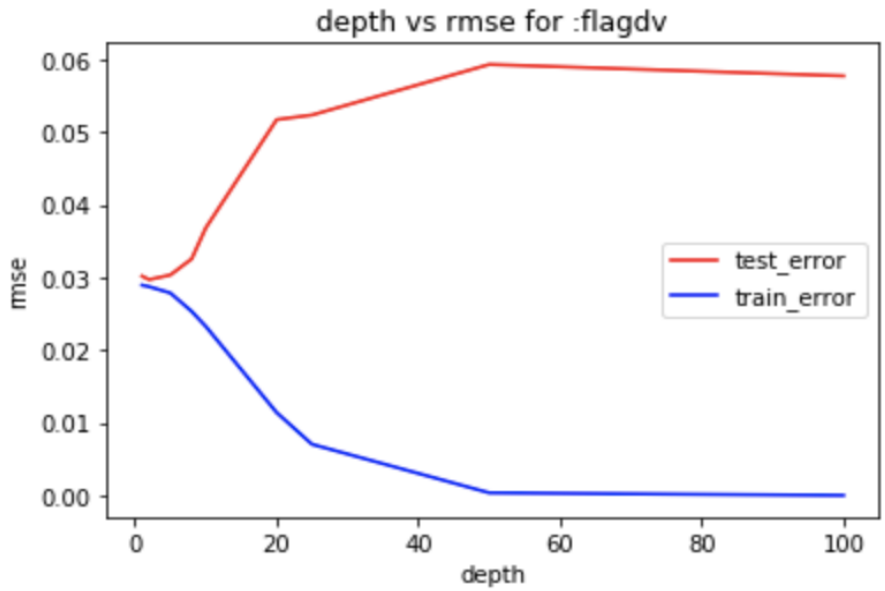


Figure 8: Dept vs rmse - 4

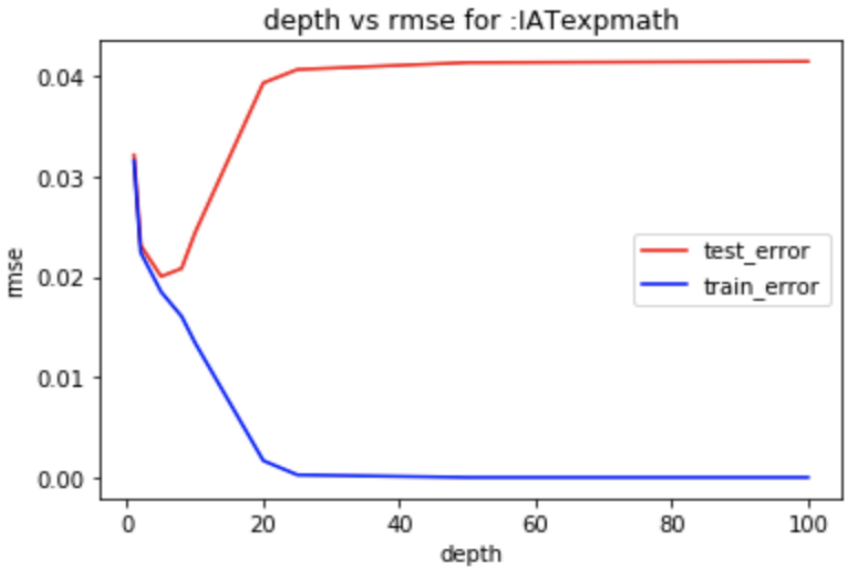


Figure 9: Dept vs rmse -5

8 Model Evaluation

8.1 Categorical Data

Accuracy Score:

Here the accuracy score in the context of classification is the ration of total number of correct predictions to the total number of input samples. This is how we can understand it in an easy way: Given 2 sets, say, y-true and y-pred we can perform the accuracy score calculation in the following way:

1. For every corresponding element in the above 2 sets, compare them and count the total number of matches.
2. Divide the value in step 1 with the total number of samples.

This is how we do in our case:

1. From the imputed dataset randomly select 20% of values for each column such that these have corresponding values in the original dataset i.e these values are not missing in the original dataset.
2. Compare them and count the total number of matches.
3. Divide the value in step 2 with the total number of samples considered (20% of dataset).

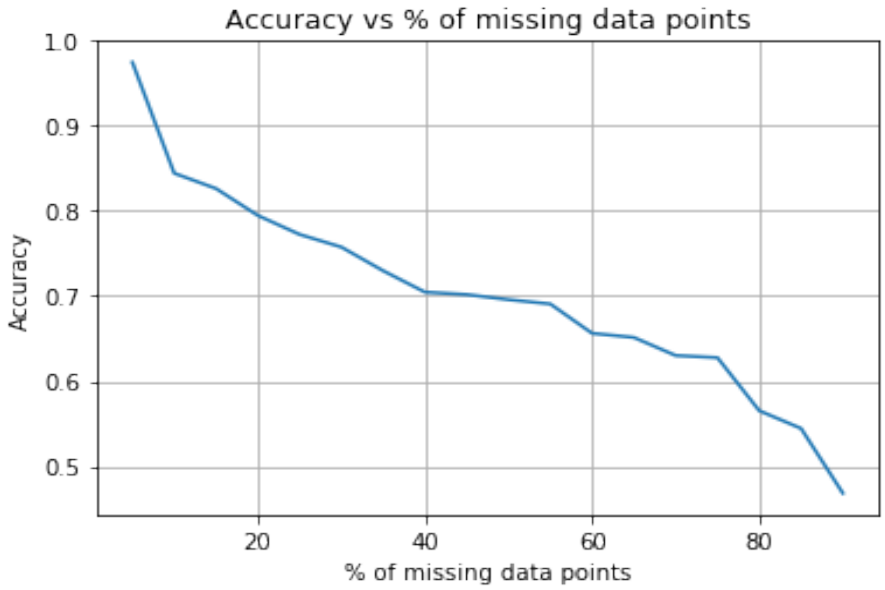


Figure 10: Accuracy vs missing data points

The graph above shows us the relationship between the accuracy and % of missing data points. We can infer from it that the accuracy is high when there are less missing points. The accuracy continued to decrease with the increase in % of missing data points. The accuracy is almost 1 when we have very less missing data points and it fell to almost 0.4 when we have more than 90% of missing data points.

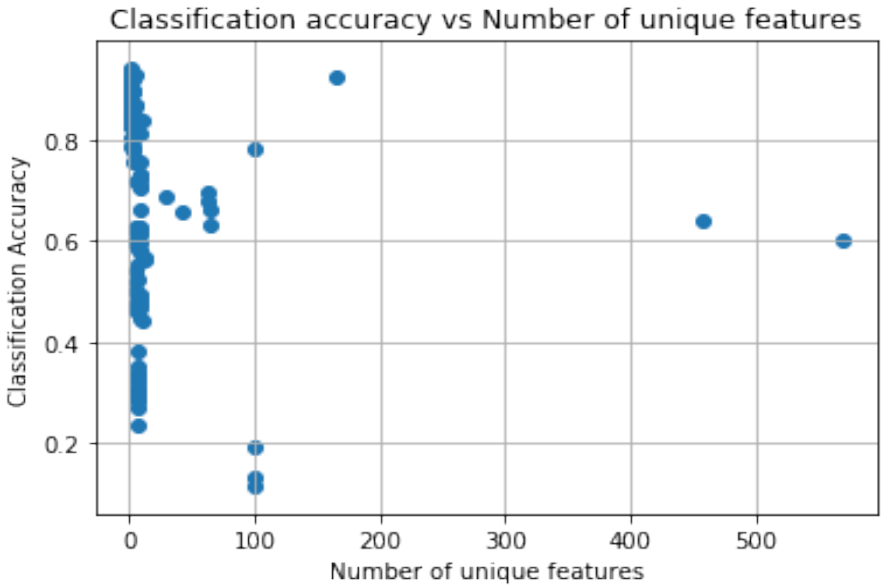


Figure 11: Accuracy vs unique data points

The graph above is the plot for classification accuracy(Y-axis) against number of unique features(X-axis). We can say that our model performed well even when there are almost 500 unique features. But, the accuracy is a bit poor in the case of columns with 100 unique features. We can see that accuracy is more than 0.6 for most of the cases.

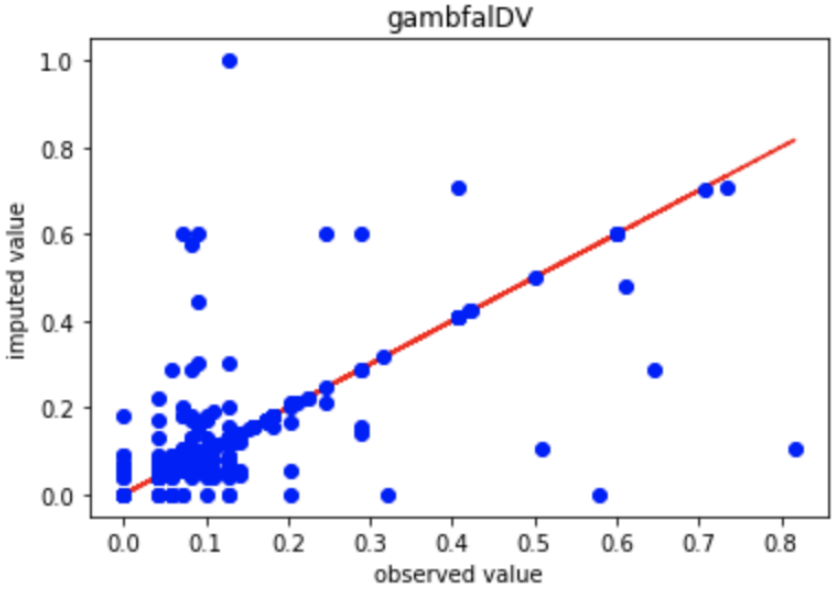
8.2 Numerical Data

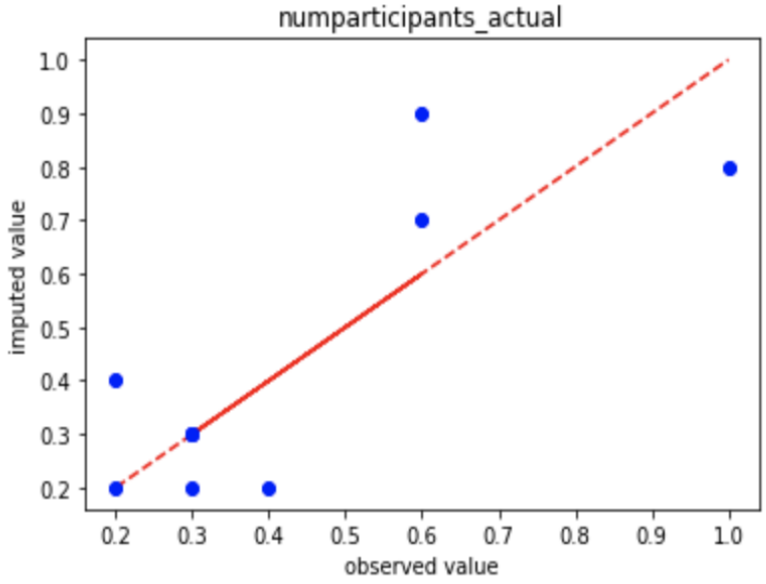
1. Statistical comparison:
   We tabulated the summary statistics of the observed data and the imputed data. Here we compared the means, standard deviations, minimum and maximum values for the observed data with the imputed data. We got pretty good results with only 3 imputed sets of data. These kinds of checks provide a method to compare the distribution of observed data with the imputed data which is good. This gives us a way to tell whether the imputed data is reasonable or not. Even though the distributions are

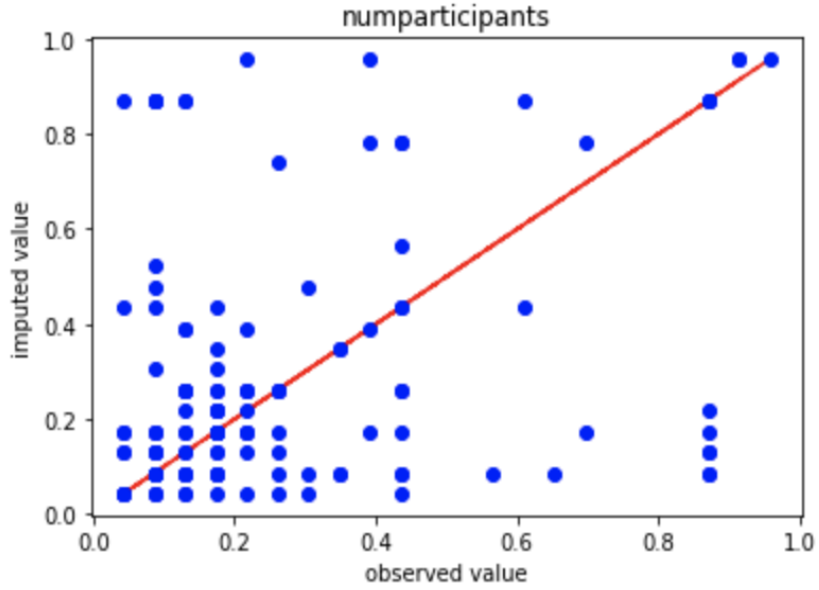
almost similar we can have some values in imputed data that fall outside the range of possible values of a particular variable. This is the drawback of this approach. At sometimes these may not have a bigger impact, but we need to consider different approaches for the sensitive data.

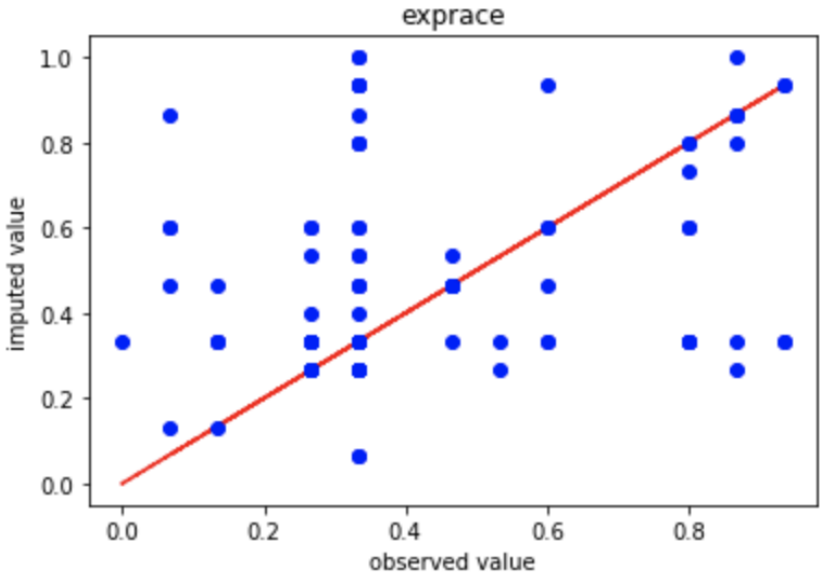
2. Cross validation:
   We can even use cross validation to check the performance of our imputed model. Here we take 5% of the observed data (from our original dataset) for each column. We used only 5% of the observed data for this process as our dataset is too big for this process and we need to perform more iterations on the values. We then remove one value from the selected 5% of values. We fit our model on the remaining data (which doesn't have deleted value). We then try to predict the removed value using our model. This process is repeated for each observation, by deleting that observation, fitting the model on the remaining dataset and using the model to predict the deleted value. This is called leave-one-out-cross-validation. We do this for every numerical column in our dataset (57 columns). The predictive performance of the model can be observed graphically by plotting the observed values against the imputed values.
   In each column of the original dataset, since we are standardizing the column, there are redundant data. When we predict each of the 5% of the observed values in each of the columns, we get multiple predicted values for the same standardized observed value. This can be seen in the graphs, where on the X axis we have the observed values and on the Y axis we have the imputed/predicted values, that for the same X value we have multiple Y values. We see that in most of the columns, when the standardized observed value is in the lower ranges, the predicted values are closer to the actual values. This indicates that our model is not so successful in predicting the higher ranges of the observed values i.e. 0.6 or higher.

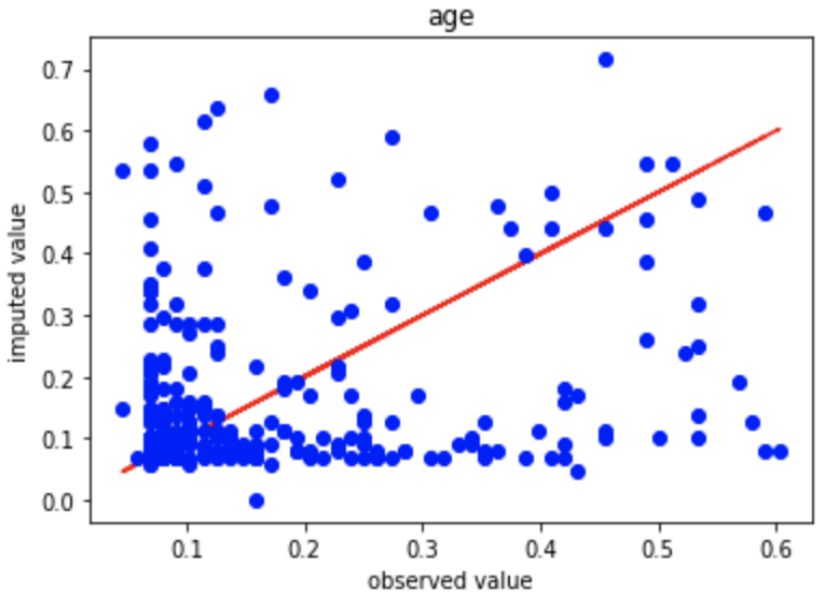
   Below are the results of cross validation.


gambfalDV
imputed value
observed value
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


numparticipants_actual
imputed value
observed value
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

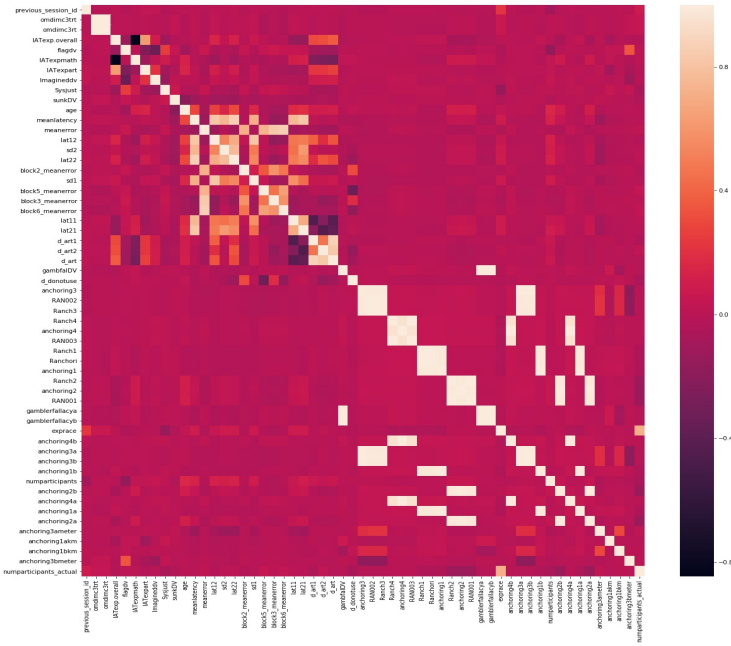

numparticipants
imputed value
observed value
0.0
0.2
0.4
0.6
0.8
1.0


exprace
imputed value
observed value
0.0
0.2
0.4
0.6
0.8
1.0



9 Feature Importance

Here we used correlation matrix to determine feature importance.Correlation is used to measure the strength and the direction of the linear relationship between two variables. We need to notice that correlation is a function of covariance. The reason for which we used correalation is that it has standardised values whereas covariance values are not standardised ones. correalation has values in the range of -1 and 1. It is obtained by dividing the covariance of two variables by the product of their standard deviations. If the value is closer to +1 it means that the variables are strongly related to each other and have positive correlation i.e. if one value (variable) increases then the other value will also increase. If the value is closer to -1 it means that the variables are strongly related to each other and have negative correlation i.e. if one value (variable) increases then the other value will decrease. Now, with this understanding of the concept we can view our correlation matrix. We can see that our correlation matrix has values in the range -1 and +1. All the diagonal values are +1. This is because the correlation of a variable with itself is always +1. Now, coming to off diagonal values we can see that almost all the values have red/maroon/pink colour which indicates weak correlation or nearly no correlation. We can also observe that some of the values are black or very dark, for example d-art1 and lat1, which indicates strong negative correlation. Also, some of the variables have strong positive correlation, ex: gambfalDV and gamblerfallacya. These values have very light color or almost white/wheatish color.



10 Data Generation

For generating new data after imputing the missing values we use Gaussian Mixture Models. GMM can be used to build a generative probabilistic model describing the distribution of the data. Since we have both categorical data and numerical data we need two separate GMM's. However, since the dimensionality of both the numerical and categorical data is high, GMM will have problems in converging in such a high dimensional space. So, we have to use an invertible dimensionality algorithm. We use PCA for this purpose. Once we apply PCA to the numerical data the dimension reduces from 57 to 27., with almost no information loss. We determine the number of Gaussian components in the GMM by plotting the error vs number of components. After determining the number of components, we generate new data by first drawing samples from the 27-dimensional space. After drawing a sample from the GMM we perform inverse transform of the PCA object to get new data. Since we are using GMM, and GMM models the new data based in the underlying distribution of the given data, the statistical measures like mean and variance of the new data will be the same as the given data. Though this is not a good measure to estimate the 'goodness' of the generated data, this gives us a guarantee that the generated data won't be too far away from the given data. One approach we used to

correct over-fitting is adjusting the model likelihoods using Akaike information criterion (AIC). Using our imputed data and multiple GMM models we get, we use AIC to estimate the quality of each model, relative to each of the other models. This helps us to select the most appropriate model for our data.

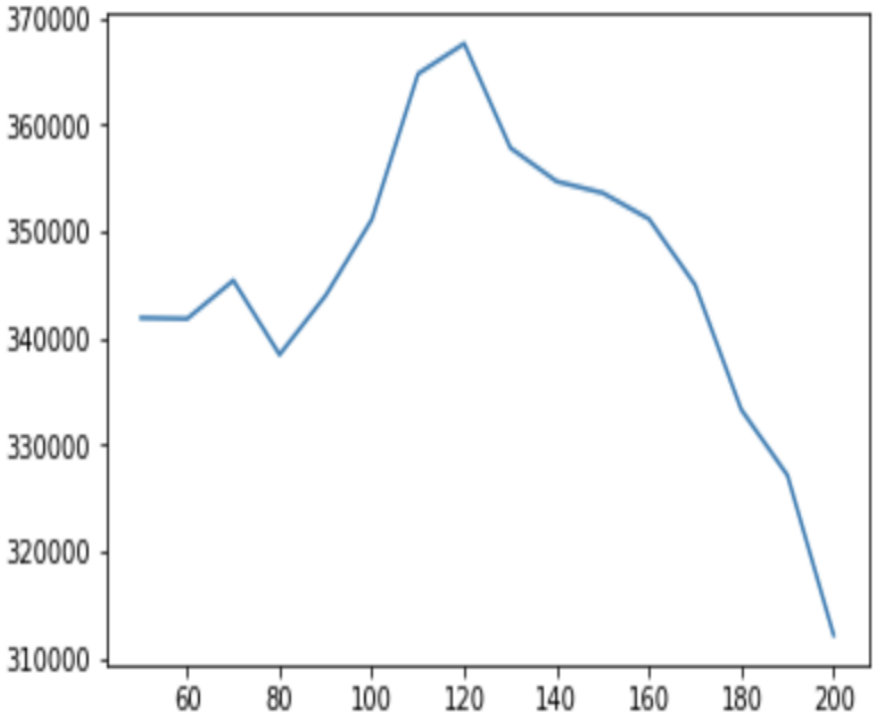


Figure 12

11 Bonus 1: Natural Language Answers Analysis

The data had natural language answers in the column 'imaginedescribe' where subjects write what they imagined for a minute. The first way we tried incorporating natural language answers was to detect the language in which the answer was in and use it as a feature. The language which a person wrote the answer can say what part of the world he/she is from and this might in turn affect various societal factors that influence a lot of the answers he gave in the survey. However, the data was very noisy and not all the answers could be correctly processed, and these cases were considered as missing data. We used a package called 'langdetect' which is a port for Google's language detection library. After

detecting the language, we consider it as categorical data and do the general pre-processing, analysis as before.

The other approach we thought of for using the natural language answers was to use clustering based on the similarity of the answers. For this purpose, for the meantime we consider only the samples that don't have a missing answer in the 'imaginedescribe' column. Given all the answers and the pretrained glove, we find the p dimensional embedded word vectors for all the words in each of the answers. Now given the embedded vectors, for each answer we find the document vector. To calculate the document vector from the embedded word vectors for each answer we consider using a bag of words approach, where we weight all the word vectors in the vocabulary with the count of the word in each answer and sum them to get the document vector for that answer. After we get the document vectors, we consider calculating the similarity using cosine similarity. Using the cosine similarity as a distance metric we cluster all the subjects. The reasoning behind this is that given a minute to think about, if two people are thinking about the same thing, it indicates a similarity in their personality traits and this again influences the answers given in the survey. Now that we have clusters if subjects we can fit our imputation models on these clusters separately rather than on the complete data set. As the subjects are of the same personality or thinking the answers they give are supposed to be closer to each other.

12 Bonus 2/3: Natural Language Answer generation and Evaluation

For predicting the answers in the 'imaginedescribe' column we take help of the clusters we just created. After imputing the rest of the features for the samples in a cluster using our previous model we try to predict the missing values in the 'imaginedescribe' column. We have a n-dimensional cluster center for each of the clusters, which doesn't include the answers to the column 'imaginesdescribe'. Now for the samples that don't have the answers, we have to generate them. But we have to first impute the other missing values. We do this by matching each sample with a missing answer to cluster center based on the features that are present. For the observed features we find the cluster center that has the closest features as the observed features. Once we get the closest cluster we impute the missing features as the corresponding features of the closest cluster. For the missing answer as well, we assign the same answer as the answer of the sample that is closest in the cluster.

Another approach to generating the natural language answers would be to use VAEs. We could use separate VAEs for each of the clusters, which would be fitted on the answers in that cluster. The input to the VAE would be a vector representation of all the features while the output would be the natural

language answer. For the samples with the missing answers we find the closest cluster center and impute the missing features as before using the feature from the closest cluster center. Now to generate the answer we input the vector representation of the features, observed and imputed, to finally get the natural language answer.