**Overview  
*Models and techniques:***

We used two different models for the learning part of this project: Random Forest and Neural Network. For each model we performed hyperparameter tuning. For Random Forest, we turned the following hyperparameters: number of trees, max features, max depth, min leaf node size, and bootstrap. For Neural Network, we tuned the following hyperparameters: number of layers, layer type, number of hidden units, dropout probability, batch size, number of epochs. For data reduction, we deleted columns that were dominated (>98 %) by a single value. Additionally, we normalized data. We also performed reduction of input data dimensions through PCA. However, we abandoned the PCA-approach for final models. For scoring, we used 5-fold cross-validation, AUC score, and test accuracy.

***Timeline:***

Thursday: We planned the schedule for our project and selected models for the project.  
Friday: We started working on the data reduction part of the project and our hyperparameter turning for the Random Forest model.

Saturday: Finished parameter tuning and submitted initial results for tuned Neural Network and Random Forest model.

Sunday: Worked on finding the input features that had the highest influence on AUC score and test accuracy for Neural Network and Random Forest models. Work on initial report answers

Monday: Worked on final parameter tuning and changing input features for training and testing. Submitted final csv answer-files for part 1 and part 2.

Wednesday: Finalized and submitted report.

**Approach**

***Data processing and manipulation***We attempted several different manipulation techniques of the data. First, we decided not to include features that represented the ID-number, month of interview, and year of interview in our input data. This is because these numbers would not be representative of factors that would influence voter turnout. We then decided to remove any features in the data where answers were primarily the same. We created a function called data\_reduction that would find any columns where one answer-value had been chosen more that 98% for that particular feature question, using a few numpy functions. Applying these two data reduction techniques reduced our data from 382 columns to 266 columns. We tested whether the full input with 382 columns vs. the reduced input with 266 columns would give a higher AUC score with a default RandomForestClassifier from the sklearn module. We observed that the AUC score was slightly higher with the reduced data.

We then proceeded to test whether further reduction of input dimensions would be necessary to achieve a good test score. For this we created a function called Dimensionality\_reduction\_PCA. This function would fit and transform the input data to any number of dimensions/columns using the sklearn.decomposition.PCA function. At first, we calculated what the AUC score would be if reducing the input data to 10 dimensions. We observed that it would be rather low ~0.6 compared to 0.72 with the full data. We then calculated the AUC score of a standard RandomForestClassifier model for a range of different input dimensions from 50 to 266 with an interval of 10, where dimensions were reduced through our Dimensionality\_reduction\_PCA function (Fig. XX). However, we observed that reducing the dimensions through PCA did not improve our AUC score since performing a default RandomForestClassifier model with the full data yielded an AUC score of 0.72. Hence, we decided to perform all future model fits with the full reduced dataset of 266 columns.

Lastly, we performed column-wise normalization of all data points of the input data before performing any model fits and test accuracy/AUC calculations.



Figure XX: Number of dimensions/columns for input data, reduced by PCA, vs. the calculated AUC score for a default RandomForestClassifier model form sklearn.

***Details of models and techniques:***

The first model that we used was RandomForestClassifier from the sklearn module. We picked this model because it is a relatively stable non-linear approach to classification of large amounts of data. Additionally, several machine learning blog posts had performed classification and prediction of voter data through Random Forest, and they suggested that Random Forest models appear to do well on election prediction based on voter data. The advantages of Random Forest models are that they can perform non-linearly, they reduce overfitting the data and generally producing low test errors/high test accuracy, and they are generally fast. In addition, there are many hyperparameters such as node size, number of trees, max depth, max features etc. that can be tuned in order for the model to perform well. The disadvantages of this model is that the tuning of hyperparameters takes a significant amount of time. Additionally, one need to consider that the forest needs to be deep enough otherwise overfitting can still easily occur although Random Forest is based on a bagging approach.

We performed several hyperparameter tunings of the Random Forest model before fitting the final model. We scored the Random Forest models using 5-fold cross-validation and calculating CV accuracy and AUC score (read more under Model Selection) in a function called cross\_validating\_randomforest. Namely, we tuned the following parameters: n\_estimators (number of trees), max\_features (number of features to consider for best split), max\_depth (depth of forest), min\_samples\_leaf (minimum leaf node size), and bootstrap (whether or not a random subset of data is used for splitting) (Fig. XXX). The data in Fig. XXX were produced using the function perform\_randomforest\_sensitivity\_analysis.



Figure XXX: Hyperparameter values for number of trees, max features, min samples leaf, and max depth versus AUC score.

We initially wanted to run a model with n\_estimators = 1000, but we found that each model would take > 1 hour. Hence, we tried a different number of trees from 10 to 160. We found that number of trees > 110 would not significantly improve the AUC. We also tried a number of 500 trees which yielded a similar AUC score to 110 trees. Additionally, we observed that bootstrap = True would cause the Random Forest model to perform slightly better. Hence, the final hyperparameter values ended up being: n\_estimators = 110, max\_features = auto, max\_depth = 11, min\_samples\_leaf = 25, and bootstrap = True.

Once the final hyperparameters had been chosen, we performed the Random Forest model fit suing the .fit() method on the RandomForestClassifier object from sklearn. We then used the .predict\_proba() method in order to predict the probability of the positive class for the 2008 and 2012 test data.