COMP 4448

19 November 2021

Executive Summary

This analysis explored how well different machine learning algorithms classify to which political party House of Representatives members belong based on their voting records. Specifically, the research question explored was "Between Random Forest, Logistic Regression, XGBoost, Neural Network MLP (Multi-Layer Perceptron), and Support Vector Classifier (SVC), which supervised learning algorithm most accurately classifies Republicans and Democrats into the correct political party based on 16 Congressional votes from the 1984 House of Representatives?" The data consisted of 435 records, one for each House member of the 1984 Congress. The features were voting records for 16 different issues. After cleaning the data, the performance of the five different machine learning algorithms was tested on the data to see which could most accurately classify members based on the voting records. The sklearn implementation of each algorithm was instantiated, fit to the data, and then analyzed for performance. The algorithms' hyperparameters were then tuned and the models were re-run to obtain the best possible predictions. All the algorithms performed quite well on the data. Neural Network MLP was the top performer with an accuracy of 98% and the other metrics, precision, recall and F1 score were close behind with a tie at 97%. The next three algorithms, Logistic Regression, XGBoost Classification, and Support Vector Classification were tied at 97% accuracy and 96% F1 score with varying Precision and Recall. Random Forest Classification was last with 95% accuracy and a 93% F1 score.

Research Question

This project explored a variety of machine learning models to see which most accurately is able to make a prediction on the data. The goal is to classify to which political party a member of the House of Representatives belongs to based on 16 votes from the 1984 Congress. Between Random Forest, Logistic Regression, XGBoost, Neural Network MLP (Multi-Layer Perceptron), and Support Vector Classifier (SVC), which supervised learning algorithm most accurately classifies Republicans and Democrats into the correct political party based on 16 Congressional votes from the 1984 House of Representatives?

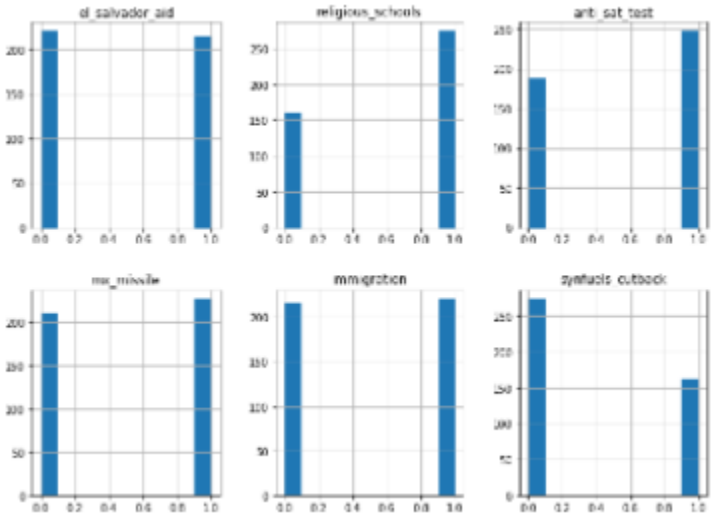
Data

The data, which contained 16 different issue votes and the political party of the member, was obtained from the 1984 House of Representatives voting records. It was retrieved from the UCI Machine Learning Repository, at https://archive.ics.uci.edu/ml/datasets/congressional+voting+records. There were 435 records, one for each member of the House. The target, political party, was imbalanced with 267 Democrats and 168 Republicans. The 16 votes were in binary format; either 'Yes' or 'No.' There were several missing values in the data. There are several reasons why a vote was classified as an unknown. This means the member either voted present, voted present to avoid conflict of interest, or did not vote or otherwise make a position known. A vote on South Africa (export-administration-act-south-africa) had the most missing values, with 104 unknowns, which equated to nearly 24% of the values were missing. Imputing that many values could skew the analysis, so that column was dropped. The next highest missing number in a vote was on water sharing (water-project-cost-sharing) and had 48 unknowns (11% of the data). All columns except the target class had some missing values. With only 435 instances in the dataset, there was a need for an imputation strategy to keep as much data as possible. With the differing reasons that members missed votes, including not wanting to make a position known, it seemed probable that the data was Not Missing at Random (NMAR). To test this, a table was created with the proportion of Democrats and Republicans that missed each vote. A t-test was performed and the result was that the means were not found to be statistically different. Apparently when it was advantageous for one political party to not make a position known, that also held true for the other party. While the data was not NMAR relative to the target class, it was Missing at Random (MAR). The data was not missing across all observations, but within sub-samples of the data. Based on this, the imputation strategy chosen was to assign the member's party majority vote to the missing value. For example, if a Democratic member had a missing value for the 'el_salvador_aid' vote, the value was filled in with a 0 (no vote). Only 55 Democrats out of 267 voted for the El Salvador Aid bill, so the majority of Democratic members voted no. The votes were rarely (only in two cases) party line votes, but while there was some cross voting on issues, it seemed likely the strategy would not introduce bias.

The data was split using sklearn's train_test_split module. This analysis used a variety of models so the train/test split was chosen being mindful of the data requirements of each of the models. With only 435 instances, a 75% train and 25% split seemed reasonable. It provided as much data as possible for the algorithm with a large data requirement, the neural network. While the other algorithms were not as sensitive to the size of the data, this split still gave a robust test set.

Exploratory Data Analysis

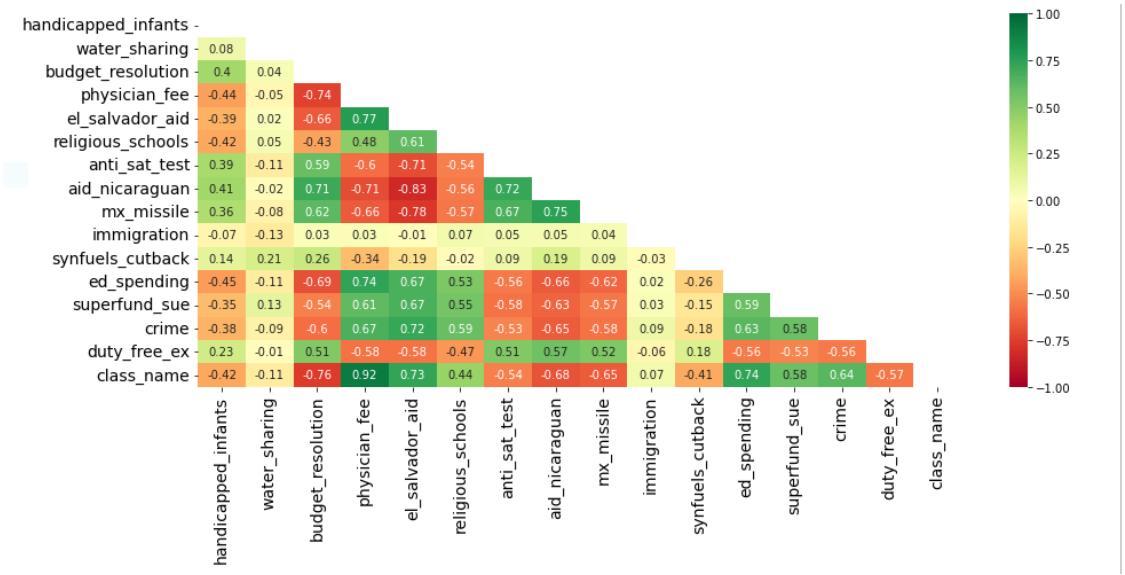
A histogram of the votes demonstrated that there was a mixture of 'yes' and 'no' votes and they are not straight party line votes. If the Representatives always voted their party position, this would be a very trivial problem. However, since there are 100 more Democrats than Republicans, the fact that many votes are similarly split between 'yes' and 'no' establishes that there is voting across party lines.



Sample of Histograms

A heatmap of the correlation matrix, which shows how closely the variables are related to each other, gives several insights. The dark green and dark red show strong positive (green) and strong negative (red) correlations. There are very few strongly correlated variables. The variable

'class_name' indicates whether the member is a Democrat or Republican and is the target prediction variable. There are two votes that are strongly correlated with the target variable. These are Budget Resolution, which is highly negatively correlated which means that the Democrats mostly voted for the issue and the mostly Republicans did not. This is because Democrats were encoded with a 0, so 'yes' votes, encoded with a 1, would show a negative correlation. Conversely, the Physician Fee Freeze vote was strongly positively correlated, which means that Republicans mostly voted for the issue and Democrats mostly did not. Republicans were encoded with a 1, so a 'yes' vote, also encoded as a 1, would show a strong positive correlation. There were also a few votes that were highly correlated with each other. The vote about El Salvador aid was strongly negatively correlated with the vote on MX Missiles, and also, interestingly, negatively correlated with the vote on Nicaraguan aid.



Correlation Heatmap

Models

The prediction models tested were Logistic Regression, Random Forest Classification, Support Vector Classification, Neural Network Classification, and XGBoost Classification.

Logistic Regression is a predictive analysis for data with a categorical outcome. Since this data has two possible outcomes, the Binary Logistic Regression is appropriate. The assumptions of logistic regression are that the dependent variable is discrete, the data has no

outliers, and there is no high correlation between the predictors. The standard for high correlation is 90%. (What is Logistic Regression? - Statistics Solutions, 2021) While there is one correlation on the heatmap at 92%, that is between a predictor and the target. Therefore, all of these assumptions are met. The sklearn implementation LogisticRegression was used. It followed the standard sklearn API of creating a classifier, fitting the X and y training data, and then generating predictions of the test set. Additionally, predictions were created on the training set to check for overfitting. The logistic regression performed very well. It had an accuracy score on the test set of 97%. The training data showed a very similar 97.5% which indicates that the model did not overfit on the training data. Next, the hyperparameters were tuned to ensure the best possible result. The ‘penalty’ parameter was tested with 'none', 'l1', 'l2', and 'elasticnet'. These are different methods to shrink the coefficients. The parameter 'solver' was tested with 'lbfgs', 'sag', 'saga', and 'newton-cg'. The solver is a mathematical approach to solve the logistic problem. ‘Newton-cg’ minimizes the quadratic cost function, and does so in an optimized way. ‘Lbfgs’ stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm. It is similar to Newton-cg but uses approximation in the gradients. ‘SAG’ stands for Stochastic Average Gradient. It incorporates a memory of previous gradient values which achieves a faster convergence rate than other Stochastic Gradient models. Finally, ‘Saga’ is a similar to SAG and also supports an L1 penalty. This makes it suitable sparse matrices which are common in large datasets. (definitions, 2021) Not all penalties and solvers can be paired together. Therefore, when GridSearchCV tried pair unlike types, errors were generated, as expected. After tuning, the best hyperparameters were L2 and LBFGS, which were what the algorithm was originally run with. A characteristic of logistic regression is that you can derive which features are the most important for the prediction. This analysis showed that Budget Resolution and Physician Fee Freeze were the most influential features. This is not surprising since those were the two-party line votes.

Random Forest is an ensemble algorithm which creates a number of decision trees classifiers on the data. It randomizes the number of instances and features used to create each tree in order to create decision trees with low correlation. Then, it aggregates the decisions from each tree and takes the majority vote as the prediction. (Pedregosa, 2011) Random Forest Classifiers are very permissive with the types of data it can use. The algorithm does not have formal distribution assumptions, it is non-parametric, and can handle ordinal and nominal categorical data for predictions. (Richmond, 2021) The sklearn implementation

RandomForestClassifier was used with the parameter max_depth equal to 4. If max_depth is not specified, then the tree will be expanded until all leaves are pure which could lead to overfitting. The algorithm was fit to the data and the predictions were generated. The accuracy score on the test set was 95% and the test set was 97%. This shows excellent performance and no sign of overfitting. To see if the test performance could be improved, the max_depth and n_estimators hyperparameters were tuned. As previously stated, max_depth is the number of tree levels. The n_estimators parameter is the number of trees in the forest. Tuning showed the optimal tree depth to be 3 and the best number of estimators to be 50. The model was re-run with the tuned parameters, but achieved the same accuracy of 95%. The model's Precision was 93%, Recall was 95% and F1 Score was 94%.

Support Vector Classification (SVC) is an implementation for a Support Vector Machine for problems with discrete outcomes. The idea behind a Support Vector Machine is that it uses a hyperplane to separate (classify) the data points. It uses support vectors, or the points closest to the hyperplane, to optimize the separation between classes. (Dhiraj, K., 2021) SVC is a non-parametric algorithm that does not make assumptions on the number or shape of clusters in the data. It typically works best for low dimensional data. (Ben-Hur, A., 2008) This algorithm was instantiated with the sklearn SVC implementation. The algorithm was instantiated with the default parameters, fit to the data, then predictions were produced. SVC also performed well on the predictions. It produced an accuracy of 98% on training and 95% on test. That the training and test are both close in number and both well performing indicates that the model is not overfitting on the data and generalizes to the test set well. The hyperparameters were then tuned to see if the performance could be improved. The parameters tuned were kernel, degree, and gamma. The 'kernel' options tested were 'linear,' 'poly,' 'rbf,' and 'sigmoid.' These are the mathematical functions used to transform the data into a higher number of dimension spaces to create the non-linear decision boundary. (Major Kernel Functions in Support Vector Machine (SVM) - GeeksforGeeks, 2021) The 'degree' parameter is the degree of the polynomial. This is used only by the 'poly' kernel option and is ignored by all other kernels. (Pedregosa, 2011) The options tested for this parameter were 1, 2, 3, and 4. The 'gamma' parameter was also tuned with the options 'scale,' and 'auto.' This parameter essentially defines how far a single training example reaches. A low value means the example can reach far, and a high value means close. (Pedregosa, 2011) It is considered for the 'rbf,' 'poly,' and 'sigmoid' kernels. Tuning the model

showed the best parameters were degree 1, linear kernel, and scale for gamma. After tuning, the accuracy went up to 97%. Precision was 95%, Recall was 97.5%, and the F1 Score was 96%.

Next, Neural Network Classification was examined. The type of neural network used was Multi-layer Perceptron Classifier. This is a type of feedforward artificial neural network. The sklearn MLPClassifier implementation was used. This algorithm optimizes the log-loss function using LBFGS (the same solver described in the logistic regression section) or stochastic gradient descent. While there are not specific data assumptions, care should be taken that the training set is representative of all data classes to allow for generalization. (Meyer-Baese and Schmid, 2014) This model was instantiated with max_iter = 500 and the rest of the parameters set to the defaults. The data was fit and predictions made. The training data had a remarkable 99.4% and the test data followed closely behind at 99.1%. To see if perfect predictions could be achieved, the hyperparameters were tuned. The hyperparameters chosen for tuning were activation, alpha, and solver. Activation refers to which activation function will be chosen for the hidden layers. The options were 'identity,' 'logistic,' 'tanh,' and 'relu.' These are different mathematical functions which define how the weighted sum of the input is transformed into an output from the nodes in a layer of the network. (Brownlee, 2021) The alpha is the learning rate. The tested parameters were 0.0001, 0.001, 0.01, and 0.1. Finally, the solvers were 'lbfgs,', 'sgd,' and 'adam.' The LBFGS solver is the same that was described in the logistic regression section. SGD is Stochastic Gradient Descent and Adam is an optimized version of stochastic gradient descent. The resulting optimal hyperparameters turned were 'identity' for the activation function, 'alpha' of 0.0001, and 'Adam' for the solver. The tuned accuracy score went down slightly and was 98.2%. The Precision, Recall, and F1 Scores all tied at 97.5%.

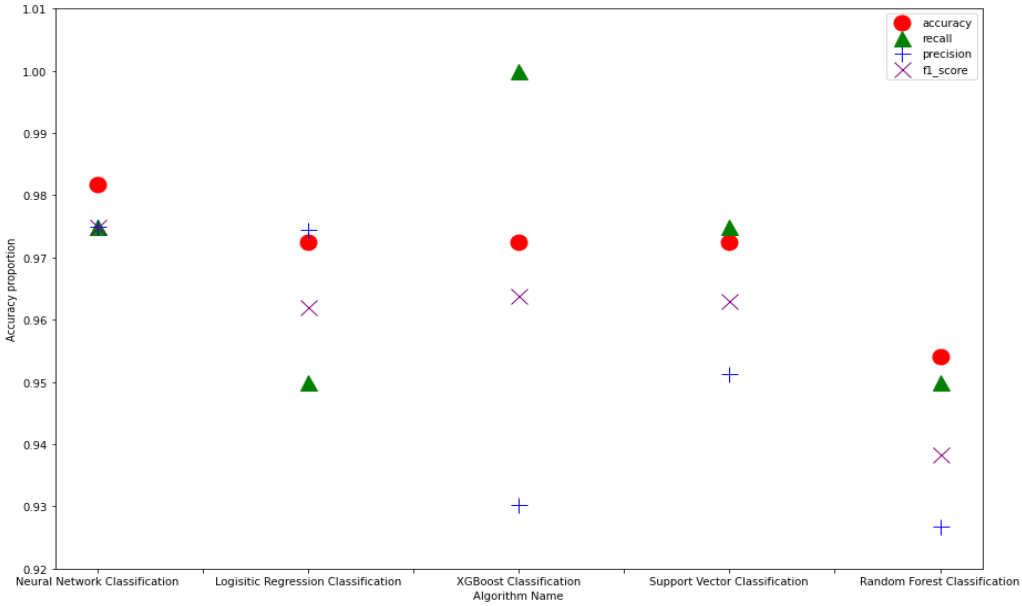
Finally, the last algorithm analyzed was XGBoost. This is an ensemble method that creates multiple decision trees. It is iterative and improves predictions based on the residual errors of the previous predictions. (How XGBoost Works - Amazon SageMaker, 2021) There are no assumptions on the data, though the categorical variables must be dummy encoded so that ordinal relationships are not inferred where they are not appropriate. While XGBoost is outside of the sklearn library, in this analysis XGBClassifier was used, which is a sklearn wrapper for the model. This model was instantiated with the use_label_encoder set to False and the eval_metric set to 'mlogloss'. The model was fit to the data and predictions generated. The training data

scored a 99.3% and the test data scored 98.1%. It followed the trends of the other models with excellent results. To attempt improvement, the max_depth, n_estimators, and learning_rate hyperparameters were tuned. Since XGBoost is based on decision trees, the max_depth and n_estimators parameters are equivalent to those in the Random Forest section. The parameters tested for max_depth were 1, 2, 3, 4, 5, and 6. For n_estimators the parameters tried were 10, 50, 100, 150, 200, and 250. The learning_rate is in this case refers to the schedule for weight updates which makes the boosting process more or less conservative. That is, to adjust more or less based on the results of the previous tree added. (Pedregosa, 2011) The test values for learning rate were 0.1, 0.01, 0.001, and 0.0001. Via GridSearchCV, the best parameters were chosen to be 0.1 for the learning rate, max_depth of 3, and n_estimators of 50. After running on the tuned model, the accuracy went down from 98.1% to 97.2%. The Precision score was 93%, Recall was 100%, and F1 Score was 96.3%.

Results

All models performed quite well classifying this dataset. The five algorithms ranged from a low accuracy of 95% on Random Forest Classification to a high of 98% on the Neural Network Classifier (MLPClassifier). Neural networks typically perform best on a large amount of data, so the excellent performance on a training set of 326 rows and a test set of 109 rows was a bit unexpected. XGBoost improved upon Random Forest's results which is not surprising since it takes a similar approach but iteratively learns from the results. On the best performing model, the results are so close to 100% accuracy that it is tempting to reach that goal. With further study, I would focus on the improving the Neural Network model. Things that may be optimized to improve accuracy would be to adjust the size of the training and test sets, and tune other hyperparameters in the model with a higher cross-validation number.

	name	accuracy	precision	recall	f1_score
0	Neural Network Classification	0.981651	0.975000	0.975	0.975000
1	Logisitic Regression Classification	0.972477	0.974359	0.950	0.962025
2	XGBoost Classification	0.972477	0.930233	1.000	0.963855
3	Support Vector Classification	0.972477	0.951220	0.975	0.962963
4	Random Forest Classification	0.954128	0.926829	0.950	0.938272


accuracy
recall
precision
f1_score
Accuracy proportion
1.01
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Neural Network Classification
Logisitic Regression Classification
XGBoost Classification
Support Vector Classification
Random Forest Classification
Algorithm Name

References

Asa Ben-Hur (2008) Support vector clustering. Scholarpedia, 3(6):5187.

Archive.ics.uci.edu. 2021. *UCI Machine Learning Repository: Congressional Voting Records Data Set*. [online] Available at: <https://archive.ics.uci.edu/ml/datasets/congressional+voting+records> [Accessed 19 November 2021].

Brownlee, J., 2021. *How to Choose an Activation Function for Deep Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/#:~:text=An%20activation%20function%20in%20a,a%20layer%20of%20the%20network.> [Accessed 19 November 2021].

definitions, L., 2021. *Logistic regression python solvers' definitions*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers-definitions#:~:text=It's%20a%20linear%20classification%20that,coordinate%20directions%20or%20coordinate%20hyperplanes.> [Accessed 18 November 2021].

Dhiraj, K., 2021. *Implementing Support Vector Machine with Scikit-Learn | Paperspace Blog*. [online] Paperspace Blog. Available at: <https://blog.paperspace.com/implementing-support-vector-machine-in-python-using-sklearn/> [Accessed 19 November 2021].

Docs.aws.amazon.com. 2021. *How XGBoost Works - Amazon SageMaker*. [online] Available at: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html [Accessed 19 November 2021].

GeeksforGeeks. 2021. *Major Kernel Functions in Support Vector Machine (SVM) –*

GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/major-kernel-functions-in-support-vector-machine-svm/#:~:text=Kernel%20Function%20is%20a%20method,required%20form%20of%20processing%20data.&text=So%2C%20Kernel%20Function%20generally%20transforms,higher%20number%20of%20dimension%20spaces.> [Accessed 19 November 2021].

Meyer-Baese, A. and Schmid, V., 2014. *ScienceDirect.com | Science, health and medical journals, full text articles and books.*. [online] Sciencedirect.com. Available at: <https://www.sciencedirect.com/topics/computer-science/multilayer-perceptrons> [Accessed 19 November 2021].

Richmond, S., 2021. *Algorithms Exposed: Random Forest | BCCVL*. [online] Bccvl.org.au. Available at: <https://bccvl.org.au/algorithms-exposed-random-forest/#:~:text=ASSUMPTIONS,are%20ordinal%20or%20non%2Dordinal.> [Accessed 19 November 2021].

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

Statistics Solutions. 2021. *What is Logistic Regression? - Statistics Solutions*. [online] Available at: <https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-logistic-regression/> [Accessed 17 November 2021].