CS 4641 Final Project

Supervised Learning with the Directed Dyadic Interstate War Dataset

December 2019

1 Introduction to the Dataset

This data set, the Directed Dyadic Interstate War dataset, contains 1364 instances of a series of lethal conflicts between at least two sovereign states and numeric features describing the conflict, including which state was the aggressor, year of conflict, and other information for a total of 16 relevant attributes [4]. The underlying supervised learning problem is, given such information describing an interstate conflict, classifying whether the result was a win, loss, or draw for the state. Solving this problem could enable prediction of war outcomes to be very accurate, and dissuade nations from aggression if models show they are likely to lose. Furthermore, whileapplying supervised learning techniques to this dataset is unlikely to provide a completely perfect model on determining the winners of future wars due to the drastic amount of factors that truly contribute to such an event, the returned model can provide an insight as to the effect certain attributes have on a state achieving victory in a conflict, such as the importance of initiating or defending against conflict, and how number of losses on the battlefield affects results beyond the assumption more deaths leads to a more likely loss. This information could be used in the future to refine overall military strategies based on how some factors are considered in each model. The final model can also give us a perspective on how these factors changed over the years to notice any trends in how various factors affected the final outcome. In order to measure how successful models predict war, accuracy on test data and total computation time will be taken into account.

2 Description of the Algorithms

Three classification algorithms were used to solve this problem: support vector machines, random forests, and neural networks. Each was implemented using the sklearn classes that implement these algorithms [3].

An SVM[1] is trained by maximizing the margin between points and the region boundary, using a penalty for misclassification. Its kernel is the function each

[1] Support Vector Machine

feature is mapped into a new space with multiplied by itself. Another hyperparameter is C, a positive value inversely proportional to how strong regularization is in training. Therefore, as regularization punishes complex models, a higher C increases complexity. Gamma is the coefficient used for any non-linear kernel (which will be the only kernels considered in this experiment). For this experiment, only two conventional ways of determining this value will be considered, $1/features$ (auto) or that value multiplied by the variance of x (scale). The decision function shape is used to determine if svm's are trained one class against another or one class against all others. Coef0 is a hyperparameter only used as an independent term when the kernel sigmoid is used[2].

The Random Forest trains a set number of decision trees on sub samples of the dataset, using the average of the decision trees in order to prevent overfitting. This implementation uses bagging, returning the label most decision trees decide on. It contains several hyperparameters to improve accuracy and avoid overfitting. Max Depth, the maximum depth allowed for each decision tree, decreases complexity the smaller it is. N estimators is the number of trees trained. Increasing it increases computation time, but controls overfitting. Max features is the highest amount of features each decision tree fits (each subset chosen randomly), which decreases complexity the lower it is. (For this experiment, only three conventional ways of determing this value will be considered). Criterion is the hyperparameter used to determine the function that measures how good a split in a decision tree is. The last hyperparameter, min-impunity, determines at what impurity a split may happen in each decision tree.

The Neural Network, implemented as a Multilayer Perceptron Classifier, has input, output, and a number (the hyperparameter hidden layer size) of hidden layers. The input layer consists of neurons, one for each feature, and the output layer consists of neurons that transform their input into one of the three labels. Hidden layers consist of a specified number of neurons, in between the input and output layers. At each neuron in each layer, the values provided by the previous layer are transformed with a linear summation, and then a non-linear activation function (the activation hyperparameter). For this experiment, the Neural Network was trained with backpropagation, calculating the error function's gradient with respect to weights in the neural network, optimized with a specified solver (the solver hyperparameter). Weights were updated on a schedule set by the learning rate hyperparameter (constant, invscaling, or adaptive). Unless learning rate is adaptive, the hyperparameter tol was specified to determine at what change in loss or score the training would stop. The hyperparameter 'alpha' was used to regularize the data by a set amount, punishing complex models.

[2]The optimal kernel, rbf, does not use Coef0.

3 Tuning of Hyperparameters

To achieve a reliable, optimally performing model, the hyperparameters were tuned using Grid Search. First, the data was split into two groups of train data (80%) and test (20%). The test data was reserved only for testing performance on the final model, while the train data was then further split with a stratified 3 fold for cross validation.

Lists of all possible qualitative hyperparameters were made, to ensure every combination of them was tested. For quantitative hyperparameters, the range of likely values was used to create a list of 5 to 10 values, spread linearly or exponentially throughout the range. The sklearn documentation was referenced in order to estimate an appropriate range, referencing all possible values from the description and size for the intervals from the default values. For Random Forests, the number of estimators was set to be 10, 25, 50, 75, and 100, a mostly linearly spread. Adding any estimators over 100 to an ensemble was unlikely to reproduce too different of results. Min impurity decrease, however, was set at values of 0 to 1e-9, divided by 10 each time, as it's size could vary greatly from 0 to 1. For the SVM, the only quantitative hyperparameter, C, was set in both linear fashion, 0.2 through 1, and exponential, 1e-4 to .1, to allow for any positive value below 1. Values above 1 were not originally tested to avoid reaching too complex of a model. Finally, for the neural network, similar criteria was used for alpha and tol as before, while the hidden layer size tested layers of size 3 through 10, with 10 and 100 neurons in each layer.

Models were trained with every combination of these hyperparameters listed and tested, using the train and validation data folds within the originally designated train data. The hyperparameters with the best results from this, scored with the mean accuracy on the validation data, are recorded in the table below[3].

Model	Neural Networks	SVM	Random Forests
Accuracy	71.86%	84.97%	99.63%
Parameters	alpha: 0.2 hidden layer sizes: 6 layers of 100 learning rate: constant solver: adam tol: 1e-05 activation: tanh	C: 0.8 decision function shape: ovo coef0: 0 gamma: auto kernel: rbf	criterion: gini max depth: 10 max features: auto min impurity: 0 n estimators: 25
Total GS Time	24:13:5	0:00:28	0:05:15
Time per Model	0.9 seconds	0.02 seconds	0.08 seconds

With grid search run, a general estimate of the best hyperparameters was ob-

[3]In total, 97,200 models were trained for analyzing neural network performance alone. Consequently, not all grid search results can be described.

tained for each algorithm. Therefore, to prove the dataset is not linearly separable and requires these non-linear models, hyperparameters were also tuned with grid search on logistic regression. The best score for each model was graphed against each other.

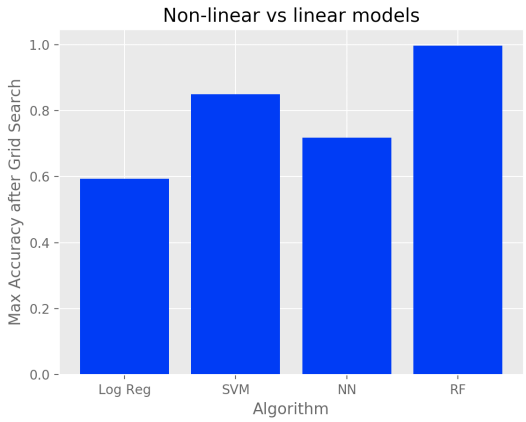

Figure 1: The non-linear models proved much more accurate

The worst non-linear model, with accuracy 71.86%, performed over 12% better than the linear one, with only 59.39% accuracy. Thus, it can be assumed the data is not linearly separable. A more complex model is necessary.

With the data established as not linearly separable and hyperparameters provided by Grid Search, a general estimate of the best parameters was obtained for each algorithm. However, while all qualitative hyperparameters were finalized, it was possible that quantitative hyperparameters could have been improved, as not every possible combination could be provided. Therefore, quantitative hyperparameters were graphed against accuracy for each of the 3 folds to find any trend in accuracy increase.

As the entire range of each quantitative hyperparameter was represented, qualitative hyperparameters likely won't improve by changing them with the further tuned quantitative hyperparameters. Though certainly possible, the obscurity of such an outcome did not warrant retraining thousands of models for each qualitative data point to find such a result.

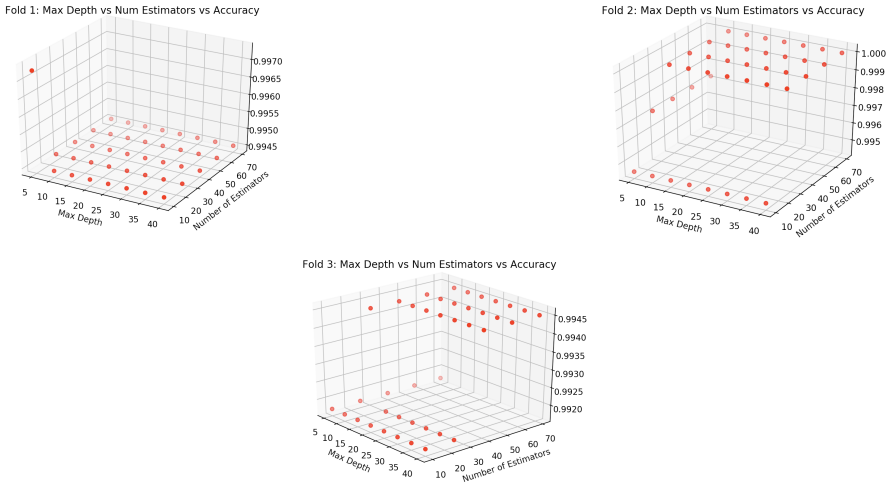

Figure 2: Random Forest Hyperparameters vs Validation Accuracy

The Random Forest was tuned first. The results show nearly 100% accuracy for each of the models. The accuracy is significantly higher with more than 25 estimators for Folds 2 and 3, with no effect in Fold 1. Accuracy essentially plateaus after 10 or 30 estimators in both cases. Changing max depth had no effect, likely due to decision trees not having such high of a complexity that would make this hyperparameter relevant. Therefore, 70 was chosen as the new number of estimators due to being the least complex graphed and having the same accuracy as others on the plateau, while max depth remained the same.

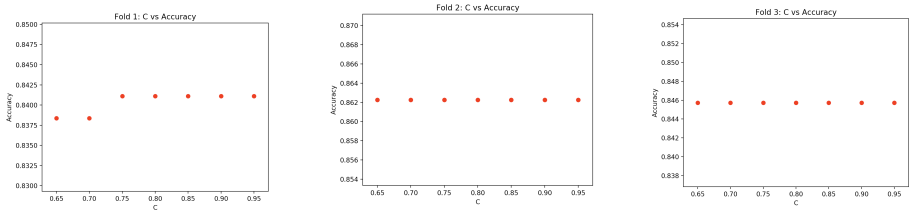

Figure 3: SVM Hyperparameters vs Validation Accuracy

Accuracy on Support Vector Machines remained fairly constant. The only quantitative hyperparameter, C, yields approximately 85% with every fold, only varying slightyly in Fold 1. Values above and below this range were rejected in the original grid search. Therefore, C = 0.75 was chosen, as it provides the least complex model while still yielding the highest accuracy in every model.

The Neural Network was tuned last, with it's three quantitative hyperparame-

ters of alpha, tol, and the amount of neurons in each layer[4].

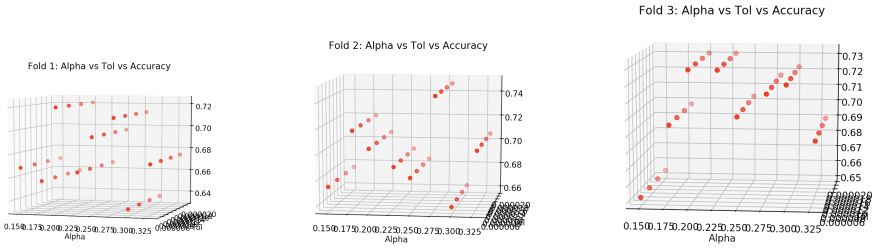

Figure 4: Neural Network Alpha vs Tol vs Validation Accuracy

Tol's effect on validation accuracy was, again, too small to visualize. Alpha, however, had a noticeable effect. A noticeable trend occurs in every fold, showing accuracy increase with a decrease in alpha from 0.33 until 0.18. Therefore, despite resulting in an increase of complexity as alpha regularizes data, 0.18 was chosen as the new value for alpha.

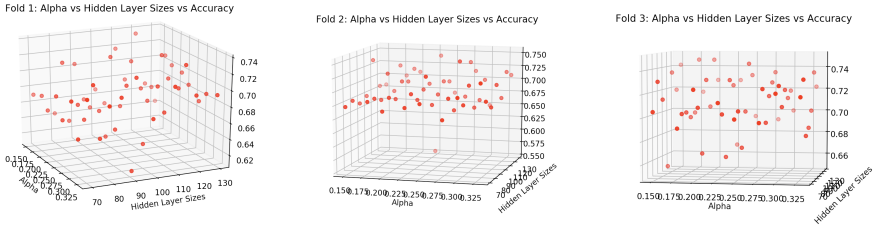

Figure 5: Neural Network Alpha vs Hidden Layer Neurons vs Validation Accuracy

No strong correlation was found when the number of neurons per hidden layer was graphed against alpha. 0.18 remained a strong value for alpha, so it was kept. The number of neurons per hidden layer was not changed.

The final parameters that were chosen are shown in the table below, with confidence intervals calculated in addition to the same single accuracy statistic. To properly compare to previous statistics from the original grid search, confidence intervals were created, using 95% confidence. No statistically significant change in validation accuracy occurred.

[4]The number of layers must be an integer, and every combination from 3 to 10 was already tested in grid search.

Model	Neural Networks	SVM	Random Forests
Accuracy	69.72%±2.40	84.82%±1.38%	99.63%±3.58
Parameters	alpha: 0.18 hidden layer sizes: 6 layers of 100 learning rate: constant solver: adam tol: 1e-05 activation: tanh	C: 0.75 decision function shape: ovo coef0: 0 gamma: auto kernel: rbf	criterion: gini max depth: 10 max features: auto min impurity: 0 n estimators: 70
Time per Model	1.19 seconds	0.03 seconds	0.1 seconds

4 Comparing Algorithm Performance

With finalized hyperparameters, each algorithm was trained with the entire train data set and tested for accuracy on both the reserved test set (untouched in hyperparameter tuning) and the train data set to evaluate both overall performance and amount of overfitting. The results are tabled below.

	Neural Networks	Support Vector Machine	Random Forests
Test Accuracy	70.33%	88.64%	99.63%
Train Accuracy	73.79%	99.91%	100%

The results from the table show that the models generalized well. The train data performs almost perfect, and quite better in the case of the Support Vector Machine, suggesting some overfitting, but overall the models are still at about 90% accuracy or higher. From this single statistic, it appears Random Forests performed the best, with almost 100% accuracy. Support Vector Machines still performed very well, performing the second best with 88.64% accuracy. The Neural Network again had the lowest accuracy, but didn't suffer from overfitting.

To provide more insight to compare these algorithms, confusion matrices[5] were produced , visualizing the results.

		Predicted Win	Predicted Loss
Random Forest	True Win	141	0
	True Loss	1	131

[5]As there was only one instance of a draw occurring throughout the data, which did not appear in the test data, the "draw" label was left out.

SVM		Predicted Win	Predicted Loss
	True Win	140	1
	True Loss	30	102

NN		Predicted Win	Predicted Loss
	True Win	111	30
	True Loss	51	81

The results align with expectations. There are slightly more wins (708) than losses (655) in the data, which explains why most accuracy loss came from inaccurately predicting a win in most models. The Random Forest performed the best, only inaccurately predicting a win once, while the SVM predicted did this 30 times, and even inaccurately predicted a loss once. Interestingly, the Neural Network predicted almost as many true wins as losses as it did true losses as wins.

However, reporting results with a single statistic, as done above, is limited, and may not truly represent generalization to the overall population. Therefore, confidence intervals were created by K-folding the test data into 50 subsets, and testing the trained model on each one to obtain a mean of errors. Using $\alpha = 0.05$, intervals for each model were obtained and are shown below. To find the inverse CDF, calculations were done outside of python using MATLAB and hardcoded into the final experiment script [2].

Random Forest	99.67% $\pm$0.65
Support Vector Machine	88.7% $\pm$3.44
Neural Network	70.7% $\pm$5.03

We are 95% confident that the true accuracy of each model is within these intervals[6]. The difference between the Random Forest and the next best algorithm, the SVM, is statistically significant.

Overall, the Random Forest performed significantly better than all other models in highest accuracy and lowest error. The SVM was almost as good, approximately 10% behind the Random Forest in all accuracy measures, though incorrectly labelling over 20% of losses in the test data.

5 Conclusion

Other metrics must be considered in order to make an overall decision on the best model to use. The Random Forest performed the best in terms of accuracy, but

[6]Intervals are reported using accuracy. For example, the interval for Random Forests error is $0.33\% \pm 0.65$

each one took longer to train than each SVM, likely due to its use of an ensemble of decision trees. The final SVM took 0.06 seconds to train, while the Random Forest took 0.05 time to train. Interestingly, the Random Forest actually took less time to train with all train data than during hyperparameter tuning. This is likely due to the wide variety of factors regarding computer architecture that influence runtime. Therefore, the computational complexities must be compared. Training this implementation of a Random forest requires $O(n^2\sqrt{d}n_{trees})$ time, while predictions require $O(dn_{trees})$ time, where n is the number of samples, d is the number of features, and n_{trees} is the number of estimators [1]. Training the SVM with an rbf kernel requires $O(n^2 n_{sv}d + n^3)$ time, while predictions require $O(n_{sv}d)$ time to complete, where n_{sv} is the number of support vectors. Plugging in the appropriate hyperparameters, it is clear that the n^3 term in the SVM makes it more computationally expensive to train, while, if n_{sv} and n_trees are treated as constants, $O(svmpredictions) = O(rfpredictions) = O(d)$. Therefore, the final Random Forest is slightly less computationally expensive to use.

The Neural Network performed surprisingly terrible. The final model itself required an entire 3.34 seconds to complete training. Grid Search required over 24 hours to return the optimal hyperparameters, in part because there were so many important hyperparameters that needed to be tuned. However, even at the most optimal hyperparameters found, the best model still yielded the worst accuracy, even with intervals of 95% confidence. Therefore, the neural network was not selected to be used for a problem with data from the same domain.

The Random Forest was selected to be used for a problem with data from the same domain. While it only performed slightly better than the SVM in terms of computation time and needed more hyperparameters tuned, it had a much better accuracy, and only incorrectly labelled one point in the entire test data set. The final model did not require too much complexity, having 70 estimators to prevent overfitting along with a low maximum depth of 10. Therefore, the Random Forest is the optimal algorithm to use on data from the same domain.

6 Acknowledgements

All sources used for this project are listed in the references, in addition to referencing lecture slides from the course webpage.

References

[1] "Computational Complexity of Machine Learning Algorithms." The Kernel Trip, 16 Apr. 2018, www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/.

[2] "p." Normal Inverse Cumulative Distribution Function - MATLAB, www.mathworks.com/help/stats/norminv.html.

[3] "Supervised Learning" Scikit, scikit-learn.org/stable/supervisedlearning.htmlsupervised-learning.

[4] Zeev Maoz, Paul L. Johnson, Jasper Kaplan, Fiona Ogunkoya, and Aaron Shreve 2019. The Dyadic Militarized Interstate Disputes (MIDs) Dataset Version 3.0: Logic, Characteristics, and Comparisons to Alternative Datasets, Journal of Conflict Resolution (forthcoming).