Machine Learning & Data Mining
Miniproject 1

Caltech CS/CNS/EE 155
February 2019.

---

# 1 Introduction [20 points]

Group Members: Roberto Mercado, Hee Won (Michelle) Park

Team Name: Squirtle

Places: C1: 50, C2: 45

AUC Score: C1: 0.78656, C2: 0.77361

Division of labor:

Roberto: created neural net, worked on optimizing hyperparameters for random forest classifier, coded SVM, worked on report

Michelle: created neural net, worked on optimizing hyperparameters for random forest classifier, worked on report

## 2 Overview [20 points]

### Models and Techniques

Random Forest Classifier: We primarily focused on a Random Forest model which gave promising early results. After testing several configurations manually we made use of sklearn's Random Hyperparameter Grid and GridSearchCV classes to automate hyperparameter tuning.

Neural Network: Referencing the code from a previous set as a starting point, we designed a neural net with 3 layers.

SGDClassifier: We considered using sklearn's SGDClassifier for this classification problem and tried it briefly. Poor initial results combined with the relatively decent AUC scores for the other models kept us from pursuing this much further.

Out of the Ordinary: We did not pursue much in the way of exotic or complicated models as we decided that the extra time spent learning how to use and optimize models we were not familiar with would most likely not result in a proportional improvement in model performance. As such, we focused on the models listed above, although the built in hyperparameter optimization class was a new method which automatized an otherwise tedious task and provided built-in cross validation.

### Work Timeline

Sunday: Having already downloaded the data and created a collaborative Github repository, we meet up to start discussing possibilities for models we think will work well. We start by loading in and modifying the datasets. We then begin development of our neural net and test various sets of hyperparameters for a random forest classifier, which on first submission does decently well. We run into some issues with the neural net regarding quick convergence and poor performance and begin looking into solutions.

Monday: We continue to attempt to fix the neural net model while trying more combinations of hyperparameters for random forest. We also look into SGDClassifier but poor early performance leads us to go to office hours to see if we can get advice to improve on our models. Afterwords we decide to dedicate our time to maximizing random forest performance and look online for suggestions on how to best tune hyperparameters. We find out about sklearn's Random Hyperparameter Grid and Grid Search capabilities via a blog post and quickly set our machines to test some combinations of hyperparameters with the added benefit of automating cross validation. This process takes many hours and processing power so we let it run as we continue to discuss what else we could change with our models. One random search completes, having been given less combinations to try in an effort to get a rough estimate for a grid search that will take even longer. The second random search completes, but the model does much worse. We decide to run one more random search and one grid search based on the first random search. With both computers' CPUs running fully on these processes we leave our computers running for the night.

Tuesday: In the morning both hyperparameter optimizations are still running. The third random search finishes first and the resulting prediction improves our position on the leaderboard by a modest amount. Unfortunately the grid search does not finish in time for the competition, with just under 5 fits remaining.

# 3 Approach [20 points]

## Data Processing and Manipulation

Upon first inspecting the data we first went through the codebook to get a sense of what the features meant and to see if we could manipulate the data to our advantage. The main change we made to the input data for most cases was removing the first three columns. The first column was the id value, which had no significant meaning to us, as our submission file was formatted so that id would begin at 0. The next two columns were also unnecessary for learning, as they represented the month and year of the survey, which was the same across all data points and thus offered no inherent information. We also created a copy of the dataset which had normalized columns so that each value's absolute value was less than or equal to 1 for models sensitive to large fluctuations in orders of magnitude across features (neural nets and SVMs). When modeling using the neural net, we also one-hot encoded the target vector as per the homework sets using keras' to_categorical utility so that we could use categorical cross-entropy as our loss function.

## Models and Techniques

Random Forest Classifier: The decision tree model felt like an obvious fit for this data. For one, it would not be necessary to normalize the data as the decisions for splits are based on orders of the features across the data points. Furthermore, since decision trees take the best split across the features, columns that were rarely filled out, e.g. most values were -1, or otherwise indicated the same information for most data points, would be naturally ignored by the classifier. Performance was promising early on and initially, we tested manually in an attempt to tune hyperparameters. The most useful tools regarding this model were sklearn's classes RandomForestClassifier(), RandomizedSearchCV(), and GridSearchCV(). RandomForestClassifier() created a base model on which we would tune our hyperparameters. Then RandomizedSearchCV() was utilized to test random hyperparameter combinations. The parameters we tested on were: n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, bootstrap, and criterion. We ran the random search 3 times, first with 10 iterations (30 fits), then with 50 iterations (150 fits) and finally with 20 iterations (60 fits). The random search with 50 iterations actually performed significantly worse than the first search with 10 iterations, potentially because the step sizes between the parameter values were too large. Using the general idea we got from the first run of RandomizedSearchCV(), we then utilized GridSearchCV() to perform an exhaustive grid search over specified parameters. This unfortunately did not finish in time, despite running it overnight, as there were 144 fits to test. We also ran GridSearchCV() based on the third run of RandomizedSearchCV(), but this also did not finish on time. We saw that our best-performing classifier had the following parameters: bootstrap = True, max_depth = 50, max_features = None, min_samples_leaf = 3, min_samples_split = 5, n_estimators = 1920, criterion = entropy. Although computationally intense and time-consuming, these classes provided a simple interface to test a variety of parameter permutations with cross validation and return an objective measure of which combination was performing the best. This could even be supplied with sklearn's built-in roc_auc_score function so it would return the model which performed best based on the metric of the competition.

Neural Network: We decided to test out neural networks because there was a high volume of data. First, we shuffled our normalized data and then one-hot encoded the target vector using keras' to_categorical utility. We first tried implementing a neural net with 3 hidden layers, 3000 hidden units, categorical crossentropy as our loss function, RMSprop as the optimizer, and the accuracy metric. However, we saw

that it did not perform very well and quickly converged to about 0.75 accuracy. We tried changing the number of hidden layers, the number of hidden units, optimizer, dropout rates, but our neural net always converged to about 0.75. Seeing that the neural net has about 3/4 accuracy, it is evident that it performs no better than a naive classifier that always predicts 0, as 74.46% of the training data have the label 0. Later, we switched our metric to auroc but still saw no change until we noticed a bug in the code we used to normalize the data. Data normalization is particularly important for models relying on weights connected to features as differences in relative magnitudes between features can artificially raise the importance of certain features. Unless a given feature or features are known prior to be sufficiently more important to warrant being an order of magnitude greater than other features, normalization is essential for learning.

Support Vector Machine: We tried implementing SVMs because they can efficiently perform non-linear classification and because we were dealing with a large dataset with a large number of features. To run the SVM, we simply utilized sklearn's SGDClassifier(). For the parameters of this classifier, we set loss = "hinge", max_iter = 1000, tol = 1e-3, and shuffle = True. We trained the SVM on the first 40,000 datapoints and then validated on the remaining data. Our SVM did not perform well, returning a roc_auc score of about 60%. Although this model did not perform well it still serves as an example of an important concept when learning with loss-based gradient descent. This model provides an argument to allow for the shuffling of data, which is important to avoid stochastic gradient descent from falling into a local minimum. If the data is shuffled then across multiple iterations of fitting, the likelihood that the performance of the model is consistently in a local minima rather than the global minimum decreases as we start on different parts of the loss function each time.

# 4 Model Selection [20 points]

## Scoring

Early on we used accuracy as our optimization objective, leading to mixed understanding of comparative performance of models when submitted and confusing results with respect to the neural net. However, once we realized that the competition utilized the ROC metric, we utilized sklearn's built in roc_auc scorer so that we could compare performance in the same metric as the scoreboard, and as a result our submissions began to improve. We saw that the random forest model scored the best initially and thus chose to focus on improving the random forest model instead of neural nets or SVMs. The first random forest model we tried (without hyper-parameterization of any sort) returned a score of 0.78411 on the public leaderboard. Our first attempt at hyper-parameterization returned a score of 0.78923. The second and third attempts returned 0.64072 and 0.79373, respectively.

## Validation and Test

We decided to implement some sort of validation technique because we did not have access to the labels of the testing data, but still needed to minimize overfitting somehow. We used a roughly 2-1 train-validation split for our manually tested models, such as our SVM and neural net. This was meant to give us an idea of the generalizability of our models and determine if and when they were overfitting. We were considering coding up our own method of testing and comparing various hyperparameter configurations for our models but upon further research discovered that sklearn had this capability built in. Not only were we able to use roc_auc as a comparative metric but also were able to quickly set up comparative testing across hyperparameter ranges with cross-validation built in. On the fits that were performed using the random hyperparameter variation and exhaustive grid search, we provided the entire set and opted for 3-fold cross validation. This cross-validation was built in to the class and the number was chosen as a compromise between validation efforts and computational time taken. Thus, for the neural nets and SVMs, we used about 2/3 of the dataset to train our models and then found how well it did on the remaining 1/3 as validation data. The neural net showed promise in training auroc score but validating on the remaining data suggested a fair amount of overfitting, which dropout layers partially alleviated but not to a satisfactory degree. The best performing model by a good margin was the random forest classifier, with each hyperparameter-optimized iteration performing better than the last.

| rank_test_score | n_estimators | min_samples_split | min_samples_leaf | max_features | max_depth | bootstrap | mean_test_score | mean_train_score |
|---|---|---|---|---|---|---|---|---|
| 7 | 1400 | 2 | 1 | auto | 90 | TRUE | 0.778266765 | 1 |
| 6 | 1400 | 10 | 1 | sqrt | 80 | TRUE | 0.778268111 | 0.999076069 |
| 1 | 1600 | 2 | 2 | auto | | TRUE | 0.779073339 | 0.999922682 |
| 3 | 600 | 5 | 2 | auto | 110 | TRUE | 0.77842056 | 0.999792569 |
| 10 | 1000 | 2 | 4 | auto | 60 | TRUE | 0.778057452 | 0.986798121 |
| 9 | 600 | 2 | 4 | auto | 110 | FALSE | 0.778093264 | 0.999552188 |
| 4 | 1600 | 5 | 1 | auto | 60 | TRUE | 0.778275473 | 0.999999291 |
| 5 | 1800 | 5 | 2 | sqrt | 90 | FALSE | 0.778272748 | 0.999999985 |
| 8 | 1200 | 5 | 1 | sqrt | | TRUE | 0.778236595 | 0.99999905 |
| 2 | 1400 | 10 | 4 | sqrt | 70 | FALSE | 0.778431512 | 0.999210402 |

Figure 1: Results of the first RandomSearchCV() run. Chart shows mean test score, mean training score, and a ranking of the test scores for random parameter combinations.

| rank_test_score | n_estimators | min_samples_split | min_samples_leaf | max_features | max_depth | bootstrap | mean_test_score | mean_train_score |
|---|---|---|---|---|---|---|---|---|
| 2 | 1600 | 2 | 1 | auto | | TRUE | 0.778727945 | 1 |
| 3 | 1700 | 2 | 1 | auto | | TRUE | 0.778604234 | 1 |
| 7 | 1800 | 2 | 1 | auto | | TRUE | 0.778418668 | 1 |
| 11 | 1600 | 3 | 1 | auto | | TRUE | 0.778279493 | 0.999992268 |
| 4 | 1700 | 3 | 1 | auto | | TRUE | 0.778511451 | 0.999984536 |
| 1 | 1800 | 3 | 1 | auto | | TRUE | 0.778743409 | 1 |
| 8 | 1600 | 4 | 1 | auto | | TRUE | 0.778372276 | 0.998902069 |
| 5 | 1700 | 4 | 1 | auto | | TRUE | 0.778449596 | 0.998878873 |
| 10 | 1800 | 4 | 1 | auto | | TRUE | 0.778310421 | 0.9989098 |
| 9 | 1600 | 5 | 1 | auto | | TRUE | 0.778356813 | 0.994301575 |
| 5 | 1700 | 5 | 1 | auto | | TRUE | 0.778449596 | 0.994077351 |
| 12 | 1800 | 5 | 1 | auto | | TRUE | 0.778202174 | 0.994239721 |
| 15 | 1600 | 2 | 2 | auto | | TRUE | 0.777923825 | 0.977538777 |
| 18 | 1800 | 2 | 2 | auto | | TRUE | 0.777831042 | 0.977561971 |
| 20 | 1600 | 3 | 2 | auto | | TRUE | 0.777614548 | 0.977732075 |
| 16 | 1700 | 3 | 2 | auto | | TRUE | 0.777892897 | 0.977654757 |
| 14 | 1800 | 3 | 2 | auto | | TRUE | 0.777954753 | 0.977306819 |
| 17 | 1600 | 4 | 2 | auto | | TRUE | 0.777877434 | 0.97771661 |
| 19 | 1700 | 4 | 2 | auto | | TRUE | 0.777800114 | 0.977616096 |
| 13 | 1600 | 5 | 2 | auto | | TRUE | 0.778155783 | 0.970819737 |

Figure 2: GridSearchCV() results based on first RandomSearchCV() run. Chart shows mean test score, mean training score, and a ranking of the test scores for gridsearch parameter combinations.

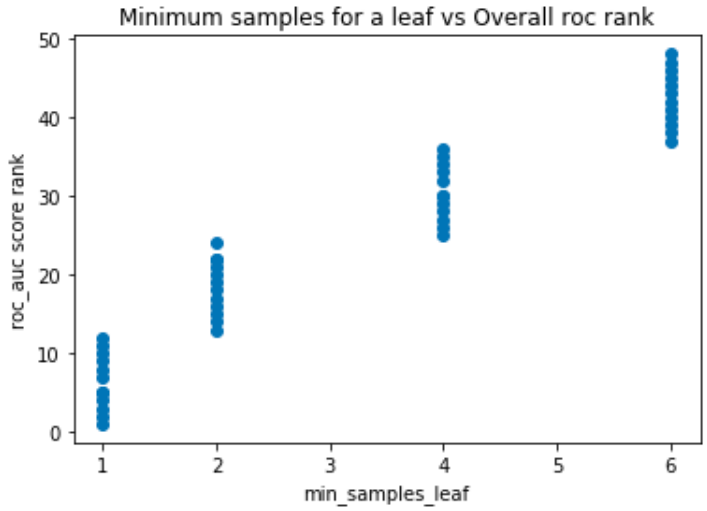


Figure 3: Performance results from GridSearchCV(). Plot indicates that a lower min_samples_leaf parameter tends to increase roc_auc performance across other varied parameters, suggesting its importance in optimizing the model.

# 5 Conclusion [20 points]

## Insights

1) The features' importance in descending order was: PEAFEVER, PERRP,PXSCHLVL, HETELAVL, HUTYPB, PESCHLVL, PWORWGT, PWLGWGT, PXGR6COR, GEREG. Some of these fields were not explained very in depth in the code book but some indicators, such as veteran status, family status, education, geographic region, and weights relating to financial status, made sense as to why they would heavily impact the decision to vote.
2) We use AUC as our Kaggle metric because it provides a method for accounting for Type I and Type II errors at a variety of thresholds. This makes perfect sense for the given problem as for two models if just accuracy was the metric and both performed similarly but had statistically significantly different AUC scores we would be losing out on statistically important information. AUC allows for distinguishing how much better a given model is at predicting either class over another model in a way that conveys information important to a statistician who would naturally be concerned about Type errors when predicting voter behavior.
3) Many of the sklearn models that were used have an option of n_jobs which allows for multiple CPUs to be assigned to the task such as the fitting and prediction methods. Additionally, with more computational power we could run multiple instances of the hyperparameter tuning optimizer across varying ranges for certain key hyperparameters and use those to further refine what we exhaustively search over.
4) We learned what it was like to work with real-life data and how to approach solving a real problem. We realized that there is no one right answer and that various methods can all work, each with their own pros and cons. We also learned to be very meticulous and to make sure no small mistakes are being made, as a small bug can cause a model to fail and not learn. For example, we realized that our neural net wasn't learning anything and would constantly converge to the same value because our data wasn't normalized properly. Additionally, we realized the importance of researching for efficient ways of doing something. For example, we had intended to implement our own algorithm to optimize hyperparameters and apply k-fold cross validation. However, with some research, we found that there were existing classes in sklearn that already did exactly what we had wanted, and thus saved us from unncessarily wasting time.
5) One important lesson learned is the time it takes to develop models. With large data sets comparing even a few variations of settings such as hyperparameters or different ways of manipulating the data can take a very long time. As such it is important to narrow down what to focus on in order to avoid wasted time pursuing models with less than satisfactory results. Additionally, we now have a better understanding of how to select a model depending on the question we are trying to solve. Instead of just simply knowing how to set up a neural net or how to run a SVM, we now know how to apply them in real-life problems and on real data.

## Challenges

If we could do this project again, we would probably start earlier, as we did not realize the hyperparameter optimization would take so long. The machines running this optimization were laptops and even when connected to a power outlet, took a significant amount of time to fit each model to the over 60,000 data points and run cross validation. It was challenging to improve our random forest with this method

of hyperparameter optimization, as it would take several hours to receive any feedback on our estimations and figure out if we were taking a step closer in the right direction or not. With our neural net, we encountered a problem of it converging and not learning anything. We did realize later, however, that this was caused by improper normalization of data in the beginning.

## Concluding Remarks

Unfortunately the last hyperparameter optimization run did not finish in time for the contest as it was very computationally intense. It completed its processing after the contest was closed and a late submission was made but did not result in an improvement, suggesting the ranges used were sub-optimal or more processing on the data should have been conducted to achieve a better result. Furthermore, after the bug with the data normalization was fixed we trained a neural net which did see improving roc_auc scores over the epochs but this model also fared poorly on the competition data.

## References

The hyperparameter optimization was found at the following link: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

The code for allowing roc_auc as a metric for a Keras neural net was found on Stack Overflow at the following link: https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras