CS 536: Machine Learning Final Project

Data Completion and Interpolation

Aditya Vyas av634
Vedant Choudhary vc389

May 12, 2019

For the final project, the data set is of the form $x^1, x^2, ..., x^m$, and we have to construct a system for predicting or interpolating missing features (frequently given as empty or NA) from the present features in new records. In this case, instead of singling out a single feature or factor for prediction / regression / classification, any feature of X might be missing and needs to be predicted or reconstructed from the features that are present - and the features that are present may vary from instance to instance.

1 The Data

For this project, we choose Many Labs 1 dataset. It comes from psychology studies in the Many Labs series, an attempt to test the replicability or generalizability of psychological effects. Multiple labs attempted the same experiments, to determine the extent of the results of those experiments generalized and could be relied on in diverse circumstances. Being psychology studies, the data set is a record of personal answers and reports from subjects on a variety of questions relating to topics like perception and mental biases. Additionally, the data set includes demographic information about the subjects and information about the researchers and experimental environments. All are potentially useful for the problem of prediction and interpolation. There are 28 psychological studies, across 60 different labs, trying to determine to what extent the originally studied effect was reproducible. Questions given to subjects touched on a diverse array of topics from nationalism, to the perceptions of numbers, to feelings about art and mathematics.

2 Requirements

All the codes written for this project can be accessed at:

$$https://github.com/vedantc6/ML\_FinalProject$$

. The codes have been written in Python language and has used only the basic essential libraries like numpy, math, pandas, statistics. For visualization, seaborn and matplotlib has been used.

2.1 Describe your model

2.1.1 Representing the data

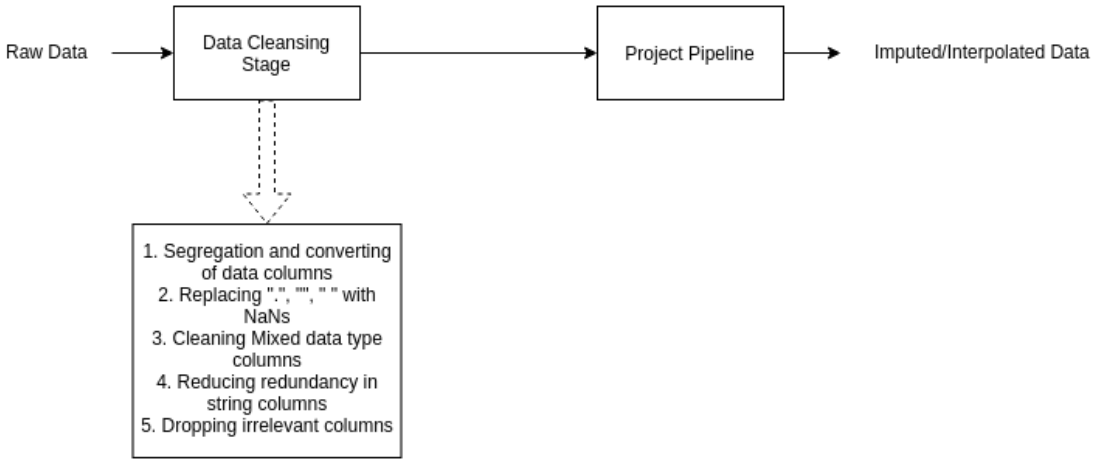

Figure 1: High Level Flowchart for the Project

The data is quite varied in the sense it has a lot of columns, and different language texts. Before building the model, it is very important to clean the data, so that we avoid the scenario "Garbage in, Garbage out". For cleaning the data, we followed the following steps:

- We segregate the data depending on its data types since they are quite diverse and range from integer values, float values, categorical, binary values, ordered categorical values, natural language responses etc. The segregation is semi-manual. We take help from two sources to determine which data type the column belongs to.
  - Unique values: Finding out unique values of a column through code helped us see what kind of data is being stored. A long list of numbers meant it is a numerical column
  - Code book: We took help from the Code book attached with the assignment to figure out what kind of values can occur in a column. Columns like "feedback" or "imagineddescribe" were pretty explanatory through the code book that they'll be string columns and potential columns for natural language processing.

  Essentially, we have segregated the data into the following categories (based on our coding process flow):

  - Mixed-columns: These are the columns which have both numerical data and strings. After checking it's unique values and referring to code book, we established the fact that these are in reality categorical columns and some kind of cleaning has to happen.
  - All NaN-columns: These columns have no value in the data. We dropped these columns since it is practically impossible to predict something on which we have no

knowledge. There were only three columns like these - $task_status$, $task_sequence$, $beginlocaltime$

- Numeric-columns: Columns which contain numeric values (either integers or floats). One step of caution here was to double check from the code book what each column means. This sanity check prevented us from having a categorical column in numeric columns.
- Date-columns: There were some columns which had date-time stamps, which cannot really be applied for predicting or imputing missing values. Finding them were easy as almost every date column had "_date" in their column name.
- Exclude from data-columns: We believe these columns do not add any weightage to the model since they are just URLs. Finding them was again easy, as the common thing these columns shared in their name was "_url". Apart from urls, there are some other columns too, which are considered redundant according to the code book.
- Categorical columns: These columns contain categorical data and should not be confused with numeric columns. Columns such as "artwarm", "mathwarm" are ordered categorical features as they have a range of values from 0-100.
- NLP features columns: These are string columns which represent textual data and can possibly be used to derive features through natural language processing.
- String columns: Although NLP columns should be a part of string columns, but the way we have set up our code, we have segregated those. Later on, mixed-columns and categorical columns are added to string columns (after converting everything to numbers - label encoding)

- After segregating the data columns, they are converted to their respective columns. Further, as a sanity check, we replace row values having "." (dot), " " (space), "" (empty) to NaNs (which we will predict later).

- Few columns which have been cleaned:

  - As discussed before, we clean the mixed columns and convert them to categorical data. An example: column "flagsupplement2" is a question from $FlagPriming$ (Carter, Ferguson, Hassin, 2011;Study 2). It asks the user to rate from 1-7 the extent to which the typical American is a Republican or Democrat (1=Democrat, 7=Republican). Being a mixed column, there were values like "Democrat", "2", "3" .... "Republican", so we cleaned columns like these.
  - Column user_agent has been cleaned in order to be used as a feature for our model. It has been converted from a value like

    $$Mozilla/5.0(WindowsNT6.1;WOW64;rv:17.0)Gecko/20100101Firefox/17.0$$

    to $Windows$.
    This results in having only around 4-5 unique values in the column, making it a viable label encoded feature.

– Column $exprunafter2$ has been similarly cleaned. It had multiple values which meant the same thing, but were different due to spelling mistakes or a different style of writing.

– The same methodology has been applied to the column $nativelang2$

Finally, before the model is passed onto the model, we have dropped all the unnecessary columns described so far, reducing the feature space from 382 columns to 267 columns. For the remainder of this paper, the model is established to fill in the missing values for this new cleaned dataset.

2.1.2 Model Pipeline

After having the cleaned data, the next step is to create a Machine Learning model pipeline, which will take the clean data, do some processing and in the end, return the filled dataset. The following steps are taken to ensure a smooth pipeline:

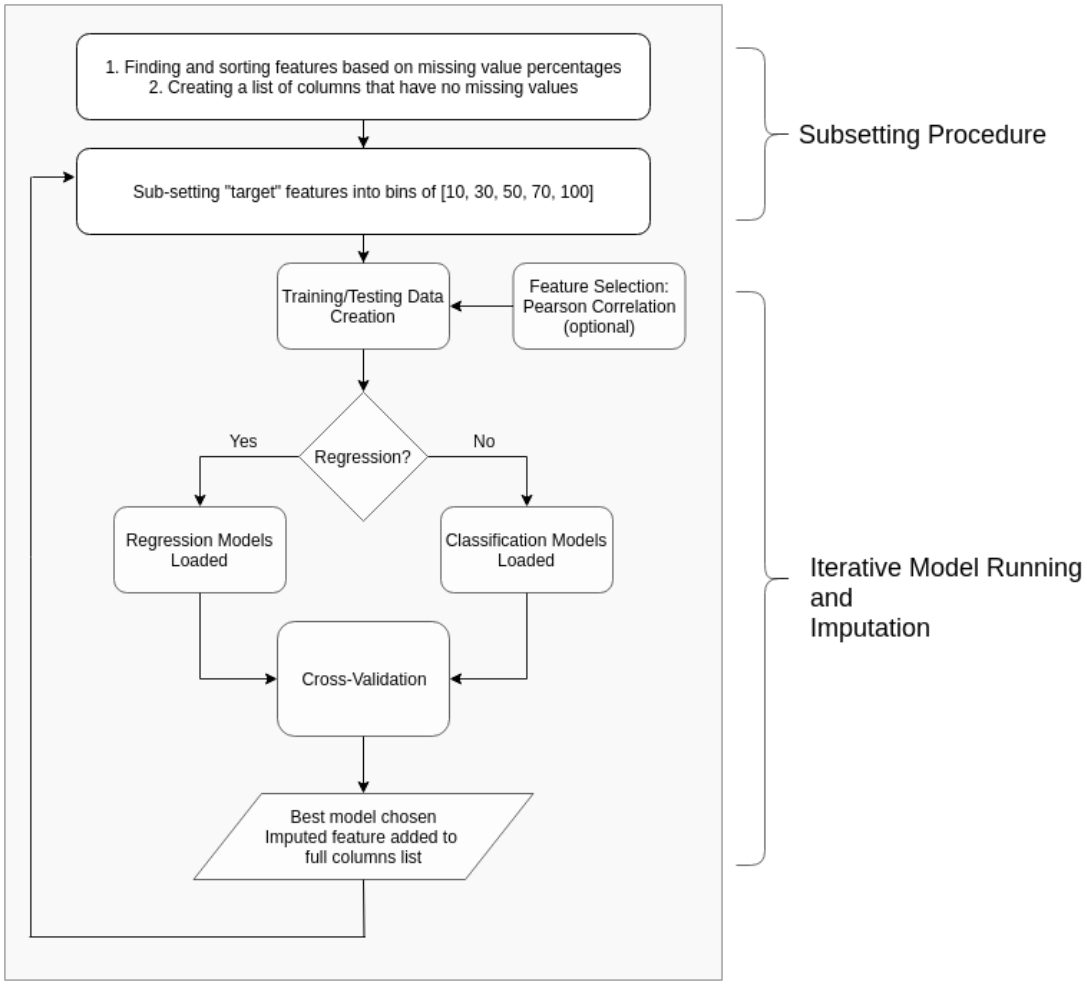

Figure 2: Inside the Pipeline

- Our pipeline methodology starts imputing columns which have the lowest percentage of missing values i.e. columns having 1% missing values will be imputed before columns having 10%. This procedure has been chosen because of the notion that this will ensure that the columns which can be predicted most accurately (since they have the most number of data points) will be imputed first. Then, for the next column to be imputed, the previous column is attached to the training data. An example:

Col1	Col2	Col3	Col4
1	6	87837	787
2	7	NaN	5415
3	NaN	NaN	7507
4	545	18744	7560
5	88	NaN	6344

  In the above table, Col2 has 20% missing values, while Col3 has 60% missing values. How our first stage of pipeline will work is, first it will take Col1 and Col4 to impute values into Col2, then it will take Col1 Col2 and Col4 to impute values in Col3.

- Along with finding the set of missing value data, we also find the data which has no empty value or NaN.

- After this, we have created subsets of missing value data according to their percentages. This step has been added for evaluating how good our model is doing in terms of missing value percentages.

- Our procedure is an iterative procedure, where the first thing it does is selects the first subset (columns containing less than 10% of missing data). Then for every feature/column, it does the following:

  - Training and testing data is first created from the base data. This is done by selecting all the columns which have 100% filled values along with the feature to be predicted. For train, only those rows are kept which have filled values in feature column, and for test, rows containing NaNs are kept. Additionally, we have an option to either directly go with all the columns of training and test set, or use some basic feature selection algorithms like Pearson Correlation to reduce the feature space for predicting values.
  - After this, we have a check to see if the current column/feature is a numerical one or categorical one. If it is a numerical columns, a set of regression models are loaded, else a set of classification models are loaded. The models present in our project are:
  - When the training data, test data, and model is fixed. We enter cross-validation function in our code. Here, the first step is splitting the data into k-folds. The details of this technique will be covered in the model validation part of this report. After the models have been validated on k-folds, an average of their evaluation score is used. The value imputed is also the average prediction value from all the k-folds.

Regression	Classification
Mean	Mode
Linear	Logistic
Ridge	Softmax
KNNs	KNNs
Decision Trees	Adaboost
Neural Networks	-

Table 1: Machine Learning Algorithms Used

- After cross-validation, the best model for a particular column is selected: lowest loss if regression, highest accuracy if classification. The predicted values are added into the actual data, and the full value columns are incremented by one, adding the latest imputed column into it.

2.2 Describe your training algorithm

The following is the list of machine learning algorithms we tried on the dataset. Everything is written by us, from scratch, using only Numpy. The algorithms are categorized according to the class of problems they belong to, i.e. Regression or Classification.

- Supervised Learning: We used the following supervised learning algorithms to impute the missing values.
  - Regression Algorithms:
    * Mean Imputation (Baseline): This is the baseline model for regression algorithms. Since this is not a smart algorithm, in the sense that it takes no knowledge of different samples and the loss it incurred on a prediction, we have chosen it to be the base model. There is not much to say about this algorithm since all it does is take the mean of dependent variable (from the training set) and impute it in the validation set. Ex: Training set has values 1 10 15 20. The mean will be 11.5, so the NaNs will be replaced by this value.

    * Linear Regression: We wanted to try an algorithm which was a little complex than our baseline mean imputation model but not so complex. This is why we used linear regression for making predictions. For training, we used mini-batch stochastic gradient descent and optimised the mean squared error loss.

    * Ridge Regression: Although, the linear regression model did perform better than the mean imputation model, there was no way to prevent it from overfitting to the data. Furthermore, with just 6000 data points and even less than that for our training data, it is very easy to overfit to the data and this led us to use ridge regression. We used a regularization coefficient of 10 for

this model.

* K-Nearest Neighbors: This is one of the simple models we used for the project. The reason for using this model was to see if simple models is able to learn patterns in the data or not. The mean imputation method was able to generate decent score for many columns and so it made sense to impute NaNs based on the nearest neighbours. We used the euclidean distance as the metric to get the distances between the data points, k = 5 as the number of neighbors and mean value of the top k neighbours.

* Decision Trees: We used decision trees because we wanted to see how do tree based models perform against the baseline and the simple linear models. The splitting criteria was based on root mean square value of a node - For a split, calculate the root mean squared error of the target column Y with Y_{mean}.

$$RMSE = \sqrt{\sum (y - y_{mean})^2} \tag{1}$$

  Finally, the RMSE of a split is the sum of the RMSE for the 2 child nodes. Furthermore, we also tried to prevent overfitting in the model by :

  · Limiting the tree depth

  · Limiting the number of samples on which to split a node

  Although, our decision trees algorithm was correct, we could not use them for training as it is very slow and takes a lot of time to split the data and build the tree. Hence, we had to limit the tree depth to 2 which lead to bad performance.

* Neural Networks: This was the most complex model we decided to try. We kept a very simple architecture for the neural network - one hidden layer, 10 neurons, learning rate = 0.03 and batch size = 32. The final layer had a single neuron for the output value. However, neural networks performed the worst among all our models since they require huge amounts of data to learn anything. During our training, the neural network model did not learn anything. Hence, for us the neural network was not a good model to use for the dataset.

– Classification Algorithms:

* Mode Imputation (Baseline) This is the baseline model for classification algorithms. Since this is not a smart algorithm, in the sense that it takes no knowledge of different samples and the loss it incurred on a prediction, we have chosen it to be the base model. There is not much to say about this algorithm since all it does is take the mode of dependent variable (from the training set) and impute it in the validation set. Ex: Training set has values 0 1 2 3 2. The mode will be 2, so the NaNs will be replaced by this value.

* Binary Logistic Regression: There are a lot of columns in the data which are binary and so we decided to start with a simple model. Our logistic regression uses two types of solvers - Newton solver and Stochastic Gradient Descent (SGD) solver. We tested both the types of solvers. Newton solver was not a good choice because it is based on the Newton-Rhapson method of updating the weights of the model and since it calculates the hessian of the loss function, we ran into Singularity Matrix error a lot. The mini batch gradient descent method computed the derivative of the loss function and was relatively stable giving us good results within less number of iterations.

* Softmax Regression: Softmax regression model was used for multiclass columns - columns with multiple categories. The algorithm uses mini-batch stochastic gradient descent to update the weights. The model optimizes the cross entropy loss and calculates the probabilities for each unique class of the target using the softmax function.

* K-Nearest Neighbors: For classification too, all the parameters of our earlier KNeighbours regression model was used. The only difference here is that we use the mode of the top k neighbours' target values to generate the predictions.

* Decision Trees: For classification, we tried the following metrics to split a node - gini impurity and entropy. We observed that gini impurity was better than entropy. Again, like the regression model, the classification was very slow to run and train and so we did not use decision tree classifier in our final training. Here also, we use the same techniques as the regression model to prevent overfitting.

* Adaboost: Our Adaboost classifier was a binary classifier and was used to predict only the binary columns. Using a large number of weak learners to train on our data, the Adaboost algorithm gave us a decent performance. For the weak learner, we tried
  - Binary logistic regression model with a very low learning rate and very few epochs (10)
  - Decision tree classifier with a max depth of 1

  We found that binary logistic regression performed way better than the decision tree. We tuned the number of estimators/weak models from 100 to 500 with 100 estimators giving us the optimal results.

- Unsupervised Learning and Generative Models: We also used some unsupervised algorithms to cluster the data and generative models to generate more synthetic data. Information regarding the below algorithms is presented in Section 2.5 Generate data.
  - K-Means Clustering

– Gaussian Mixture Models

– Variational Auroencoder

2.3 Describe your model validation

The dataset has around 6,500 rows and more than 250 (probably useful) columns. This means it is a fairly small dataset (in terms of ML projects) and a good validation process has to be instantiated so that the model is run properly without bias, over-fitting etc.
The goal of a cross-validation technique is to estimate the expected level of fit of a model to a dataset that is independent of the data that were used to train the model. We have used a K-Fold cross-validation technique in the project. In k-fold cross-validation, the training

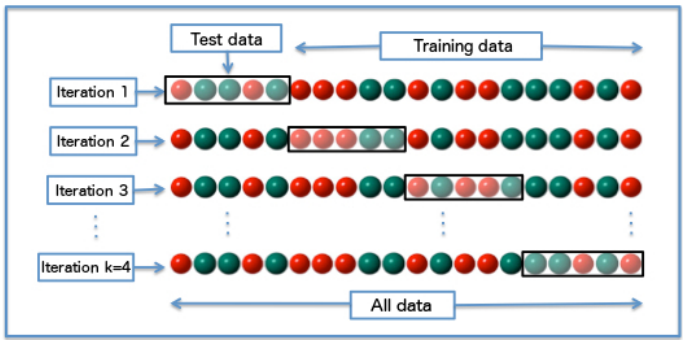

Figure 3: K-Fold Validation

data is partitioned into k equal-sized sub-samples. In our case, if the total number of data points are not exactly divided by k, there are unequal sub-samples. This has been done to ensure that each and every data point is used for training/validating the results.
Once the k sub-samples are formed, a loop is run for k times. In every iteration, (k-1) sub-samples are used for training data, and one sub-sample is used for validation. The training and validation sets are disjoints. The process is set-up such that each of the k sub-samples are used exactly once for validation. When all the iterations are done, the average of the results is taken to produce a single estimation. For our model, we have chosen k = 10.
Advantages of using cross-validation:

- Since, we had a list of algorithms to test, we decided to use k-fold cross validation as these techniques are good for comparing the performance of different machine learning models on the same data set.
- It helps in generalizing the data, since the model is trained and then tested on independent datasets k times. This helps in reducing over-fitting or selection bias
- K-fold cross estimation has a lower variance than a single hold-out set estimator (hence, our preference), which can be very important if the amount of data available is limited (our case). The variance is lowered since k-fold averages over the k partitions, so the performance estimate is less sensitive.

2.4 Evaluate and analyze your model/data

Since we are dealing with both regression and classification problems for imputing the data, there are two evaluation metrics: loss for regression (Mean Squared Error is used) and accuracy for classification. The models for each problem have already been defined in Table 1.

Overall, the distribution for the dependent/predictive feature is shown in Fig. 4. Majorly, the features to be predicted are from classification problem set, almost 2/3.
We have used Pearson Correlation as a feature selection method. It is a measure of the linear correlation between two variables. The value ranges from -1 to +1: where -1 means total negative linear correlation, 0 means no linear correlation and 1 means total positive correlation. One major assumption we have made while choosing Pearson Correlation is the linear combination between any two variables. For selecting features, we run Pearson after training data is made, and if the correlation between the predictor feature and any other feature is > 0, then only we take them. Fig. 5 shows the kind of average accuracy of all the models based on two scenarios: with and without feature selection. On an overall level, we see that this kind of feature selection has no kind of added advantage. There is only

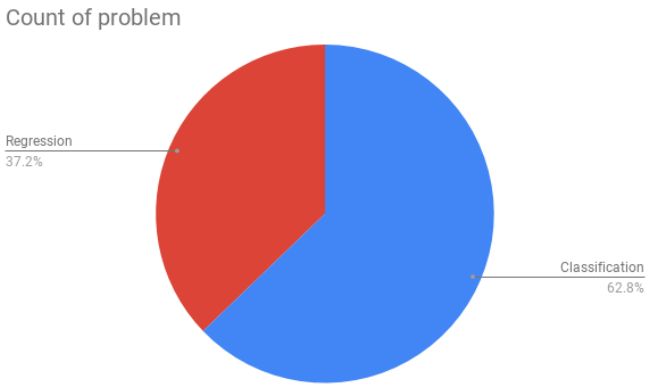

Figure 4: Percentage share by Type of Problem

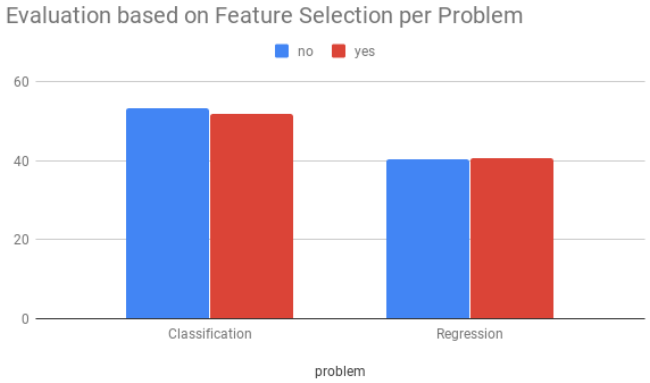

Figure 5: Evaluation Based on Problem Type (With/out Feature Selection)

a marginal improvement in overall accuracy of predicting regression variables when using feature selection.
Now, let us observe the charts on subset level. There are 5 bins created for the pipeline. Missing values ranging from 0-10, 10-30, 30-50, 50-70, 70-100. The reasoning for this has already been provided at the start. In the subset level, we see that there is a significant dip in performance when feature selection is used on the columns in subset 70-100 bin, both for regression and classification.

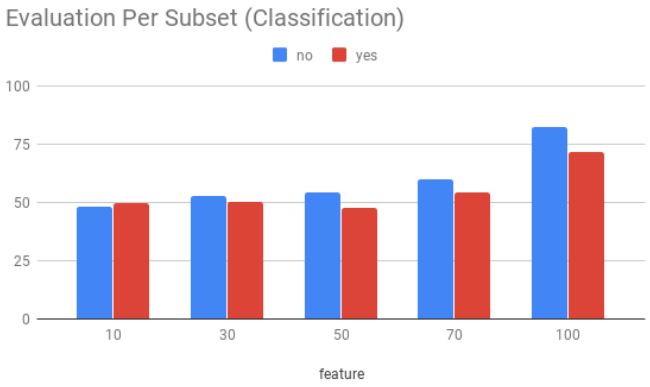

Figure 6: Classification - Evaluation Based on Subsets

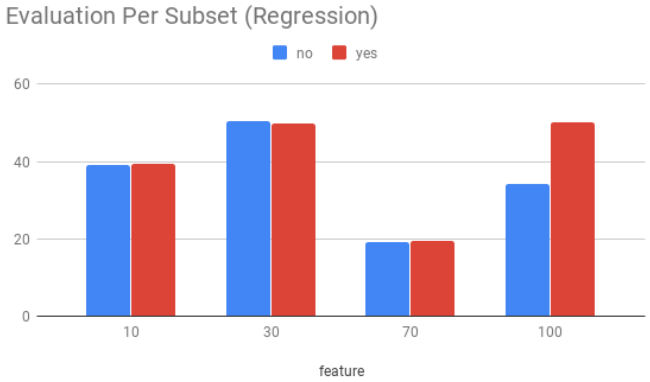

Figure 7: Regression - Evaluation Based on Subsets

Going into the feature level, we see that in Fig. 8, columns "omdimc3rt" and "omdimc3trt" have relatively lower loss. This is kind of intuitive because these columns are almost full and have very less missing values. Since, we are doing cross-validation technique, the data is being trained well for these columns.

If we look at columns from 10-30 bin Fig. 9, we see that feature selection has been helpful in lowering the loss for most of the columns: "anchoring1", "anchoring3", "anchoring4", "RAN002", "RAN003", "Ranch3", "Ranch4". "anchoring" columns are inter-related as they come from $Anchoring(Jacowitz\&Kahneman, 1995)$. In these scenarios, participants are asked to estimate the size or distance after first question had a number that was clearly too

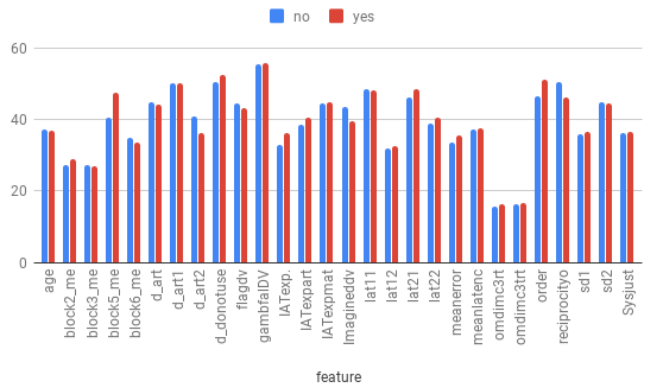

Figure 8: Regression - Subset 10 - Evaluation Per Feature

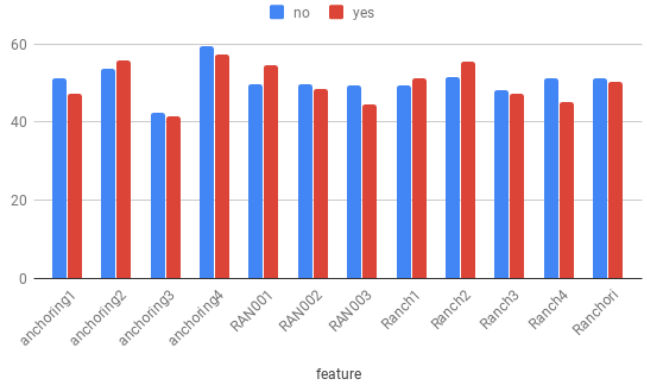

Figure 9: Regression - Subset 30 - Evaluation Per Feature

large or too small. Due, to this inter-dependency, feature selection has helped in removing the redundant variables for these columns. Similarly, according to the Code book, since "RAN" and "Ranch" columns relate to "anchoring", they show a similar trend.

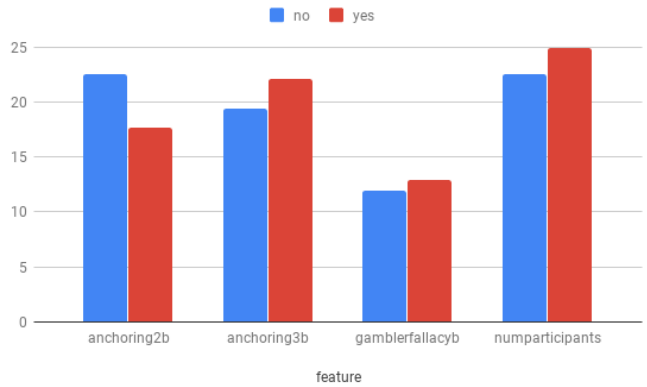

Figure 10: Regression - Subset 70 - Evaluation Per Feature

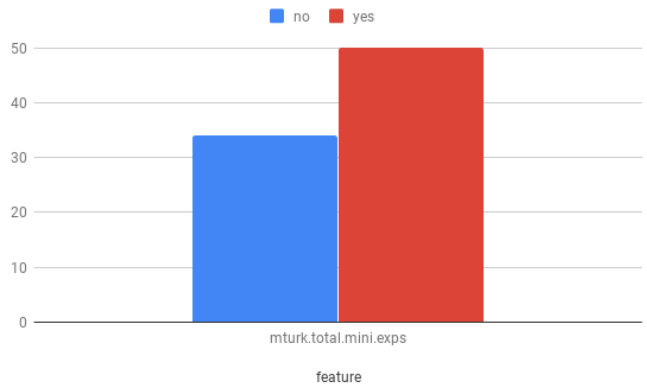

Figure 11: Regression - Subset 100 - Evaluation Per Feature

Fig. 10 and Fig. 11 show the trend of loss for the other subsets.

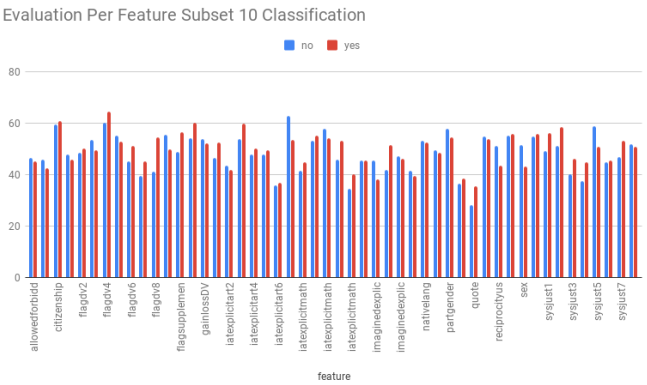

Figure 12: Classification - Subset 10 - Evaluation Per Feature

While classifying categorical features in the first subset, Fig. 12, the accuracy ranges from 25 - 60 %. The feature "quote" has the lowest accuracy of around 25%, but if you look into the data, there are 10 classes, so even 25% is a significant improvement from 10% random prediction model. Same thing can be said for the features at lower half of 12 are poorly accurate, they are much better than random prediction.

The features in subset 10-30 have shown good accuracy as can be seen from Fig. 13. "Ethnicity" column has 3 unique values, so if any value is chosen randomly, there will be 33% accuracy. However, the average of our models, give it an accuracy close to 60%. Similarly, "major" column has 13 unique values. That means if a value is chosen randomly, there will be 8%, but our average accuracy is around 40%, a big improvement.

One interesting trend which can be seen from Fig, 14, Fig. 15, Fig. 16, is that when we are imputing columns having more missing values, the accuracy on an overall level is increasing. This can be attributed to the fact that these columns might be dependent on columns which initially had missing values, but later on they got filled, and then worked as independent features while predicting these columns. It was for this reason, we established

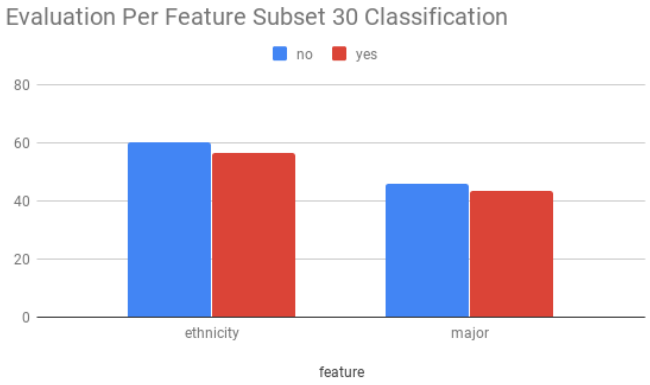

Figure 13: Classification - Subset 30 - Evaluation Per Feature

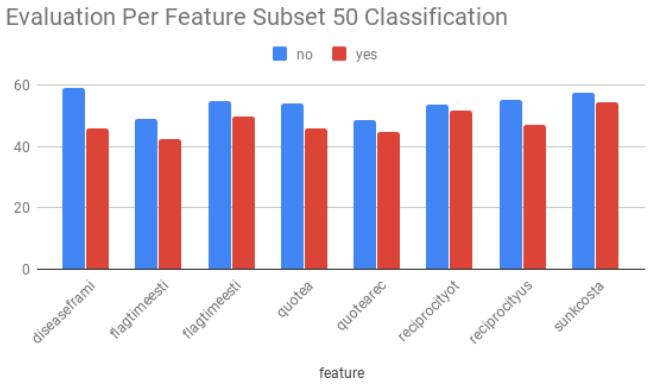

Figure 14: Classification - Subset 50 - Evaluation Per Feature

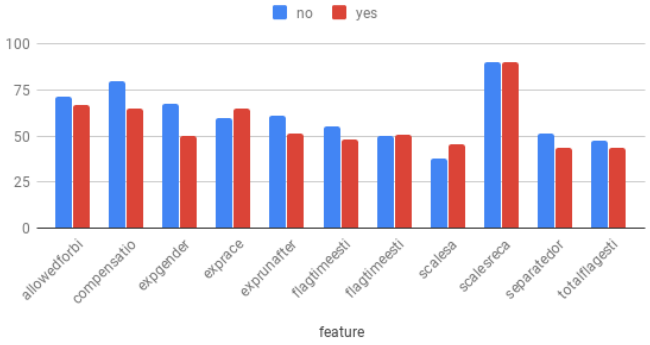

Figure 15: Classification - Subset 70 - Evaluation Per Feature

the pipeline as such.

Now, let us talk about the different models we used and how they fared. Looking at Fig. 17, we see that for classification, the model that worked best was K-Nearest Neighbor. It has been the best model in around 39% cases. KNN is a non-parametric model and does

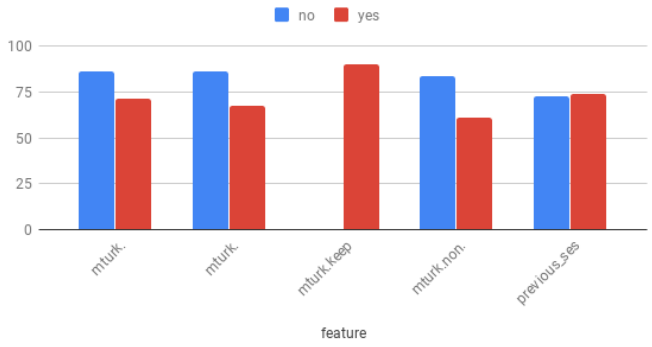

Figure 16: Classification - Subset 100 - Evaluation Per Feature

not make any assumption about the data. As a result, it can take any shape and is flexible in creating a decision boundary. This accuracy does come at a cost though, It is one of the slower classification algorithms used in our project.

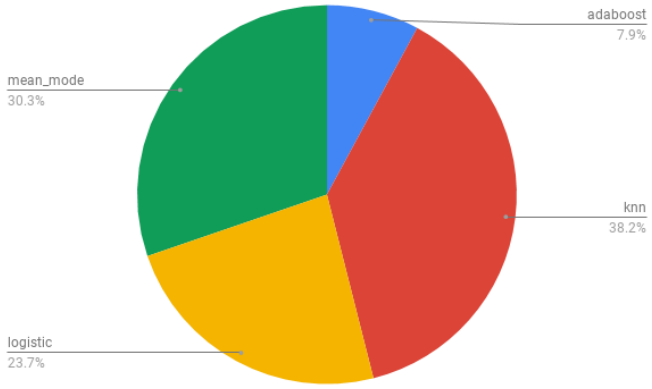

Figure 17: Model distribution for Classification

One thing to note here is that there is a low percentage given to Adaboost, this is due to the implementation of our Adaboost. We have only created it for binary classification, so it does not come into picture where we have more than 2 categories. If you look at Fig. 18, for cases when the predictor column in binary, Adaboost does really well and is the best model for 34% of the cases.

Looking at Fig. 19, we see that for regression, the model that worked best was Linear Regression. It has been the best model in around 36% cases.
Finally, let us look into the average accuracy or loss of the models when they were the best among others. According to Fig. 20, we see that again, linear regression has shown the minimum loss overall. It goes to say that the linear regression model has worked well for this dataset.

For classification, as thought so, Adaboost gave the best results when it was chosen as the best model. The ensemble nature of Adaboost is really good when it comes to classifying

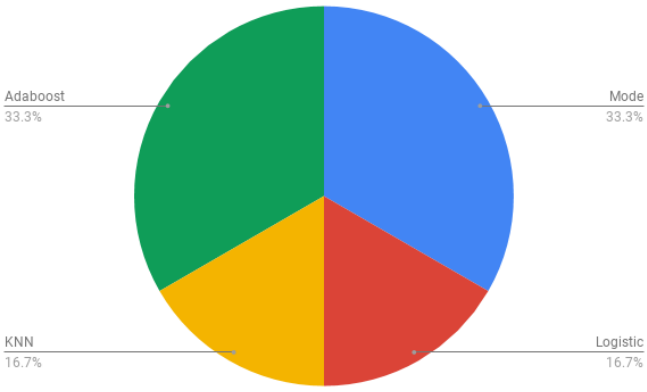

Figure 18: Binary Classification

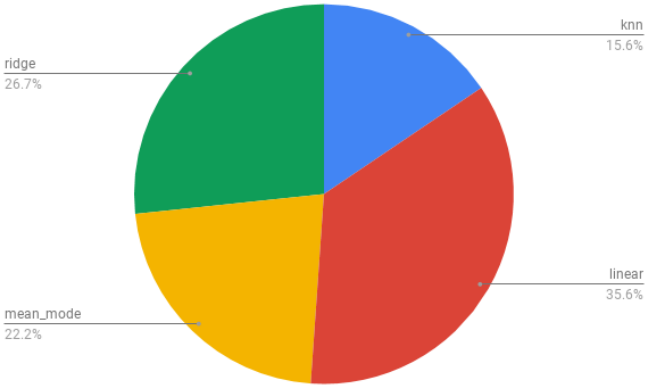

Figure 19: Model distribution for Regression

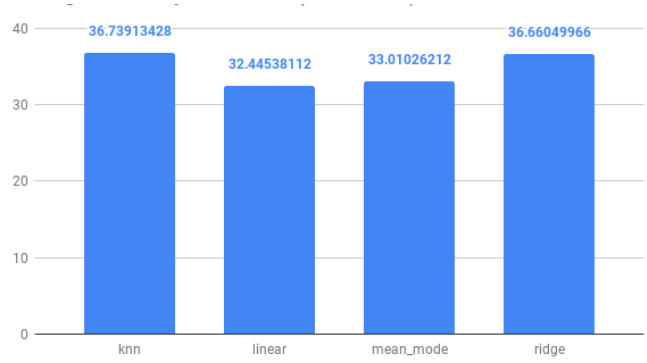

Figure 20: Average loss when model was best

features. KNN, although the best model overall, has a lower accuracy of 58%.
Another thing to look at is how beneficial an ensemble model came out to be. If you see Fig. 21, logistic regression has an accuracy of 59%, while Adaboost has 66%. Goes to say, our extra effort on implementing Adaboost was worth it.

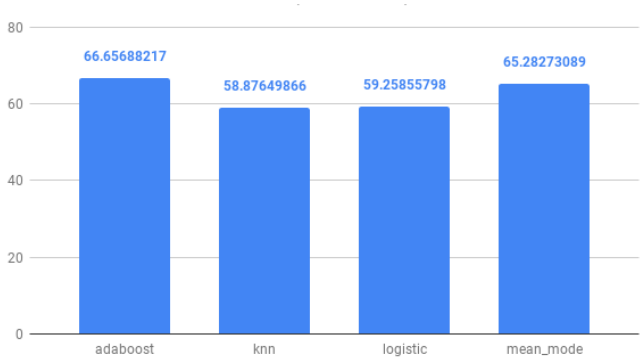

Figure 21: Average accuracy when model was best

2.5 Generate data

Unsupervised Learning and Generative Models: We also used some unsupervised algorithms to cluster the data and generative models to generate more synthetic data.

- K-Means Clustering: K-Means clustering is used by our Gaussian Mixture model (GMM) to calculate the initial estimates for the gaussian mixtures. Furthermore, we also tried to run Kmeans as a standalone algorithm to try and cluster our data into different groups.

- Gaussian Mixture Models: GMMs are based on the assumption that the data is generated from a combination of Gaussians and we need to estimate the mean and covariances of these Gaussian mixtures or distributions. The algorithm works in the following way:

  - We use KMeans to get the initial estimates for our Gaussian mixtures - the weights, means and covariances
  - The next step is the expectation step - the E Step. It calculates to get the posterior distributions of each data point for each Gaussian mixture. For e.g. if there are 4 Gaussian mixtures, then we have a probability value for each data point for all the mixtures. This value is also know as a responsibility value which conveys the information of how much does each mixture contribute in generating a data point.
  - The next step is the maximisation - the M Step. Based on the means and covariances calculated in the previous step, we calculate the new values (update the old values) in order to maximise the expected log likelihood of generating the data given our posterior probabilities.
  - Recurse through the E and the M step. This algorithm is thus known as the EM algorithm.

  Since our data has lots of variables, GMM requires large number of iterations to converge. However, increasing the iterations increased the training time a lot and so we

fixed the iterations at 100 and used GMM to generate more data for classification tasks. E.g. If the classification target contains 4 categories, then we need to separate the data into 4 Gaussian mixtures. Finally, we can sample new data based on the learned prior values - the means and the covariances of these Gaussian mixtures. Here are a few conclusions about this algorithm

  - With just 100 epochs the algorithm did not give us good results and infact decreased the prediction power of the algorithm. However, given more time for training we could have generated a better synthetic data.
  - The second point to note here is that GMM assumes that the data has been generated by a mixture of Gaussians. This is a very strong assumption to make and if a data is actually not coming from a Gaussian distribution, then it leads to very bad data generation and ultimately bad results.

  In general GMMs are a good way of generating data but require decent amount of training to converge for large amounts of data.
  usepackage

- Variational Auroencoder: We also decided to try a variational autoencoder(VAE) to get a good quality synthetic data than GMMs. The VAE is based on the idea of finding a latent distribution and then re-generating the data from this distribution. In the process, we learn the hidden latent distribution from which the original data was generated and can use it to sample new data. We coded up a VAE as a neural network with an encoder and a decoder. The encoder outputs the mean and the variance of the latent distribution and the decoder tries to regenerate the data using this latent distribution as its input. Our VAE has been implemented but the backproagation is still not working properly and given some time we would be able to implement it and test it on our data. We will attach the code for VAE with our report.

3 Bonus

For the bonus parts, we had gathered all the information we needed and the steps we would have taken to implement and incorporate NLP features into the model. However, we could not implement it due to time constraints. There are essentially 3 columns where there is abundance of text and a potential of some kind of text analysis. The columns are:

- user_agent: This column is already added to the model (as label encoded), and explained in the previous sections. We have tried to find meaning out of a long string which looks like garbage since it has terms of what kind of browser was used, the system and things like that. Going over the rows of this column, we observed that the system which was used for the test can be of importance (maybe not for prediction, but for finding some cool insights). On doing some basic analysis, we were able to see that most of the tests are done on Windows system (Fig. 22)

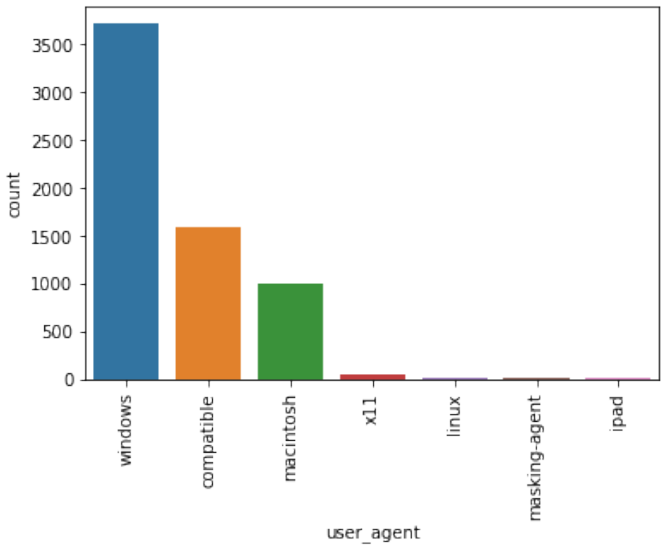

Figure 22: Most used OS for tests

- feedback: This column basically represents the feedback the participant was shown at the end of IAT questions. In these tests, participants completed four Implicit Association Tests (IATs) questions in random order, one of which measured associations of math and arts with positivity and negativity. This particular column can be a significant feature when predicting questions relating IAT columns like "while iatexplicitart", "iatexplicitmath". One major hurdle which we faced converting this into an NLP feature was the use of different languages. Since, we were not supposed to use external libraries, we had a hard time figuring out ways to clean the column, such that everything is in English language. This eventually ended up in not using this column. However, the way we had devised the extraction of information through NLP was as follows:
  - When observing the rows, we found a pattern which could be of importance to us for predicting certain column values.
  - The text usually is like - 'Your data suggest a moderate preference for Arts compared to Mathematics.'
  - Notice the positioning of "moderate", "preference", "arts" and "mathematics". This line basically means that the user preferred arts moderately over mathematics and now, if we look into the "iatexplicitart" columns, we see that moderate was labelled as 5.
  - With this linking, if "feedback" column can be converted to "art", "5", which basically will help in imputing "iatexplicitart" and "iatexplicitmath" columns. For this example, it would mean that the score in "iatexplicitart" columns should be on the higher side and "math" should be on the lower side. This extra information reduces the random guessing space by a lot, which can lead to more accurate results.

- imagineddescribe: This column is potentially open-ended. As per our knowledge, this column feeds the text the user writes when asked "describe what you just imagined for 1 minute". We believe a little background study of psychology would be required to make meaningful natural language extractions. One potential use case for this column can be to understand the meaning of what the user wrote, and relate it to the tests. Since, the user is in the middle of a test/survey, if the user is thinking about things totally different from the questions being asked in the test, there's a good possibility that the user is not interested in taking the survey and is filling random values into the questions. The similarity between the text from this column and a corpus of sentences (relevant to these lab tests, or text related to questions asked in the text like height of Mount Everest, Art, Math etc.) can be found by:

  - Converting the textual data into word embeddings, and then taking the average of the word embeddings of all words in the two sentences
  - After this, cosine angle can be calculated between the resulting embeddings. This is however a simple technique and can lead to high variance. Further, this technique does not associate the positioning of words.
  - Another technique which can be explored is using Encoders to lower the dimensionality of the text, and then using similarity methods to compute the difference.

  Further, sentiment of these texts can also be modelled using traditional sentiment analysis techniques. Finding a sentiment and having that as a feature can be important as it will also tell the current state of the user. If the user is predominantly happy, his/her chances of filling in the test answers genuinely would be more, than people who are sad or disinterested.

4 Acknowledgements

Throughout the project, we got to learn various new and existing techniques but from scratch! The project was thought-provoking and we got to think on some really cool techniques which we would like to research on, but could not due to the time constraints. Anyhow, we loved working on this final assignment mainly because it was totally different from conventional Machine Learning problems and we would like to thank Professor Cowan for giving us this opportunity.