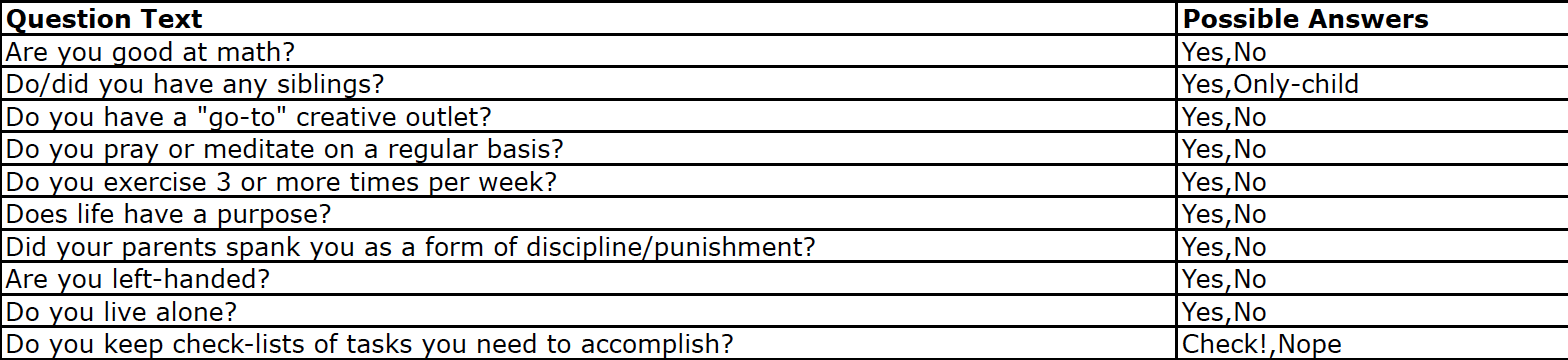
**Testing A Simple Neural Network Model with Voting Prediction Data**

In this era of great political divide, it is not hard to infer people’s political stance from what people say. For example, if a person expresses outspoken support for gun control, then I can infer that the person will likely vote for democrats. My inference here comes from my prior knowledge learned from past experiences: associating gun-control support with democrat-leaning political stance. Likewise, neural networks can learn to associate a person’s information with the person’s political leaning. Using simple deep neural networks (DNN), this project aimed to predict voting outcome on the basis of people’s responses on various questions. In doing so, I pitted the neural networks against other neural networks on the Kaggle challenge: <https://www.kaggle.com/c/can-we-predict-voting-outcomes#description>. Moreover, by streamlining the complex data provided by Kaggle as much as possible, the project tested the performance limitation of the simple DNN.

**Method**

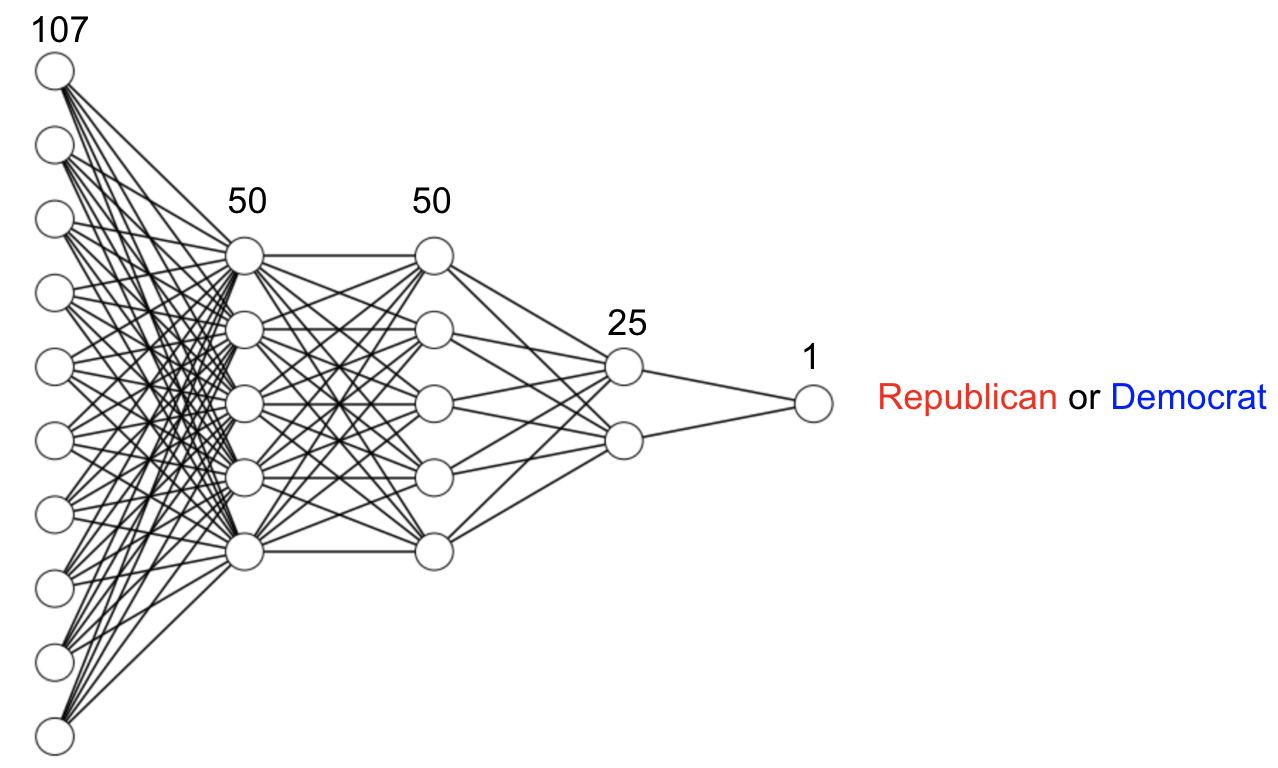
**Dataset**

The training and testing data from Kaggle came from Show of Hands, a mobile and web polling platform. For each participant, there were 107 questions to answer, including the political party the participant voted for. Of the 107 questions, 5 were demographic information, and 102 were questionnaire answers. The demographic information included gender, year of birth, income level, household status, and education level. All questionnaire questions were either subtly political or neutral (see Figure 1). For example, one question asked, “Do you personally own a gun?,” rather than explicitly asking whether a participant is in support of gun control. Participants were free to leave any question empty, but the data provided by Kaggle contained no empty answer for the political party people voted for. In total, the training set contained 5568 participant data. Of the 5568 data, 4467 data were used for training (about 80%), whereas the 1101 data were used for testing accuracy (validation).

*Figure 1*. All questions on the questionnaire required binary answers, including the political party the participants voted for.

**Neural Networks**

The present project used simple DNN built from Tensorflow’s “DNNClassifier.” The DNN model takes a 107-dimensional input (demographic and questionnaire answers) and outputs a voting prediction: Republican or Democrat. The model contains three hidden layers, two layers with 50 nodes and one layer with 25 nodes, (see Figure 2). The model used learning rate of 0.003 and L2 regularization strength of 0.001.



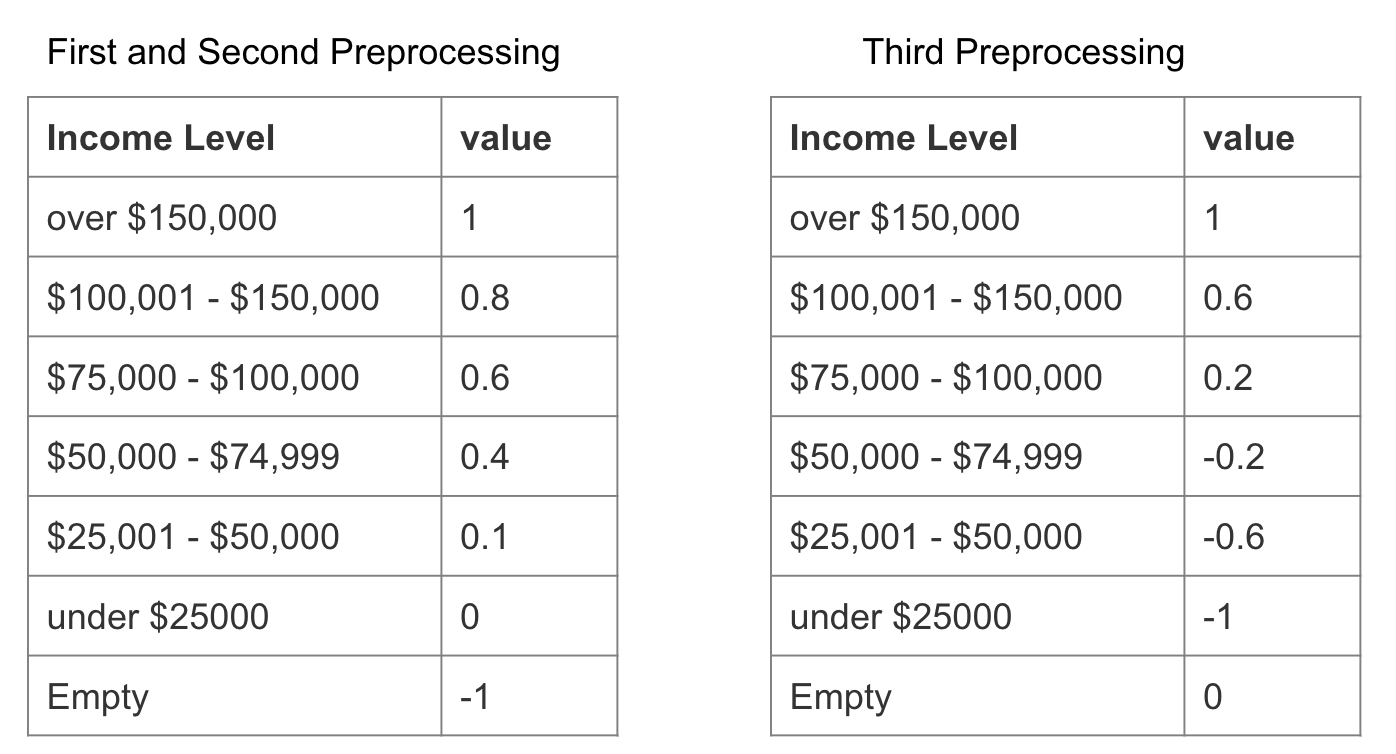
*Figure 2*. The figure shows a simplified DNN illustration of the model. The number of nodes in the figure is simplified and does not correspond to the actual number of nodes in the each layer. Each layer is a fully connected layer.

**Data Preprocessing and Results**

Except for year of birth (which was of numeric data type), all questionnaire answers were of string data type. Thus, since the DNN could only take a vector of numeric elements as an input, all string data had to be preprocessed into numeric data. Although the project used single DNN model, results were drastically different depending on data preprocessing.

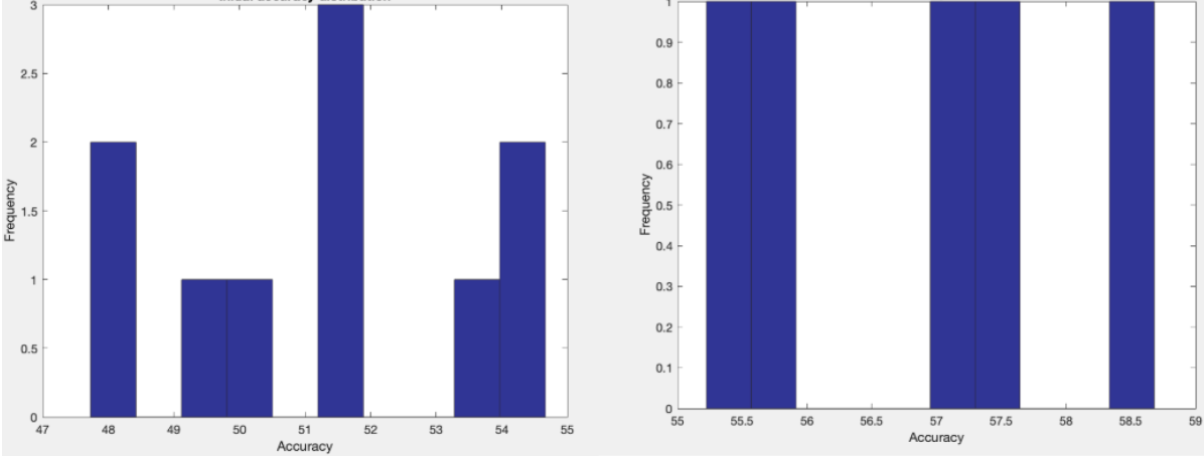
**The First Version of Preprocessing**

The first version used year of birth (YOB) without any further preprocessing, because YOB itself is of numeric data type. For the questions that required binary answers, such as “Mac or PC?”, binary answers were represented as 1 and 0. For example, the number 1 represented “yes” and the number 0 represented “no”. Some demographic questions required non-binary categorical answers. For example, for income level, there were 5 income categories between “under $25000” to “over $150000.” For such non-binary answers, arbitrary weights—ranged from 0 to 1—were assigned (see Figure 3).



*Figure 3*. The left table shows categories of income level and their corresponding weights used in the first version. The right table shows weights used in the second version.

With the first version of preprocessing, the accuracy was measured from 10 trainings (see figure 4A). The results showed that the performance of the model was no better than chance (*M* = 51.2, *SD* = 2.44).



*Figure 4A*. The figure on the left is the histogram of accuracy measured from 10 trainings with the first version of data preprocessing. The figure on the right is the histogram of accuracy measured from 10 trainings with the second version of data preprocessing

**The Second Version of Preprocessing**

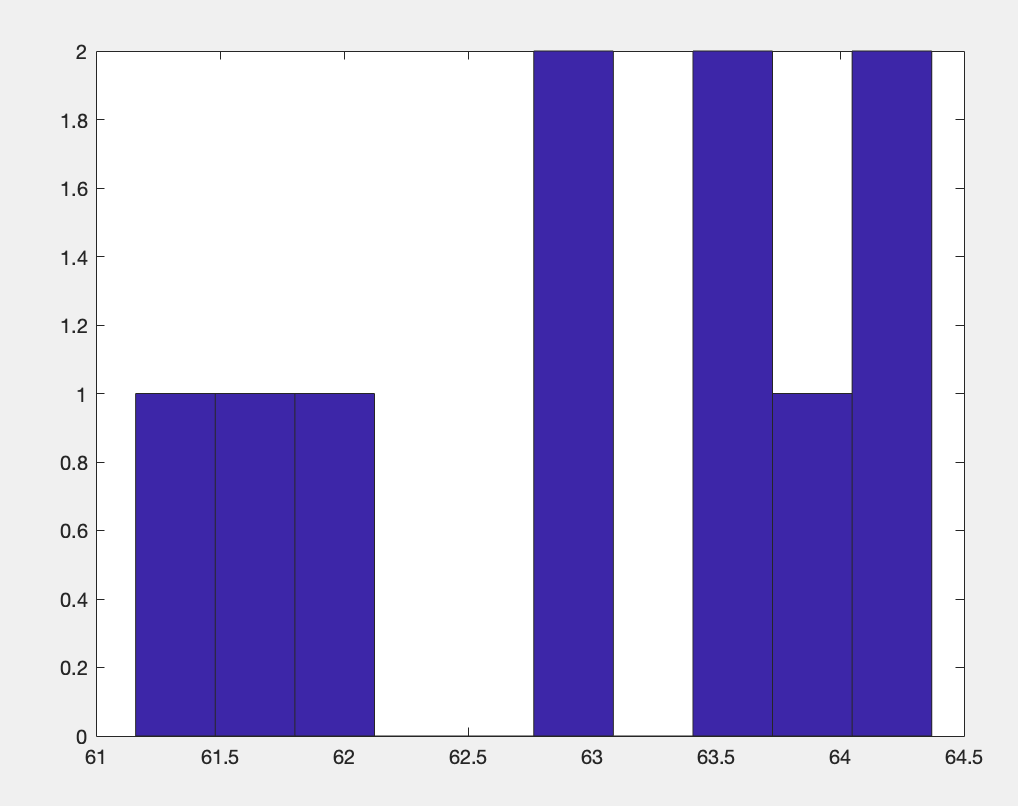
Modifying the data representation from the first version, the second version improved the data representation of YOB. In the first version, the range of YOB values (from 1900 to 2000) was far too big compared to the range of non-YOB values (from 0 to 1). Thus, YOB values were normalized such that all YOB values were between 0 and 1. For example, the YOB value 1934 was normalized to .34.

With the second version of preprocessing, the accuracy was measured from 10 trainings (see figure 4A). The accuracy results (*M* = 56.92, *SD* = 1.37) showed that the model using the second version of preprocessing achieved 5.72% accuracy improvement from model using the first version of preprocessing.

**The Third Version of Preprocessing**

Modifying data the representation from the second version, the third version improved the representation of empty answers. In the previous versions of preprocessing, empty answers (represented by -1) were represented as outliers, away from other possible values (from 0 to 1). However, indicative of uncertainty, empty answers should represent values within the possible value range. Thus, in the third version, the range of possible values was between -1 and 1, and the empty answers were represented as 0 (see *Figure 3*).

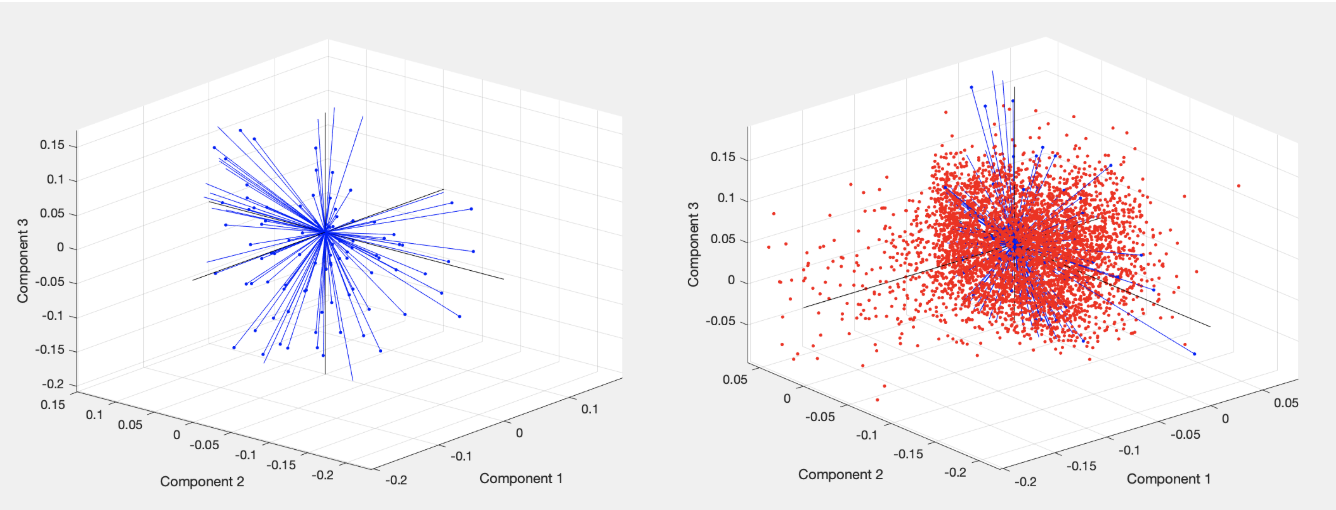
With the third version of preprocessing, the accuracy was measured from 10 trainings (see figure 4B). The accuracy results (*M* = 63.07, *SD* = 1.08). showed that the model using the third version of preprocessing achieved 6.15% accuracy improvement from model using the second version of preprocessing.



*Figure 4B*. The figure above is the histogram of accuracy measured from 10 trainings with the third version of data preprocessing.

**The Fourth Version of Preprocessing**

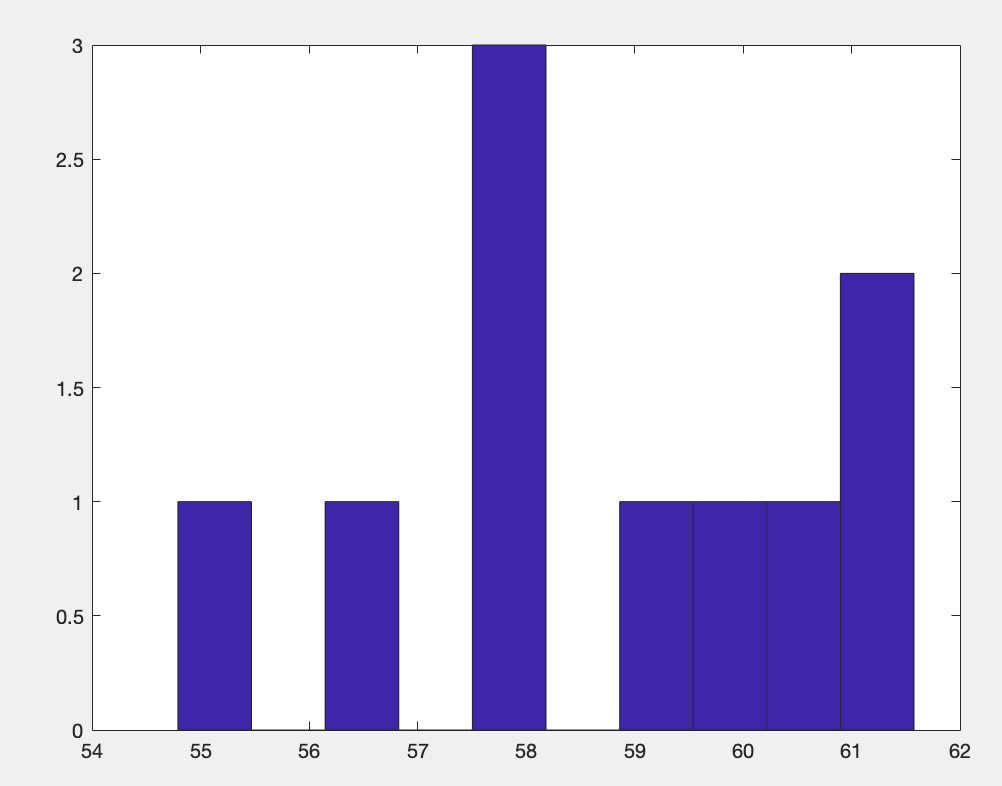
Modifying from the third version, the fourth version of preprocessing used the principal components (PC) of the input data to train the model. The PC were obtained from the principal component analysis (PCA) using the alternating least squares (ALS) algorithm. PCA simplified the inputs by reducing redundant and overlapping information in the 107 question conditions and provided a simplified, streamlined version of the inputs: the 107 PC (see figure 5A).



*Figure 5A*. The plot on the left shows the PC coefficients (blue lines) for 107 question conditions. The coefficients represent correlation between 107 question conditions. Conditions that are closer together are more correlated with each other. The red dots on the right plot are participants’ data represented in the principal component space.

Since the ALS algorithm automatically estimates the missing values in the data, all empty answer values (previously equal to 0) were made empty before PCA. After the PC were obtained from PCA, 5568 inputs were reconstructed from the PC transformation. Then, the model was trained with the reconstructed inputs.

With the fourth version of preprocessing, the accuracy was measured from 10 trainings (see figure 5B). The accuracy results (*M* = 58.6056, *SD* = 2.1789) showed that the model using the fourth version of preprocessing was 4.464 % less accurate than the third version.



*Figure 5B*. The figure above is the histogram of accuracy measured from 10 trainings with the fourth version of data preprocessing.

**Discussion**

With the DNN model having 3 hidden layers, the project achieved up to 63.07% prediction accuracy: Kaggle rank 158th~188th out of 2875 teams. Moreover, the streamlined training inputs reconstructed from PC could not achieve accuracy beyond 60%, so it seems reasonable that data preprocessing cannot improve the DNN model any further. Thus, about 63% seems to be the limitation of accuracy achievable by a simple DNN model. 94.5% of the teams could not achieve accuracy beyond 63%, and it might be due to the limitation of simple DNN models. Accordingly, rather than DNN with simple structures used in the present project, complex neural networks with multiple different layers (e.g. convolutional, recurrent, etc.) seem to be necessary for achieving high accuracy on the voting data. However, it may be still possible to improve accuracy of the DNN model by preprocessing data. The principal component variance from 107 question conditions showed that about 71 question conditions (*mean* = 71.78 from 10 results) contributed less than 1% to the dataset. Thus, excluding not-contributing conditions or adding bias towards contributing conditions may still improve accuracy on the simple DNN model.