

CS 536 Machine learning

Final Project Report

Yujia Fan - yf198
Samanth Reddy Karra - sk2051

1 Introduction

The ML1 and ML3 data sets are records of personal answers and reports from subjects on a variety of questions relating to topics like perception and mental biases. Additionally, the data sets include demographic information about the subjects and information about the researchers and experimental environments. All are potentially useful for the problem of prediction and interpolation.

We have chosen the ManyLabs1 dataset as the dataset size is comparatively higher than ManyLabs3 dataset and also the features are more in ManyLabs1 dataset compared to ManyLabs3 dataset. So, it might be good in predicting the missing values based on many features. The ML1 dataset contains 6344 rows and 382 columns without any preprocessing.

The dataset contains various types of columns of diverse data types real values, integer values, categorical or binary values, ordered categorical values, open/natural language responses.

Main Task: Given a new or partial record, we should predict or infer the values of missing features (any missing feature!) based on the features that are present.

2 Preprocessing and Representing the data

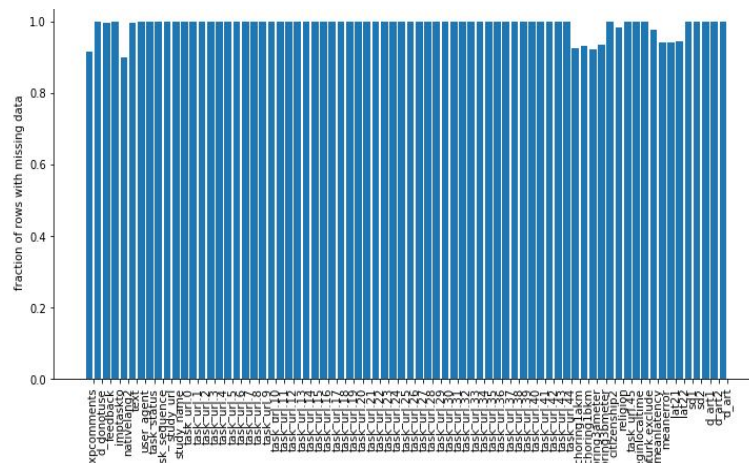
ML_1 dataset has 6344 rows and 382 columns. There are columns of diverse data types real values, integer values, categorical or binary values, ordered categorical values, open/natural language responses. we have inferred many things from the data and preprocessed the data so that the data becomes compatible to train a machine learning model. Following are the some of the inferences from the data and the steps we took to preprocess the data :

- There are many empty columns with no values and some as simply dots(.) which are converted to NaNs
- The last_update_date and session_last_update_date are same. So, dropped last_update_date column
- Dropped the duplicate rows in the dataset
- There are 57 rows which are full(100 percent) nan values , so no point in using them for predicting other missing values. Also , we don't have the filled column

values to train the column ,So dropped them off.

54	tuple	2	('task_url.26', 6344)
55	tuple	2	('task_url.37', 6344)
56	tuple	2	('task_url.4', 6344)
57	tuple	2	('task_url.35', 6344)
58	tuple	2	('d_art1', 6343)

- Percentage of null columns when plotted on graph looked like :



- Some of the columns(67 columns) have constant values. So dropped them off.For example the following columns have constant values:

Index	Type	Size	
0	str	1	task_url.35
1	str	1	task_url.4
2	str	1	task_url.37
3	str	1	task_url.26
4	str	1	task_url.30

- For some of the columns it contains both text and numbers , we checked the codebook and the corresponding number of the text and encoded the text to corresponding number.

For example:

Strongly disagree encoded to 1 for sysjust columns. Moderately Happy encoded to -2 and Slightly Happy to -1 in iatexplicit column. Very Unafraid encoded to -3 and Slightly Unafraid to -1 in iatexplicitmath6 column.

You can see the columns looks like the following before preprocessing:

iatexplicitmath5	iatexplicitmath6
4	6
5	5
Slightly Avoid	Moderately Afraid
Very Avoid	4
Moderately Avoid	6
5	6
Moderately Avoid	Moderately Afraid
5	5

- There are many date columns in the dataset, So converted the datetime column to weekday column(whether it is on weekend or not).This was done whether to check the feedback may depend on the day of the week like on weekend it is easy to keep more time to give feedback.

For example:

session_date	weekend
2013-08-09 00:00:00	0
2013-08-09 00:00:00	0
2013-08-10 00:00:00	1
2013-08-10 00:00:00	1

- The columns such as userid, session id (basically ids) doesn't contribute to prediction since all are unique , so dropped them off.
- All the categorical columns are label encoded(thought of doing one hot encoding but some of the columns have more categories which resulted in large number of columns and training it is taking a lot of time)
- All the continuous columns are converted to decimal point precision up to three digits.(This is done to make calculations easy to understand)
- Categorical columns are more in the dataset compared to continuous and text columns.
- There are two type of natural language columns. one is the normal answers like feedback which is sentence and the other is just different categories like Morning , Evening, Night etc. Some of them are texts with 2 or 3 unique

sentences in the column. For the fixed number of unique words or sentences in column , we label encoded them. For the real feedback or hand written sentences, we just kept as it is and later convert to vectors (explained clearly in bonus answer).

- we observed that there are no rows in the data which have zero nan value

```
train_without_null_row.shape
(0, 382)
```

- Some of the columns provide similar information , so we reduced the number of columns by looking into the column and eliminating columns whose information can be inferred from other column. For example: In the following example mlpoland1 of studyname and mlpoland1 of studyurl are same and studyurl doesn't contain more information which distinguishes other values in the column, In other words, any one of the two columns are sufficient to infer the output. Similarly for mlpolandonline1

study_url	study_name
mlpoland1/mlpoland1.expt.xml	
/user/kratliff/uflrick/manylibs/mlpoland1/mlpoland1.expt.xml	ratliffiab.uflrick.mlpoland1
/user/kratliff/uflrick/manylibs/mlpolandonline1/mlpolandonline1.e...	ratliffiab.uflrick.mlpolandonline1

- There are few more preprocessing steps done.Finally after all the preprocessing, we have divided the dataset into 3 categories

1.Categorical data

2.Continuous data

3.Text data

Categorical data has 43 columns with no nan values and 109 columns with at least one nan values.

Continuous data has 0 columns with no nan values and 21 columns with at least one nan values.

In text columns,There are two types as we mentinoed before. For the fixed number of texts in the column, we label encoded them to categorical values , so it goes into the categorical data and for the text consisting of sentences (like feedback) , we noticed that there is one column 'imagineddescribe' which can be useful in predicting other columns.

For better understanding:

Categorical data looked like after preprocessing:

cat_data_with_null_data - DataFrame		
reciprocityus	allowedforbidden	quotearec
0	1	2
0	1	5
0	1	5
0	1	nan
0	0	nan
1	0	nan

Continuous data looked like after preprocessing:

cont_data - DataFrame		
age	anchoring1	anchoring2
18	nan	900000
19	nan	nan
18	5000	400000
18	3000	300000
18	1750	3500000
18	nan	300000

Text(Sentence) data looked like before preprocessing (detailed description of converting to vectors in bonus part):

imagineddescribe
nan
nan
the beach, sand between my toes, the sound of waves crashing, sea...
nan
Camping grounds in West Texas, sunset with deep orange red yell...
nan

3 Model Description

We have tried mainly 3 models to predict the missing values:

1. Base_Model(variants of it)
 - i)1st_variant(predict with full known columns)
 - ii)2nd_variant(predict with full known columns and append)
 - iii)3rd_variant(impute first and predict)
2. KNN Based imputation
3. Multivariate imputation by chained equations(MICE)
 - i)column_by_column imputaion
 - ii)value_by_value imputaion
 - iii)value_by_value followed by column_by_column
 - iv)column_by_column followed by value_by_value
4. AutoEncoder

i)Base_model(variants of it):

1st_variant in Base_model:

First we have tried predicting the NaN columns using the fully filled columns.we have predicted the categorical as well as continuous columns based on the fully filled columns. In our case, there are 43 columns which are fully filled categorical columns and 0 fully filled continuous columns. we used the categorical columns to predict the unknown values of both categorical and continuous columns. We have used k-fold cross validation scheme with k=3 to check how our model is performing.we used k-fold cross validation as we want to get models to train on diverse datasets . Finally we averaged the results of individual folds.

2nd_variant in Base_model:

Similar to the 1st variant, we predicted the NaN values in columns with known columns, but here we appended the results to the training dataset again before predicting the other columns. For suppose, we used 43 columns to predict the 44th column (with nan) and the predicted column is appended to the train dataset and we predicted the 45th column and so on, we have done this iteratively till all the columns are filled.Here the choice of the column to predict matters where as in previous case , it is independent of the sequence of selection of columns. So, here we have chosen the columns which has less number of nan values to predict first and predicted the columns with nan values with decreasing number of number of nan values.We have chosen this way of selecting columns first because we get to have more training dataset since it has less nan values compared to the other columns. we found out that this methods gave better results that the 1st variant.

3rd_variant in Base_model:

Here first we filled all the nan columns with mode for categorical columns and mean for numerical columns except for the column we need to predict and we predicted the nan columns with known columns, we have done this iteratively to get all the columns filled. But the main drawback of this method is we get the nan values similar to the mean or mode values and also our assumption of filling the nan with mean or mode might give bad results if there are outliers in the data

ii)K-Nearest Neighbour Approach:

we later implemented KNN Approach for predicting missing values. For each subject with missing values, we found the k nearest neighbors using a Euclidean metric, confined to the columns for which that subject is not missing. Each candidate neighbor might be missing some of the features used to calculate the distance. In this case we averaged the distance from the non-missing coordinates. Having found the k nearest neighbors for a subject, we imputed the missing elements by averaging those (non-missing) elements of its neighbors. This can fail if all the neighbors are missing in a particular element. In this case we use the overall column mean for that block of subjects. K-nearest neighbour can predict both discrete attributes (the most frequent value among the k nearest neighbours) and continuous attributes (the mean among the k nearest neighbours). Additionally, There is no necessity for creating a predictive model for each attribute with missing data. Actually, the K-nearest neighbour does not create explicit models. Thus, the K-nearest neighbour can be easily adapted to work with any attribute as class, by just modifying which attributes will be considered in the distance metric. Also, this approach can easily treat examples with multiple missing values.

iii)Multivariate imputation by chained equations (MICE):

We later implemented Multivariate imputation by chained equations (MICE) which is a multiple imputation technique. This method is similar to the 3rd variant of Base model but here we did it for a fixed number of iterations. MICE operates under the assumption that given the variables used in the imputation procedure, the missing data are Missing At Random (MAR), which means that the probability that a value is missing depends only on observed values and not on unobserved values. MICE approaches have been used in datasets with thousands of observations and hundreds of variables. In the MICE procedure a series of models are run whereby each variable with missing data is modeled conditional upon the other variables in the data. This means that each variable can be modeled according to its distribution. The model (chained equation process) we implemented can be broken down into four general steps:

Step 1: A simple imputation, such as imputing the mean, is performed for every missing value in the dataset. These mean or mode imputations can be thought of as "place holders".

Step 2: The "place holder" mean or mode imputations for one variable ("var") are set back to missing.

Step 3: The observed values from the variable "var" in Step 2 are regressed on the other variables in the imputation model, which may or may not consist of all of the variables in the dataset. In other words, "var" is the dependent variable in the model and all the other variables are independent variables in the model. These models operate under the same assumptions that one would make when performing models outside of the context of imputing missing data.

Step 4: The missing values for "var" are then replaced with predictions from the regression model. When "var" is subsequently used as an independent variable in the regression models for other variables, both the observed and these imputed values will be used.

Step 5: Steps 2–4 are then repeated for each variable that has missing data. The cycling through each of the variables constitutes one iteration or cycle. At the end of one cycle all of the missing values have been replaced with predictions from re-

gressions that reflect the relationships observed in the data.

Step 6: Steps 2 through 4 are repeated for a number of cycles, with the imputations being updated at each cycle. The number of cycles to be performed can be specified by us. At the end of these cycles the final imputations are retained, resulting in one imputed dataset.

we have created few more variants of mice based on above algorithm. One is column by column imputation, second value by value where we will be predicting each value as in leave one out fashion. Third is, in the first iteration, we will be predicting column by column and in the subsequent iterations, we will be predicting value by value. Fourth is, in the first iteration, we will be predicting value by value and in the subsequent iterations, we will be predicting column by column.

why we choose this model:

Creating multiple imputations, as opposed to single imputations, accounts for the statistical uncertainty in the imputations. In addition, the chained equations approach is very flexible and can handle variables of varying types (e.g., continuous or binary) as well as complexities such as bounds or survey skip patterns.

iv) AutoEncoder:

Later we implemented AutoEncoder Approach. Autoencoders are a special type of neural network architectures in which the output is similar as the input. Autoencoders are trained in an unsupervised manner in order to learn the extremely low level representations of the input data. These low level features are then deformed back to project the actual data. An autoencoder is a regression task where the network is asked to predict its input. These networks has a tight bottleneck of a few neurons in the middle, forcing them to create effective representations that compress the input into a low-dimensional code that can be used by the decoder to reproduce the original input.

Internally, it has a hidden layer h that describes a code used to represent the input. The network may be viewed as consisting of two parts: an encoder function $h=f(x)$ and a decoder that produces a reconstruction $r=g(h)$. The learning process is described simply as minimizing a loss function $L(x, g(f(x)))$, where L is a loss function penalizing $g(f(x))$ for being dissimilar from x , such as the mean squared error.

In practice, the procedure has the following steps:

Step 1: Replace missing values in data with random values.

Step 2: Input the data to the trained Autoencoder:

- a. Sample from the latent variable distribution (the output of the encoder) to generate Z , given X ;
- b. Sample from the reconstructed data distribution (the output of the decoder) to generate X , given Z .

Step 3: Replace the missing values with the reconstructed values, leaving the observed values unchanged.

Step 4: Compute the reconstruction error of the observed values.

Step 5: If the reconstruction error is below a specified tolerance, or the iteration limit has been reached, end. Otherwise, return to step 2.

Denoising AutoEncoder for imputing missing values:

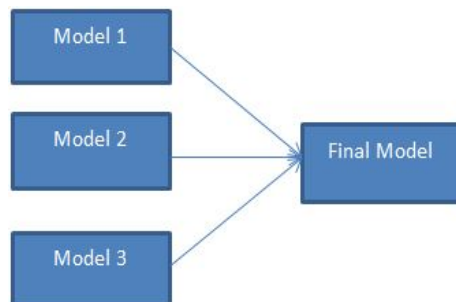
Denoising Autoencoder (DAE) is a variant of Autoencoder. DAE is a deep architecture that is designed to recover noisy data which can exist due to data corruption

via missing data. The idea of DAE is to train the Autoencoder to reconstruct the input from a corrupted version of it in order to force the hidden layer to discover more robust features and prevent it from simply learning the identity. The random corruption process randomly sets some of the inputs to zero. Hence the denoising Autoencoder is trying to predict the missing values from the non-missing values with randomly selected subsets of missing patterns. Then, Gibbs sampling idea is applied. Being able to predict any subset of variables from the rest is a sufficient condition for completely capturing the joint distribution between a set of variables. To convert the Autoencoder class into a Denoising Autoencoder class, we add a random corruption step operating on the input. Additionally, we have built our own sigmoid function, cross entropy metric for evaluating the model.

For each of the 4 main models, we took categorical data to predict categorical missing values and continuous data to predict continuous missing values.(in the base model 1st and 2nd variant, we used categorical data to predict continuous missing values since there are no column without nan)

Ensemble Model :

Model ensembling is a supervised learning technique for combining multiple weak learners/ models to produce a strong learner. Ensemble model works better, when we ensemble models with low correlation. Model ensembling helps in reducing generalization error by reducing variance in machine learning tasks. we tried ensembling the model results. when we tried ensembling all the model results, we didn't get good accuracy. So, we tried ensembling the results of top 3 models and it gave good validation results. So, in our final ensemble model we took the average of top 3 models for regression and for classification, we took the majority vote of 3 classifiers.



Predicting the model goodness:

we have used three methods for predicting the model goodness

1. There are 43 features which no nan values. so randomly put nans in this features and predict these features. we can again quantify it as training and validation loss as we take a subset of rows as training data and rest as testing data.
2. Fill up the nans with 0s and find the imputed matrix and check the error between imputed and original matrix.
3. There are 43 features which no nan values. we can take indexes of rows for which we know the values for a column and split it into train and validation set and predict the known values of column and find the loss.

Identifying and Excluding Irrelevant variables and Handling missing data:

To tackle the missing data, we have tried few approaches like imputing the value with mode of the corresponding column, keeping 0s etc. The best way is chosen with the help of validation results. Here the mode gave better results. For the algorithms like mice where in we can use an algorithm to predict the unknown values, the algorithm automatically gives lower weight to the features that are irrelevant like decision tree for classification and ridge regression for continuous features. Additionally we have checked which features are not relevant in prediction with the help of graphs.

4 Training Algorithm

Once we choose our model, we have used Decision Trees Classifier, Random Forest Classifier for Categorical data prediction and Decision Tree Regressor, Ridge Regression, Lasso Regression for Continuous data prediction.

Avoiding Overfitting:

while using Decision Tree Classifier or Regressor, we have limited the tree depth and minimum number of samples needed to split on each node to avoid overfitting on to the data. For Ridge Regression, we have controlled the regularization parameter alpha to control overfitting onto the data.

Some of the Computational difficulties we faced while training the model and our solution:

1. we first one hot encoded all categorical variables, it became nearly 5000 feature vector which took very long time to train on our scratch models. So, we label encoded categorical variables instead of one hot encoding it.
2. During Running the MICE algorithm for a number of iterations, it is taking a lot, so let the iteration number to 3.
3. Also the decision tree algorithm is taking lot of time if we don't fix the maximum depth and minimum number of samples to split at each node, so we added them as parameters to the algorithm.
4. KNN approach is taking a lot of time to run if we increase k, so, we put small value of k (k=2).

For the autoencoder part:

we have built our own sigmoid function, cross entropy loss metric for evaluating the model. we haven't used any inbuilt libraries, We used stochastic gradient descent while training.

5 Model Validation

Tackling the overfitting and dataset size problem: while using Decision Tree Classifier or Regressor, we have limited the tree depth and minimum number of samples needed to split on each node to avoid overfitting on to the data. For Ridge Regression, we have controlled the regularization parameter alpha to control overfitting

onto the data.

Since we had the problem of low dataset size, we used 3 methods to avoid overfitting and handle the modest size of dataset.

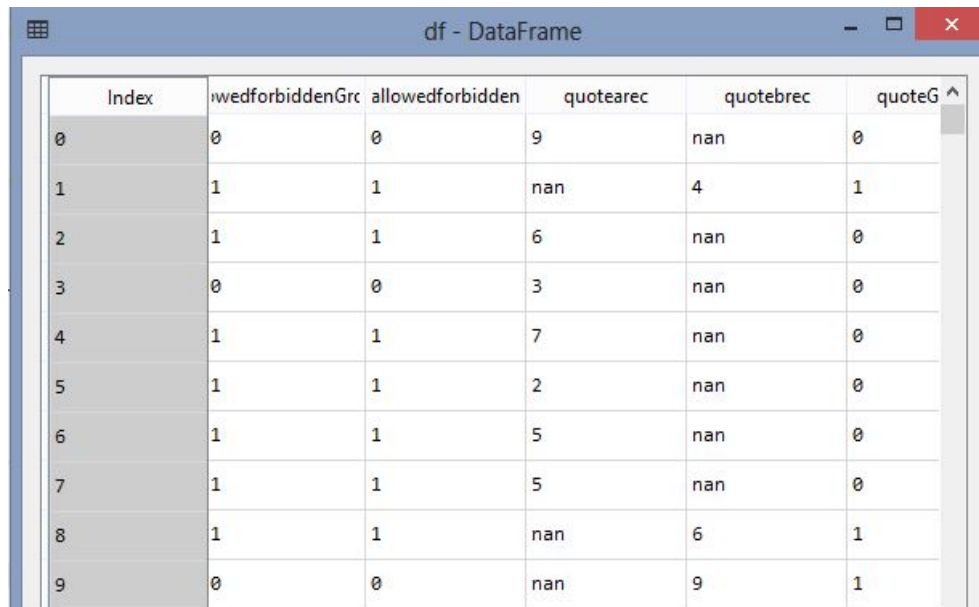
1. Bootstrapped the dataset
2. k-fold cross validation
3. Leave one out cross validation (here k=no of samples)

we used **bootstrapping** to increase the dataset where in we randomly sample the dataset. Bootstrapping seems to work better than cross-validation in many cases. In the simplest form of bootstrapping, instead of repeatedly analyzing subsets of the data, you repeatedly analyze subsamples of the data. Each subsample is a random sample with replacement from the full sample. We tried this but bootstrapping did not work well for decision trees

In **k fold cross validation** we take the base model where we predicted the unknown nan columns based on the known columns, we have splitted the train data into 3 datasets (because k=3) and tested on the remaining dataset respectively and averaged the final results of 3 folds.

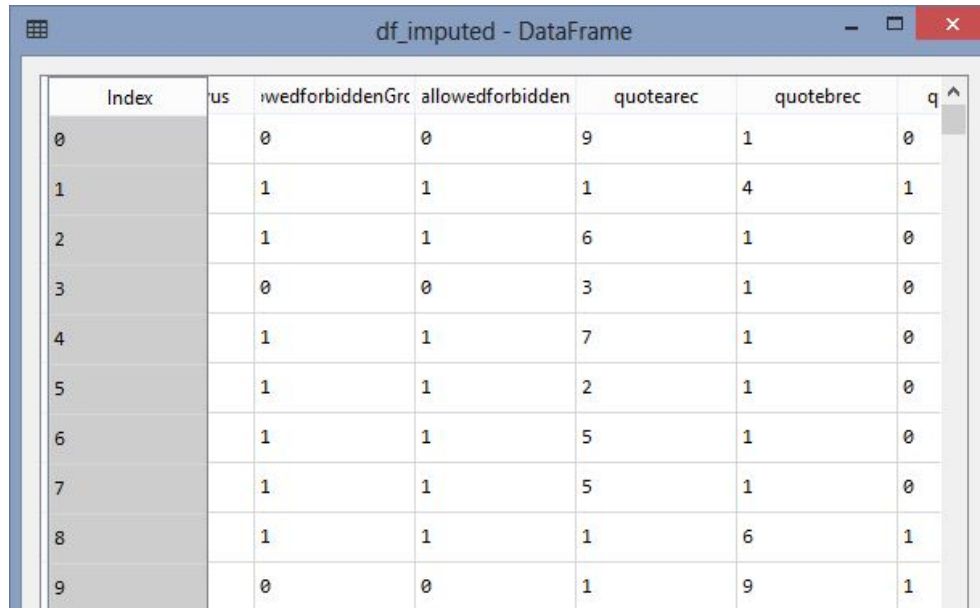
Leave one out cross validation is a variant of k fold cross validation scheme where k=number of samples. here we, test on a single datapoint while we train on n-1 samples. In this way we can get almost full dataset to train the model.

For the autoencoder model, we took the random 10 percent of the known dataset and took it as validation set and predicted the values. For continuous variables, we have taken the root mean square error as the evaluation metric where as for classification case, we took the f1_score and cross entropy as the evaluation metric. we have built our own sigmoid function, cross entropy metric for evaluating the model. The input data looked like the following :



Index	wwedforbiddenGrc	allowedforbidden	quotearec	quotebrec	quoteG
0	0	0	9	nan	0
1	1	1	nan	4	1
2	1	1	6	nan	0
3	0	0	3	nan	0
4	1	1	7	nan	0
5	1	1	2	nan	0
6	1	1	5	nan	0
7	1	1	5	nan	0
8	1	1	nan	6	1
9	0	0	nan	9	1

The output data looked like the following :
we observed that the output is overfitted to the data for autoencoder model since the data available is less.



Index	us	wedforbiddenGrc	allowedforbidden	quotearec	quotebrec	q
0	0	0	9	1	0	0
1	1	1	1	4	1	1
2	1	1	6	1	0	0
3	0	0	3	1	0	0
4	1	1	7	1	0	0
5	1	1	2	1	0	0
6	1	1	5	1	0	0
7	1	1	5	1	0	0
8	1	1	1	6	1	1
9	0	0	1	9	1	1

6 Model Evaluation

we have followed two approaches for model evaluation .

1st_approach:

For the categorical features, we have used F1_score, precision and recall as our evaluation metrics and for continuous features, we have used Root mean square error(RMSE) as the evaluation metric. We have used k.fold cross validation scheme to find the validation results. For example if we take the base model where we predicted the unknown nan columns based on the known columns ,we have splitted the train data into 3 datasets(because k=3) and trained on the 66 percent of data and tested on the remaining 33 percent data and averaged the final results of 3 folds.In the leave one out cross validation we test on each sample separately by training on n-1 samples. 2nd_approach:

we have randomly taken out 10 percent of the known values and put it to nan and predicted those values. For categorical columns, we have checked the f1_score, precision and recall and for continuous variables, we have checked the root mean square error. we have analysed that 1st approach is good for evaluating the model since we have less dataset and we get more models for prediction and also we get the averaged results.

Based on the above approaches, we have analysed some results. we sorted the f1_score which takes both precision and recall into consideration in decreasing order and found the features that the model is predicting correctly and incorrectly.

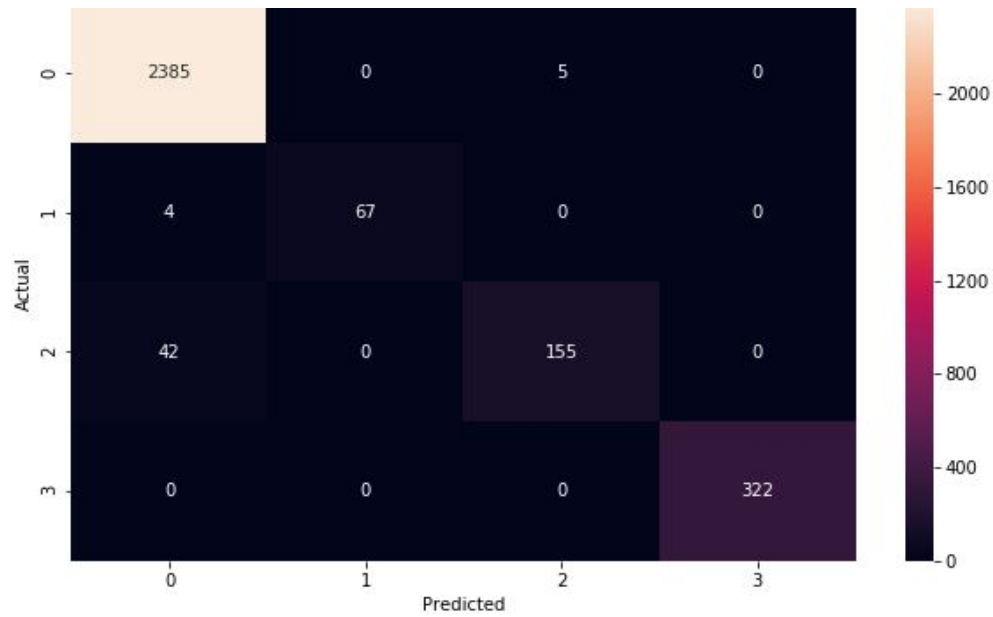
For example for the column 'compensation' prediction:

F1_score=0.959192614672

precision=0.992090553674

Recall=0.932406795551

The confusion matrix:



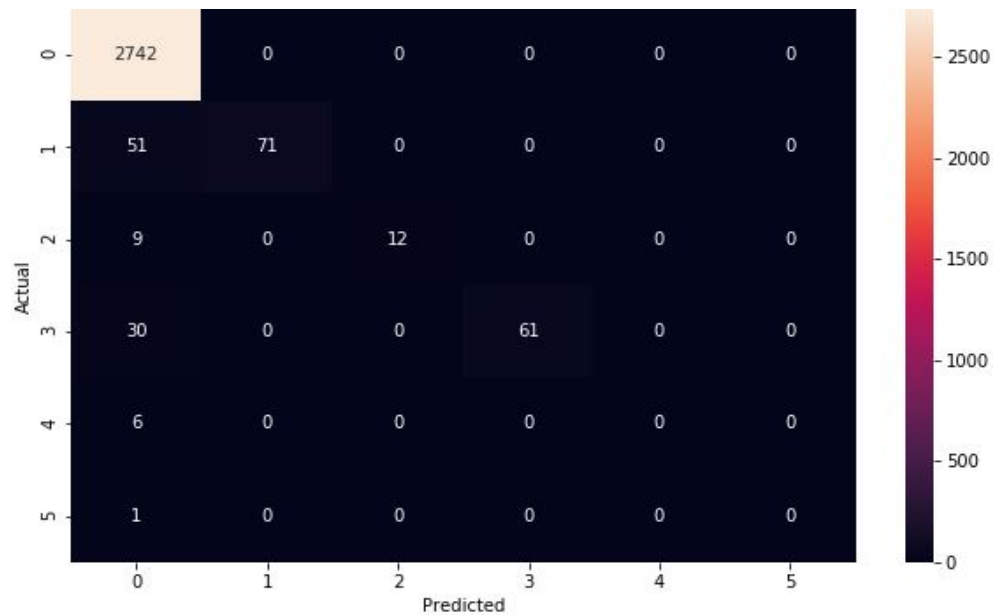
For example for the column 'recruitment' prediction:

F1_score=0.54137920063

precision=0.6609721733

Recall=0.470620909145

The confusion matrix:

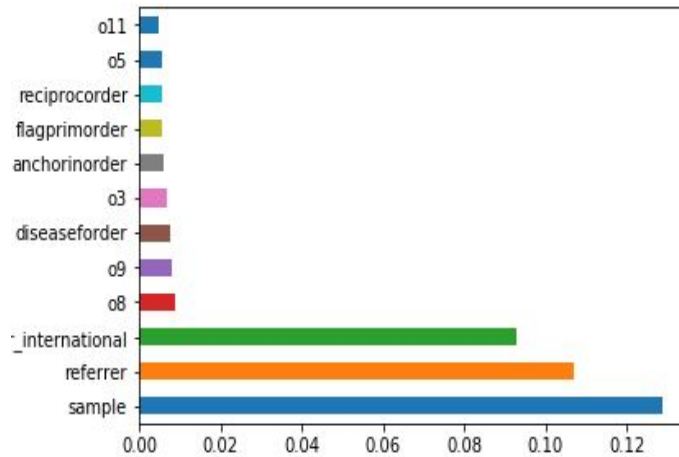


The model is good at predicting some features like compensation and not good at predicting recruitment,num_participants. the feature importance while predicting column 'compensation' are :

Feature_name	Gini_importance
referrer	0.084607
sample	0.151219
sunkgroup	0.000503271
anch1group	0.00125818
anch2group	0
anch3group	0
anch4group	0.00180077
gambfalgroup	0.000754907
reciprocityg...	0.00189925

For the tree models, the importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. The gini importance is calculated by the following formula: $N_t / N * (\text{impurity} - N_{t,R} / N_t * \text{right_impurity} - N_{t,L} / N_t * \text{left_impurity})$

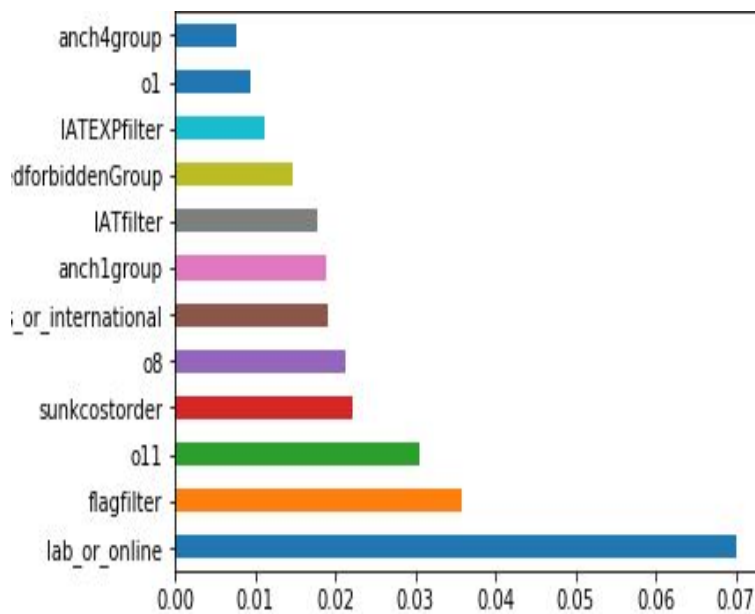
where N is the total number of samples, N_t is the number of samples at the current node, $N_{t,L}$ is the number of samples in the left child, and $N_{t,R}$ is the number of samples in the right child. The feature importance (while predicting compensation column) when plotted for top 12 showed the following result:



For continuous features, we have used ridge regression and decision tree regression. With ridge regression, the weights are penalized for less important features. for example predicting age using ridge gave the following weights:

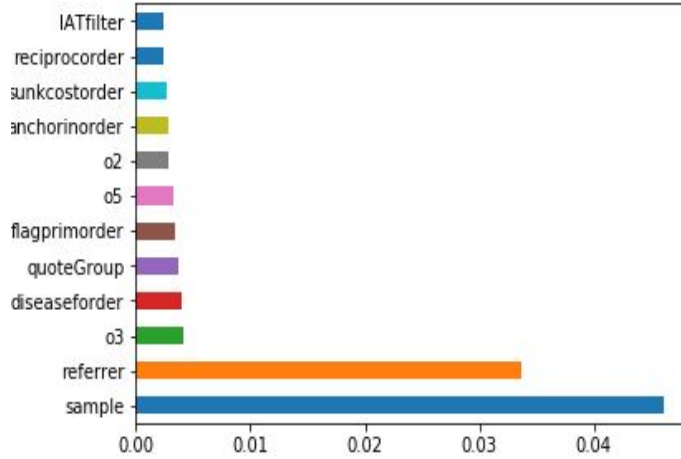
Feature_name	weight_value
referrer	0.00328927
sample	0.00328927
sunkgroup	-0.0150364
anch1group	0.0188393
anch2group	-0.66131
anch3group	-0.0115978
anch4group	0.00765911
gambfalgroup	-0.00546991
reciprocityg...	0.00416

And when we plot the top 12 weights , we get the following, may be the age is dependent on whether the answers are filled in lab or online. May be certain group of people may be doing it in lab and other online. that's why age may be dependent of lab or online column. Similarly we can check which feature is more important for predicting the values.



Basically what we observed is the features which don't have very high cardinality as well as very low cardinality are good at predicting the features. The features with high number of unique values doesn't help that much in predicting the other column. similarly very low number of unique values also cant distinguish between classes or predict the right value for regression.

Following figure shows the feature importance for prediction of 'recruitment' column:



If you observe the above figure ,we see the feature weights or features importances for two features are very high for prediction of nan values in a column, not all the features are significantly important. Similarly, for all the columns, we have seen that only a few columns are important in prediction.

If we go down in the decreasing order of weights, there are features with nearly zero weight, we eliminated the features with less weight and check the validation results, it gave better results after eliminating the variables with less weights.From this , we can say that only some of the features are important in predicting the age and not all the features. similarly we have checked with few other columns like lat11, RANCH2 which showed that not all the features are important in predicting the unknown values.

For finding which features contribute more to predicting all the features in the data. I averaged the model weights for each column prediction and averaged it for regression case, and took the average of feature importance for classification case.we observed that for most of the columns we observed that referrer and sample are getting high weights in the prediction, may be the feature is dependent upon the location of data collection.There are many features which didn't contriubute to the prediction like d_art, task_url etc. They are basically at random or doesn't provide any useful information while prediction.

For autoender part:

we evaluated the model by the cross entropy loss for the classification and root mean square error for the regression. we got high values of rmse values like in 1000s (for example we got 2835.91 for 5 iterations) . This may be because the values in the dataset are very high like in 10000s , so the predicted values are in 1000s. while taking square error , it might blow up. If we have more dataset , we might get the model trained more robustly and could have got better results.

To compare various models implemented by us , we took 10 percent of the data from continuous columns and set them to nan and predicted those values. It is done for 10 iterations and the RMSE scores are averaged. Similarly for categorical columns, we took 10 percent of the data from categorical columns and set them to nan and predicted those values. it is done for 10 iterations and the accuracy,f1_score, precision,recall scores are averaged. Here we are doing it for 10 iterations to get

diverse models and avoid overfitting.

Following are the results we got for continuous column prediction(here training algorithm used is ridge regression, Decision tree regressor also gave similar results):

we can see that MICE(Column by column imputation) is performing better than all the models and is faster too.

s/o	Model	Avg_Rmse
1.	Base_Model_1 st _variant	2350.262
2.	Base_Mode_2 nd _variant	1950.524
3.	Base_Mode_3 rd _variant	1992.752
4.	KNN_Based Imputation	2600.856
5.	MICE(Column by column imputation)	1800.258
6.	MICE(Value by value imputation)	1963.159
7.	MICE(Column followed by value)	1750.257
8.	MICE(Value followed by column)	1900.123
9.	AutoEncoder	3212.256

Ensemble model which is the average of top 3 models gave us the avg_rmse 1782.623 which is not the best one but if we had more data it would have performed better.

Following are the results we got for categorical column prediction(here training algorithm used is decision tree):

we can see that base model with 2nd variant (i.e., predicting unknown values with known values and appending the results to get next predicted values and so on) is performing better than all the models.

s/o	Model	Avg_Precision	Avg_Recall	Avg_f1_score
1.	Base_Model_1 st _variant	.823	.724	.7703
2.	Base_Mode_2 nd _variant	.862	.845	.8534
3.	Base_Model_3 rd _variant	.756	.782	.7688
4.	KNN_Based Imputation	.714	.812	.7598
5.	MICE(Column by column imputation)	.801	.792	.7964
6.	MICE(Value by value imputation)	.625	.852	.7210
7.	MICE(Column followed by value)	.821	.752	.7849
8.	MICE(Value followed by column)	.825	.796	.8102
9.	AutoEncoder	.692	.742	.7162

Ensemble model which is the majority voting of top 3 models gave us the avg_f1_score .8234 which is not the best one but among the top2. If we had more data it would have performed better. For future work , we may try giving different weights to each classsifier. The weights can be predicted by training .

7 Increasing the dataset

Increasing the dataset is one of the interesting tasks of this project.We have tried few approaches to increase the dataset.

1st_ approach:

we tried increasing the dataset with autoencoders. Autoencoders are trained using both encoder and decoder section, but after training then only the encoder is used, and the decoder is trashed. So, basically we got dimensionality reduction. we have set the layer between encoder and decoder of a dimension lower than the input's one. Then trash the decoder, and use that middle layer as out put layer which gave the compressed version of input data. we evaluated whether the reconstructed data is good or not by using reconstruction loss (here cross entropy loss for classification and rmse for continuous).we have also tried using the decoder output to append to the dataset, but the results we got by second approach were similar to the result without increasing the data.

2nd_ approach:

we have tried using SMOTE to increase the dataset,but it is for categorical variables.The main purpose of using SMOTE is deal with problem of imbalanced classes.we found that the SMOTE didn't produce the good reconstructed output . we checked the goodness using rmse and cross entropy

3rd.approach:

we tried bootstrapping to increase the dataset where in we randomly sample the dataset .Bootstrapping seems to work better than cross-validation in many cases.In the simplest form of bootstrapping, instead of repeatedly analyzing subsets of the data, we repeatedly analyzed subsamples of the data. Each subsample is a random sample with replacement from the full sample.We tried this and we got better results than the previous approach.

After bootstrapping, the dataset looked like(for categorical):

Name	Type	Size
bigdata	DataFrame	(12688, 144)

4th_ approach:

Basically increasing the dataset , we want to get a robust model on any data. This can also be done by leave one out cross validation where you leave out a sample and train the remaining part of the dataset and predict the left over one. here for predicting each sample, we are getting almost whole dataset. This is one way to tackle the less data issue. we have tried this and results are almost same from cross validation method.

5th_ approach: we also tried increasing the dataset with eliminating some of the features . the features to eliminate are chosen by the decision tree algorithm, we can tell the features that are important in splitting and which gives some information. we eliminated the features with importance less than 0.001 and append it to the dataset again. This method gave us good validation score for predicting unknown values compared to other methods.

6th_ approach:

Artificially increased the number of the samples via duplication of columns/rows. we tried giving weights to the column so the rows with larger value in the column are more likely to be sampled. we saw this is one of the methods to increase the dataset. Here the columns which have no null values are chosen first as it may give more information about the data. This might be useful sometimes because for example in random forest, we take the random subset of data and features and the row/column duplicated may get in most of the decision trees compared to non duplicated row. this might influence the model performance. we applied this on our dataset but couldn't get good results. This might be due to the fact that the answers are diverse and most of them are nans and dataset size is also small.

8 Bonus

There are two types of natural language answers.

1. categorical text
2. Text based on Opinion or feedback

For the 1st type of text, we label encoded each category since there are only few categories in each column, we tried using one hot encoding but it is taking a lot of time.

For the 2nd type of text, we tried 2 approaches to convert the sentence to vector notation:

1. Bag of Words representation
2. Tfidf representation

1. Bag of Words representation:

Bag of Words (BOW) is a method to extract features from text documents. These features can be used for training machine learning algorithms. It creates a vocabulary of all the unique words occurring in all the documents in the training set. In simple terms, it's a collection of words to represent a sentence with word count and mostly disregarding the order in which they appear. we will go through the steps in detail.

Flow Diagram.



Input text column looked like :

Index	Type	Size	Value
2091	str	1	nan
2092	str	1	nan
2093	str	1	we talk about religious and attitudes that muslim has towards america
2094	str	1	nan
2095	str	1	nan
2096	str	1	nan
2097	str	1	its normal the muslim stranger for is like any other stranger ,i don,t ...
2098	str	1	nan

Step 1: Clean text

First we converted whole sentence to lower case. Then, we removed stopwords from the sentences. Stopwords are words which do not contain enough significance to be used without our algorithm. We would not want these words taking up space in our database, or taking up valuable processing time. For this, we can remove them easily by storing a list of words that you consider to be stop words.

Step 2: Tokenize

Tokenization is the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens. Tokens can be individual words, phrases or even whole sentences. In the process of tokenization, some characters like punctuation marks are discarded. The method iterates all the sentences and adds the extracted word into an array. The tokens look like:

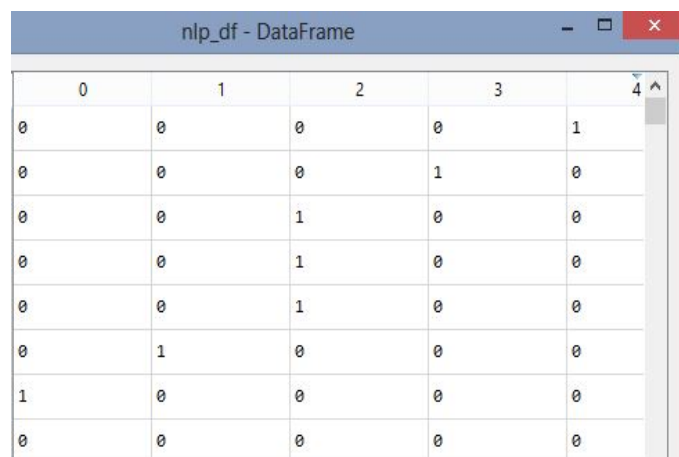
Index	imagineddescribe
2091	['nan']
2092	['nan']
2093	['talk', 'religious', 'attitudes', 'muslim', 'towards', 'america']
2094	['nan']
2095	['nan']
2096	['nan']
2097	['normal', 'muslim', 'stranger', 'like', 'stranger', 'care', 'ther', 'elegation', 'people'...
2098	['nan']

Step3: Build Vocabulary and generate vectors

Used the methods defined in steps 1 and 2 to create the document vocabulary and extract the words from the sentences. The vocabulary looked like:

Index	Type	Size	
20	str	1	abbastanza
21	str	1	abbia
22	str	1	aperto
23	str	1	abitudini
24	str	1	abile
25	str	1	abroad

Each sentence was compared with our word list generated. Based on the comparison, the vector element value may be incremented. These vectors can be used in ML algorithms for various tasks like document classification and predictions. We implemented the above method and generated vectors. Here nan is also treated as a word as it is a string column. The output data looked similar to this: Here each row corresponds to the encoded sentence to vector



	0	1	2	3	4
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	0	0
0	0	0	1	0	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

From the above figure, you can see there is lot of sparsity in the matrix because the words are not frequent in all the documents. we have appended the vector representation of the sentence of each row to the original data. When tried to find out the unknown values, the weights of the coefficients are almost zero for most of the entries (almost all). So, it didn't give much information about the other features. This may be due to the fact that most of the entries are zero in the vector representation of the sentence and the model may have thought the sparse numbers (>0) as noise.

Limitations of Bag of Words:

1. Semantic meaning:

The basic BOW approach does not consider the meaning of the word in the document. It completely ignores the context in which it's used. The same word can be used in multiple places based on the context or nearby words.

2. Vector size:

For a large document, the vector size can be huge resulting in a lot of computation and time. You may need to ignore words based on relevance to your use case.

Possible Extensions:

For example, instead of splitting our sentence in a single word (1-gram), you can split in the pair of two words (bi-gram or 2-gram). At times, bi-gram representation seems to be much better than using 1-gram. These can often be represented using N-gram notation.

From the previous figure, you can see there is lot of sparsity in the matrix because the words are not frequent in all the documents. So, we moved forward to another implementation `tf_idf`.

2. Tfidf Representation:

The TFIDF weight is used in text mining and IR. The weight is a measure used to evaluate how important a word is to a document in a collection of documents. When using a simple technique like a frequency table of the terms in the document, we remove stop words, punctuation and stem the word to its root. And then, the importance of the word is measured in terms of its frequency; higher the frequency, more important the word.

In case of TFIDF, the only text pre-processing is removing punctuation and lower casing the words. We do not have to worry about the stop words. Implemented method:

TFIDF is the product of the TF and IDF scores of the term.

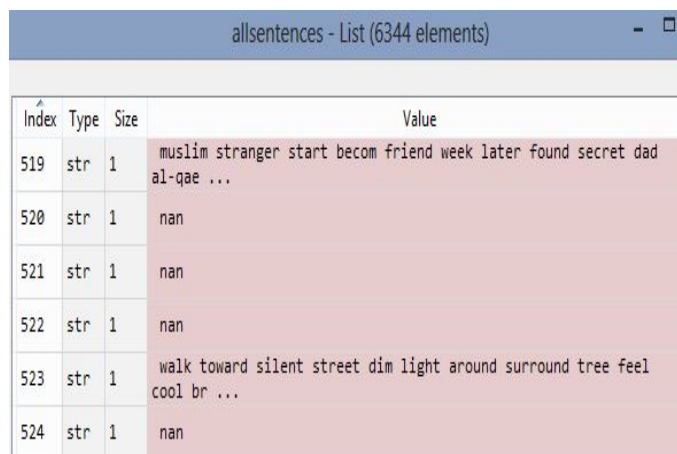
TF = number of times the term appears in the doc / total number of words in the doc

IDF = $\ln(\text{number of docs} / \text{number docs the term appears in})$

Higher the TFIDF score, the rarer the term is and vice-versa. TFIDF is successfully used by search engines like Google, as a ranking factor for content. **The whole idea is to weigh down the frequent terms while scaling up the rare ones.**

We will discuss each step by applying on our dataset.

First we will be preprocessing the data like converting to lower case, removing punctuation marks, html tags, removing stop words etc. The preprocessed sentences look like :



Index	Type	Size	Value
519	str	1	muslim stranger start becom friend week later found secret dad al-qae ...
520	str	1	nan
521	str	1	nan
522	str	1	nan
523	str	1	walk toward silent street dim light around surround tree feel cool br ...
524	str	1	nan

Term Frequency:

This measures how frequently a word occurs in a document. This highly depends on the length of the document and the generality of word, for example a very common

word such as “was” can appear multiple times in a document and with the length of the document the count increases, so to normalise the value, we divide the frequency with the total number of words in the document.

So, in worst case if the term doesn't exist in the document, then the TF value will be zero and in other extreme case, if all the words in the document are same, then it will be one. The final value of the normalised TF value will be in the range of [0 to 1]. 0, 1 inclusive.

Hence we can formulate TF as follows.

$tf(t,d) = \text{count of } t \text{ in } d / \text{number of words in } d$ still there are few problems, for example, stop words which are the most common words like “is, are” will have very high giving those general words a very high importance, that is the reason we also consider IDF along side TF.

Document Frequency:

This measures the importance of document in whole set of corpus, this is very similar to TF. The only difference is that TF is frequency counter for a term t in document d , where as DF is the count of occurrences of term t in the document set N . we consider one occurrence if the term consists in the document at least once, we do not need to know the number of times the term is present. $df(t) = \text{occurrence of } t \text{ in documents}$ To keep this also in a range, we normalise by dividing with the total number of documents. Our main goal is to know the informativeness of a term, and DF is the exact inverse of it. that is why we inverse the DF

The document frequency looked like for column 'imagineddescribe' looks like:

V - Dictionary			
Key	Type	Size	
treat	int	1	1
tree	int	1	163
treestand	int	1	1
trekk	int	1	1
tress	int	1	2
tri	int	1	8

you can see that word 'tree' occurred in 163 documents etc.

Inverse Document Frequency:

IDF is the inverse of the document frequency which measures the informativeness of term t . When we calculate IDF, it will be very low for the most occurring words such as stop words. This finally gives what we want, a relative weight. $idf(t) = N/df$ Now there are few other problems with the IDF, in case of a large corpus, say 10,000, the IDF value explodes. So to dampen the effect we take log of IDF. In worst case, there could be no document which has 0 occurrence, and we cannot divide by 0. so to smoothen the effect we generally add 1 to the denominator.

$$idf(t) = \log(N/(df + 1))$$

Finally, by taking a multiplicative value of TF and IDF, we get the TF-IDF score,

there are many different variations of TF-IDF but for now let us concentrate on the this basic version.

$$\text{tf-idf}(t, d) = \text{tf}(t, d) * \log(N/(\text{df} + 1))$$

These two methods are used to convert the text to feature vectors. Yes, they provided the useful information which is confirmed by the validation results.

The tfidf representation looks like the following:

tf_idf - Dictionary (16362 elements)			
Key	Type	Size	Value
(523, 'seem')	float64	1	0.41723895135217798
(523, 'silent')	float64	1	0.50388234892217121
(523, 'street')	float64	1	0.36655581283865746
(523, 'surround')	float64	1	0.40987751162365405
(523, 'toward')	float64	1	0.42558466339121065
(523, 'tree')	float64	1	0.22846239596815535

Things to infer from the above tfidf chart: you can see that 'tree' word has large number of count i.e, it is present in most of the documents , so its importance is decreased while the 'silent' word importance is increased.

we have appended the vector representation of the sentence of each row to the original data. When tried to findout the unknown values, the weights of the coefficients are almost zero for most of the entries(but not all). So, it gave some information about the other features.This may be due to the fact that it takes relative importance of the words in each sentence.

Bonus 2:

For the natural language answer prediction, we label encoded the text word (or combination of words). For example morning to 1, night to 2, evening to 3 . we later thought one hot encoding is apt for the text column since our machine learning algorithm might assume that average of 1 and 3 is 2 and predict the answers but in actual the average of morning and evening does not mean night. So, we one hot encoding(putting 1 at the respecting column and remaining zeros) after label encoding the natural text columns(which resembled categorical columns).we can predict the natural language column like any other column now and finally predict the column nlp word (or text) based on the predicted value.But the problem with one hot encoding case is it may be possible that both the columns can be 1 while predicting which corresponds to two words(example morning and evening) at the same time. If we want to convert back to the text format , we can replace the encoded numbers with the text as we have the information in dict.we have observed that label encoding performed better than one hot encoding.

Before encoding categorical nlp columns looked like:

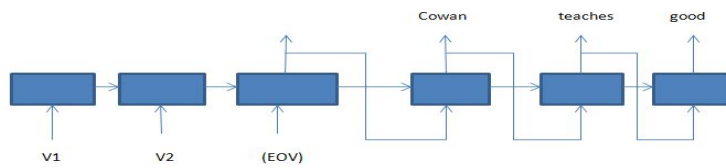
train_nlp_cat - DataFrame		
o4	o5	o6
diseasef	scales	allowedf
moneypri	quote	sunkcost
moneypri	anchorin	gamblerf
allowedf	scales	sunkcost
gamblerf	allowedf	sunkcost
scales	allowedf	quote

After encoding categorical nlp columns looked like:

train_nlp_cat_encoded - DataFrame		
o4	o5	o6
0	0	0
1	1	1
1	2	2
2	0	1
3	3	1
4	3	3

Bonus 3:

Generating the natural language answers is a bit difficult. As we have done in the previous question like converting the natural language to vector. we can think of the text column as vector of columns of numerical values.(For example bag of words representation) This becomes similar to the numerical column prediction as we have done in the project.For example, if we predict the values [1,0,0,2,1] ,index1 corresponds to ml , index2 subject index3 very index4 interesting index 5 study, then we can say that there are 1 occurrence of ml, 2 occurrence of interesting, 1 occurrence of study in the sentence .The main draw back of this approach is we don't know the correct ordering of the words in the predicted sentence. we just know whether the word in the built vocabulary occurs or not.We tried one more method like finding the cosine distance of the vectors and take the sentence with most similarity. This gives us the sentence with almost same words. But the draw back is there is nothing like training the model, its just calculating the nearest distance vectors. So, for generation of natural language answers, we can use RNN and give as input the known numerical columns and train in such a way that outputs the ordering of words. The architecture looks like:



we tried generating natural language answers, but they are not that great. Some of the words are getting generated but they lacked the meaning. If you looked closely at the 'imagineddescribe' column in the data, some of the sentences even lack the meaning in the original dataset. so output sentences generated may have lacked meaning because of this. Sample output sentence generated looks like:

generated_sentence - List (1 elements)			
Index	Type	Size	Value
0	str	1	soothinggg lakii tres lewej bird\$ feet soakk gras air

One way to check the quality of prediction is checking the similarity of the predicted vector with the original true vector. we used cosine similarity for example.