Scott Bamford

6/24/2021

Final Paper

DATA 71200

Predicting Economic Freedom in the World

The dataset that was used for projects 1 – 3, is about the scores of Economic Freedom based upon a multitude of different features. These features can range from if they have a black market, the free flow of currency, economic capital, taxes, tariffs, trade barriers, etc., Each of these features are used to predict as well as decide the value of their economic freedom. These were done from 2000 – 2019 and are from a number of different countries. Although for the projects the actual year as well as country doesn't matter mainly just the data. For project 1 and 2 I predicted if the freedom score was above or below 5. While for Project 3 it was based upon a rounded freedom score.

For project 3 the score was based upon the economic freedom score. The economic freedom score is a float value, because of this I decided to create both a rounded up and a rounded down score in order to than categorize them. There was an issue that I did run into which was that the score labeled "0" only had one actual predictor because of this I had to drop that value since I couldn't actually split the dataset equally. I cleaned my data by dropping rows with missing values in the target variable. This was chosen because since the target values were hard labeled and I didn't want to fill in the target with wrong values, especially because it is the target.

For the missing labels for the features, I used the mean value of the specific columns. The mean that was being used is the mean from the Training split for both the training split columns and the testing split columns. It was tested that filling the missing values based upon the full dataset produced a better result compared to just the Training dataset. The results that were showed that using the full dataset was better for supervised learning. Although I didn't test this with unsupervised learning so as a result, I cannot comment on this. But for the projects moving forward I used the mean filling methods based upon the training sets.

I chose to use the mean filling methods based upon the training set and using the training set to fill in the missing values for the testing set. I chose to use this method because it prevented bias from developing since it would be using the testing data when normally the testing data was unavailable. After I cleaned the missing and null values, I looked at used the Sklearn's standard scaler in order to lessen the mean and standard deviation in the data to make the data closer to each other.

Finally, I used correlation tables on the Economic Freedom score in order to determine the most correlated features. None of the values were below 0.2 correlation, but I did end up dropping the values below the 4c_black_market feature. After visualizing the dataset, it would appear that when using the binary predictors (above or below 5), it showed there were more above 5. But when I was using the multi labeled data it showed a gaussian distribution.

When splitting the dataset, I made sure to stratify the dataset by the labels in order to guarantee a balance training and test data set. Later on, I will use Stratified Shuffle instead of specifying stratify in the training split function from sklearn.

Supervise Learning Experiments:

For the supervised learning I used both the KNN, Logistic Regression, and Support Vector Machines. The predictors that I chose was the binary predictors (above or below 5). The Dataset preprocessing was the same as described above but instead used the 4b_regulatory_trade_barriers as the minimum correlation and would be used to subset. A question that I wanted to identify before running the supervised learning was to see the effects of filling in missing values before splitting the data or after splitting the data. The dataset that was standardized after the split for training and test data, used the training data in order to fill in the mean values as well as using the training data to standardize both the training and testing data. When running the KNN model in order to see the effects there were three datasets being used. The first dataset (pre-split) was the data that was preprocessed as well as standardized all together, the second dataset (post-split) was the data that was went through preprocessing after the split as well as using the Training set to standardize both training and test sets. And the final dataset (Base) was just the dataset after preprocessing.

The Presplit data preformed the best with the KNN model, with a 0.976 F1. The base preformed second best with a F1 score of 0.974, and post-split preformed the worse with a 0.860. I did not go further with these findings and test it on different models aside form KNN, but I was able to understand why each dataset preformed as they did. The Post-Split preformed the worst because when predicting, the values that were being predicted were not representational of the testing data instead it had parts of the missing data attached. These parts were the filled in missing values using the training data. Because of this the model is unable to run efficiently since the post-split dataset had a mix of training and testing values which caused the prediction to be thrown off. The Presplit data performed so well because the testing and training data were standardized together. At the end of the day, I went forward with the post-split dataset because the potential for bias to be developed because I was using the testing data, which I wasn't supposed to be using.

Now for Supervised learning I used two models, K Nearest Neighbors and Support vector machines. K Nearest Neighbors (KNN), utilizes the distance between values and based upon the number of neighbors will chose the labels based upon the most populated within that number. Changing the Neighbor can change how the actual model predict as well as changing the metric of distance. The distance is important because it is the way in which you're able to rank the specific labels. The parameters that were being chosen are neighbors, metric, weight, and algorithm. The metrics are Manhattan, Euclidean, and minkowski. Manhattan distance is calculated through the same idea as walking city blocks, Euclidean is direct paths, so instead walking a city block you're able to walk through it in a diagonal fashion, minkowski is the absolute value of the distance which scale based upon the value being less than one. The weights are just if we're using as specific value to change how much one value will be counted and just normal. For algorithm are auto, ball tree, kd tree, brute. Auto automatically chose the right one, while ball tree uses that method, kd tree uses that method, while brute force is just going through ever possible option.

Using optimization using grid search, a test was also run, on using the stratified shuffling for the folds or just using 5 for the folds. When optimizing for both using CV and not, they showed different optimal hyperparameters. When using stratified shuffling it said that Auto, Manhattan, 8 Neighbors, and distance. When using just five as the fold it said that the optimal hyper parameters was auto, Manhattan, 10 neighbors, and distance. So, after running the values that were hyperparameters it showed that the hyperparameters based upon stratified shuffle, resulted in a 1.0 training accuracy, and a 0.97 testing accuracy. When running the parameters with just the folds being 5, it also had a 1.0 training accuracy, but a 0.976 testing accuracy.

Overall, I think that the stratified split was better even though they had generally the same results. The reason for this is the number of neighbors that were being used. I think that the stratified shuffle was better at capturing the different labels from the overall dataset and was able to balance it better compared to the other way. The other method that was used was Support Vector Machines. Support vector machines are in my eyes similar to clustering but instead of grouping based upon distance instead it makes hyper plane on two features that portions the data off. These hyperplanes are the boundaries from one label to the other. The different parameters that are being tested are the C value, Gamma, and the kernel. The C is the value in which your will penalize the number of features the dataset has. The Gamma is the strength of each training example on being able to predict outcomes. And the Kernel is how the data is going to be used and run.

In the project, I ran an early grid search looking at both C, Kernel, and Gamma. I quickly identified that Gamma and Kernel were locked in at linear and 0.5 for gamma. The optimal C value that was chosen was 0.7 for folds equal to five, and 0.8 for folds equal to the stratified shuffling. For Stratified Shuffling the training Accuracy was 0.993 and Test accuracy was 0.990 with a F1 of 0.989. For folds equal to just five, the accuracy for the training set was 0.993, and testing set 0.99 with a F1 of 0.989. Each predicting a 595 / 601 of the labels correctly.

After this I wanted to further look at the results and see if I could expand and further improve the outcome by looking a larger range of C, while keeping the Gamma and kernel the same. So, the next test was to look at C values in the range from zero to one with a 0.01 distance between each value. As always, I did this twice, one with stratified folds and one with just folds equal to 5. For stratified folds the optimal C value was 0.94, while for folds equal to five the optimal C value was 0.68. After re running the prediction and testing the values it showed that the stratified fold optimal C produced a better result than the folds equal to 5. For stratified folds, training accuracy was 0.993, testing accuracy was 0.991, F1 of 0.991, and 596/601 correctly labels. For folds equal to 5, the training accuracy was 0.993, 0.990, F1 of 0.989m and 595/601.

These results showed a slight improvement in prediction as well as displayed the benefits of stratified folding. The Reason these predictions ended up the way that they were was because compared to KNN where I was looking at the distance between everything, in SVM I am looking for a single line or plane to bisect the data in order to portion the labels off from one another. Because of this it was able to make an improvement over KNN. I can say that my labels are closely packed and are able to be separated easily based upon different lines. Not only this but it also shows that my data is somewhat unbalanced in terms of when you decide to search for optimal hyperparameters. The Stratified folds got an improvement while the unstratified did not. This improvement was because of the equally split data in order to better account and distribute the two different kinds of labels.

PCA is a way in which an experimenter is able to cut down on the number of features that the overall dataset has. PCA uses clustering in order to accomplish this feat, by looking at looking at the data, by looking at the data it then decides which features are needed in order to either meet the number of clusters specified, or to mean the percentage of the dataset that was needed.

The dataset that was being used for the project three was different in terms of the number of labels, although the preprocessing as well splitting was the same as in the previous projects. The labels that were used for classification was switched from instead of looking either above or below five is now looking the Economic Freedom Score rounded up and down. I made two separate splits of training and testing data for the rounded up and down data. I was unsure of exactly how the rounded data would affect predictions as well as identify the different kinds of labels. It turns out that after rounding the data there was a single label which didn't have more than one example of it. Because of this I decided it was for the best to drop all rows that had that specific outcome. I am sure that this could've been circumvented if in the preprocessing I replaced missing target values with either mean or median of that column. But because it is the target that I am modeling for, I didn't want to affect the dataset with this filling in of data.

For this dataset I used the Support Vector Machines, and the results for rounded down data with a training accuracy of 0.877, testing accuracy of 0.851, F1 score of 0.838, and number of correct predictions were 506/ 601. For the rounded-up data the training accuracy was 0.88, testing accuracy 0.843, F1 score of 0.839, and the number of correct predictions were 507/601. From this I decided to just use the rounded-up data moving forward. After hyperparameter tuning, as well as using stratified shuffling for the folds, a C value of 0.1, Gamma of 0.5, and a kernel of poly was the optimal hyper parameters. These parameters produced a Training Accuracy of 0.97, testing accuracy of 0.836, F1 score of 0.837, and number of correct predictions were 503/ 601. Clearly it shows that this specific training was overfitting. And I decided to just use Base SVM when testing for PCA selection.

For PCA selection with the selected purpose of identifying 95 percent of the dataset, identified eleven number of columns compared to the seventeen. When using the PCA selected data on the SVM, it produced an even worse result compared to the default parameters of SVM. Although I do think that this could be a result of the number of labels it would have to predict. The results of the PCA transformed dataset, had a training accuracy of 0.86, testing accuracy of 0.823, F1 of 0.818, and 396 number of correct predictions out of 481.

KMeans was also used as another form of clustering. I predicted that the ideal number of clusters would be around 8 or 9, mainly because that was the number of different target values. I ran KMeans and identified its elbow point, in which the distortion score was just slowing down as the number of clusters increased. Doing this I saw that the correct value was 7, which is 8 labels, which makes sense. I ran KMeans on the training data set without PCA scaling and with PCA scaling. The number of clusters for both of the datasets had a value of 7. But the ARI and silhouette scores were different. Without PCA the silhouette score was 0.1809 and an ARI score of 0.284. With PCA transformation on the dataset had a silhouette score of 0.193, and a ARI of 0.304. For KMeans it identified that PCA transformation was able to cluster better that without PCA transformation in the KMeans model.

Another clustering model that was looked at was looked at was the Agglomerative Clustering. Using the same number of clusters that were shown in the KMeans (7), I ran the model with both PCA transformation as well as without. Without the PCA transformation the ARI was 0.31, and had a silhouette score of 0.15. With PCA transformation the ARI was 0.26, and a Silhouette score of 0.16. Agglomerative Clustering is clustering based upon a distance from one another. This means that the goal is the minimize the distance between points. The goal is to group clusters based upon their similarities. The PCA transformation did worse because of the lower number of features for the algorithm to identify. The Agglomerative clustering works based upon grouping based upon similarity, naturally with a lower ability to identify similarity it would cause a resulting worse score since PCA reduces the number of features to use in the model.

The final model used was DBSCAN. The DBSCAN has two parameters the distance the points have to be from its neighbors and the minimum number of datapoints that a cluster has to be in order to be considered a cluster. DBSCAN then uses these parameters to create clusters. The results was taken from using PCA transformation and without. Without PCA transformation the ARI had a score of 0.008, and an Average Silhouette score of 0.225. With PCA transformation the ARI score was 0.069, and a silhouette score of 0.313. I am sure that these results could've been changed but after running some optimization over these the best result had a minimum of 1 per cluster which resulted in almost 890 different clusters where in reality there should only be about 9 based on the actual target labels for the dataset. Not only this but this dataset is very densely packed in the sense that the relationship between each cluster isn't easy to see on a 2-dimensional plane. This density can throw off this specific clustering because it looks at distance from another point as a value to optimize as well as number per cluster to identify these new clusters.

I learned a lot in this class, a lot of it was to do with visualization. Most of the models I have had some experience with and because of that I had spent a lot of time adding things to look at. These are exemplified by look at the effects of standardizing at different points of the training testing split, identifying the benefits and costs of stratified shuffling on the folds of a grid search. Looking at the differences in Multi Class classification and binary classification. The last was the most important that I learned. If I had to re do anything, I would most likely work with multi class labeling from the start. Since the dataset's original target is a float from 0 – 10. In which I converted to either a binary classification or in project 3 a multi classification. I find using multi classification created more opportunity for me to experiment with the model in terms of causation for different outputs. As well as expanded my expertise onto different kinds of predictive modeling.