A Replication & Extension of Caruana & Niculescu-Mizil (CNM06)

Urmi Suresh USURESH@UCSD.EDU
Department of Cognitive Science
University of California San Diego
La Jolla, CA 92093, USA

Abstract

This paper explores the process and results of a small replication of the analysis done in the paper by Caruana & Niculescu-Mizil (CNM06), with the purpose being to evaluate a few different supervised learning algorithms and determine if any of them outperform the others at binary classification. The performance of four various learning algorithms on four data sets are measured by three different metrics: accuracy, AUC, and F1 score. The algorithms performed are support vector machines (SVM), logistic regression, k-nearest neighbors (KNN), and random forests. Overall, random forests consistently outperforms the other models, matching the results of CNM06.

Keywords: Binary Classification, Support Vector Machines (SVM), Logistic Regression, K-Nearest Neighbors (KNN), Random Forests, Supervised Learning Algorithms, Performance Metrics

1. Introduction

The driving force between CNM06 was that Caruana and Niculescu-Mizil felt that there was a lack of comprehensive empirical studies that compared and evaluated the performance of various learning algorithms. Besides STATLOG (King et al., 1995) that was comprehensive at the time of its publication, there was no study that explored newer learning algorithms such as boosting, SVMs, bagging and random forests. Therefore, Caruana and Niculescu-Mizil took it upon themselves to create a complete comparison of ten supervised learning algorithms measured by eight different performance metrics on eleven different data sets. The learning algorithms assessed in CNM06 are bagged trees, boosted stumps, boosted trees, decision trees, logistic regression, memory-based learning, naive bayes, neural nets, SVMs, and random forests. These models are evaluated on accuracy, average precision, cross-entropy, F-score, Lift, precision/recall break-even point, ROC Area, and squared error. Furthermore, CNM06 takes the comparison of models one step further by calibrating the predictions of each algorithm with Platt Scaling and Isotonic Regression to address the issue of how some of the performance metrics interpret predictions as probabilities and some models are not capable of predicting probabilities. The algorithms are compared both before and after the calibrations.

As the reasoning behind my study is to replicate CNM06 on a smaller case as well as provide more support for the results present in CNM06, I recreated and expanded the experimentation

done by Caruana and Niculescu-Mizil. In an effort to simplify the complexity of CNM06, I chose a smaller subset of the algorithms, performance metrics, and data sets present in CNM06. The four learning algorithms I chose to recreate are support vector machines, logistic regression, k-nearest neighbors, and random forests. The three performance metrics are accuracy, ROC-AUC, and F-score. Finally three out of the four data sets were chosen from the UC Irvine Machine Learning Repository with the fourth one being from the StatLib Repository. Two of the main data sets are already used in CNM06 and two of the data sets are independent from the CNM06 study. For each of the models performed, a variety of hyperparameters are tested in order to produce the highest accuracy, AUC, and F-score. The specific parameters are detailed below under the Learning Algorithms section. Ultimately, the goal behind the following paper is to present a smaller study that corroborates the results in CNM06 and shows that even with a smaller subset of variables the results are still consistent.

2. Methodology

2.1. Learning Algorithms

This section contains a thorough description of all of the parameters used when running each of the following algorithms. When replicating the algorithms the parameters used for each learning algorithm are as close to CNM06 as possible, with a few purposeful adaptations.

SVMs: I specify the kernel types to be used in the algorithm as linear, polynomial (poly), and radial basis function (rbf). The varying gamma widths used only for the rbf kernel are {0.001, 0.005, 0.01, 0.1, 0.5, 1, 2}. The different degrees of the polynomial kernel function are 2 and 3. For all three of the kernels, the regularization parameter C is varied by the following values: 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, and 10^{3}.

Logistic Regression (LR): For logistic regression I train on both regularized (penalty = ‘l2’) and unregularized (penalty = ‘none’) models. For both the regularized and unregularized models I keep the algorithm to use in the optimization problem as ‘lbfgs’ as when I ran the ‘l2’ penalty with ‘liblinear’, ‘lbfgs’, and ‘saga’ the ‘lbfgs’ solver performed consistently better. Similarly when I ran the ‘one’ penalty with ‘lbfgs’ and ‘saga’ the ‘lbfgs’ performance was consistently higher than the ‘saga’ performance. For the regularized model I vary the regularization parameter by the following values: 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, and 10^{4}.

KNN: For the number of neighbors I use 26 evenly spaced out whole number values ranging from K=1 to K= n/10 where n is the training set size. In this case n=5000 making the range from K=1 to K=500. For the distance metric to use for the tree I chose ‘euclidean’ and for the weight function used in prediction I chose both ‘uniform’ as well as ‘distance’.

Random Forests (RF): Due to using sklearn, I use the Breiman-Cutler implementation of random forests rather than the Weka implementation. This implementation was also the one reported by CNM06 due to it yielding better results than the Weka implementation. I chose to have 1024 as the number of trees in the forest, and the number of features being considered as

part of the decision of finding the best split are as follows: 1, 2, 4, 6, 8, 12, 16, and 20.

For logistic regression, support vector machines, and k-nearest neighbors I use Standard Scaler to transform the data such that the distribution will take on a mean value of 0 and a standard deviation of 1, but for the random forests algorithm I do not scale the data.

2.2. Performance Metrics

The three performance metrics I chose to measure the performance of the learning algorithms on the four data sets are Accuracy (ACC), Area Under the Receiver Operating Characteristic Curve (AUC), and F1-score (FSC).

Accuracy returns a number between 0 and 1 and it is the number of correct predictions made over the total number of predictions that the model makes. However, accuracy works the best only when the data sets are balanced and may even be deceitful when it is used on imbalance data sets (Mishra, 2018). This is why it is used in combination with the AUC and F1-score metrics as they result in more accurate results, even when data sets are unbalanced. The equation to calculate accuracy is shown below.

$$accuracy = \frac{\Sigma\, True_{positive} + \Sigma\, True_{negative}}{\Sigma\, Total_{population}} = \frac{TP + TN}{TP + FP + TN + FN}$$

The value resulting in a ROC-AUC calculation ranges from 0 to 1, and the higher the value the better the model is performing. What the AUC tells us is how well a classifier is able to distinguish between the classes by calculating the area under the curve of a receiver operating characteristic curve (Hernández, 2019).

The F1 score uses the harmonic mean to combine both precision and recall, and it results in a metric that tells you both how precise your model is. The values of F1 score range between 0 and 1 and the larger the F1 value, the better the model is performing. The equation to calculate the F1 score is shown below.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

2.3. Data Sets

There are four binary classification problems used to test the learning algorithms in this study. Two are used in the original CNM06 paper (CALHOUSING and LETTER) and two are taken from the UCI Machine Learning Repository (AVILA and ELECTRIC). LETTER also appears in the UCI Repository, however CALHOUSING originates from (Perlich et al., 2003). Before performing any analysis, I start by cleaning the four data sets as well as graphing them to get a visual representation of the data I would be working with (visuals shown in Appendix). Originally, I had started with four data sets directly from CNM06 (ADULT, CALHOUSING, COVTYPE, and LETTER), but after creating some visualization and performing some initial testing on my learning algorithms I introduced some external data sets. Numerical descriptions for all of the relevant data sets are shown below in Table 1.

The AVILA data set was initially split into two sets, the training set and the testing set, so I start by concatenating them together to create one data set. I first named the columns, and then because the data was all already numerical I did not have to do any one hot encoding so I isolated my label column and transformed it into binary values. I took the largest class and treated that class as positive and the rest as negative. This resulted in an only slightly imbalanced data set with 8572 counts of 1 (41% positive) and 12295 counts of 0 (59% negative).

I start cleaning CALHOUSING by renaming its columns to the appropriate attribute names for clarity. Again, the values were all numerical resulting in a simple cleaning process. I isolated the median_house_value column as the target column and transformed it into binary labels by classifying values > $130000 as 1 and values ≤ $130000 as 0. This data set was by far the most imbalanced of the four data sets with 14728 counts of 1 (71% positive) and 5912 counts of 0 (29% negative).

ELECTRIC came with numerical attributes and prelabeled columns so I only rename the target column for clarity. Then I discretize the target column by replacing the classification of 'stable' with 1 and the classification of 'unstable' with 0. This again resulted in an unbalanced data set with 3620 counts of 1 (36% positive) and 6380 counts of 0 (64% negative).

For LETTER I start by renaming the columns to make the data more comprehensive. Then I isolate the target column as 'capital_letter' which is the only non numerical attribute in the data set. I discretize this column by treating the letters from A-M as positive and the letters N-Z as negative. This yielded the most balanced data set with 9940 counts of 1 (49% positive) and 10060 counts of 0 (51% negative).

Table 1. Description of Problems

PROBLEM	#ATTRIBUTES	TRAIN SIZE	TEST SIZE	%POZ
AVILA	11	5000	15867	41%
CALHOUSING	9	5000	15640	71%
ELECTRIC	14	5000	5000	32%
LETTER	17	5000	15000	49%

3. Experiment Composition & Results

The general structure of the experiment is that for each of the algorithm/data set combinations created from the four learning algorithms and four data sets there are five trials of searching for optimal hyperparameters. In each of the trials, 5000 random samples are randomly selected from the data set to perform 5-folds cross validation on, and the remaining samples are set aside. Then the appropriate lists of hyperparameters are created. These are determined by the current learning algorithm and fed into the algorithm using a Pipeline. Next, GridSearchCV is used to streamline the process of isolating optimal performing hyperparameter combinations for each of the three performance metrics that were previously decided: accuracy, AUC, and F1. Once the optimal

hyperparameters are generated, three models are created with the best hyperparameters respective to each of the three performance metrics. With these models I was able to predict on the test set which consisted of the remaining samples not selected as part of the random selection of 5000 samples. Then the raw test performance measurements were outputted for each trial for each data set as well as computed averages for easy comparison between algorithms (Table 9 in Appendix).

3.1. Performances by Metric

The normalized scores for each of the four learning algorithms (SVM, logistic regression, KNN, and random forests) by metric averages over all of the four data sets are shown below in Table 2. The table is denoted where the bolded numbers correspond to the highest value for that metric and the ones that have an asterisk beside them are the values that after performing an independent t-test did not prove to have a significant difference from the bolded values. Accordingly, values that do not have an asterisk next to them are values that are significantly lower than the bolded values according to the independent t-test. The threshold used as a determining factor in the independent t-tests is p=0.05, and for every t-test the test did not assume equal variance and thus performed a Welch's t-test rather than a standard independent two sample test. Tables 5 and 6 are supplemental tables for Table 2 are located in the Appendix and show each of the distinct t-statistics and p-values for each t-test that was performed on the values in Table 2.

Table 2. Testing Set Performance
(normalized scores for each learning algorithm by metric averaged over four problems)

MODEL	ACC	AUC	FSC	MEAN
SVM	0.858*	0.848*	0.876*	0.860
LR	0.804*	0.861*	0.791*	0.819
KNN	0.811*	0.843*	0.801*	0.818*
RF	**0.953**	**0.985**	**0.959**	**0.966**

In this table each of the scores are averages of the performance metrics for all four of the data sets. For example, the SVM/ACC cell is the average of the four accuracy scores calculated for each of the data sets. Acting as a baseline for comparison is the exact same values calculated for the Training Set Performance seen in Table 4 in the Appendix. The rightmost MEAN column is the average over all five trials of the four data sets and three metrics for one learning algorithm. Looking at the table we can see that the pattern suggests that random forests is the strongest performing learning algorithm, however none of the algorithms performed particularly poorly. In fact, the t-test and resulting p-values showed that every value in the accuracy, AUC, and F1 score columns are not significantly lower than the bolded values in each metric's respective columns. In the entire table there were only two values, both from the MEAN column, that are significantly lower than the bolded number.

The testing set performance results can be compared to the training set performance results recorded in the Appendix. In the training results, random forests performed perfectly across every

metric and KNN was very close behind it. This differs slightly from the testing set performance as in the testing set table random forests are still the highest performing model, but SVMs are next with both KNN and logistic regression performing almost equally. The values in the Training Set Performance table are higher than the Testing Set but not absurdly high implying that overfitting did not occur. Had the values in the Testing Set been much lower than the training set there would be more of a concern of overfitting existing. Nonetheless, some variation between training set performance and testing set performance is expected and highly welcome.

3.2. Performances by Problem

Table 3 shown below contains the normalized scores for each learning algorithm by problem and each value is averaged over all three of the performance metrics. This table follows the same denotation as Table 2 where the bolded values are the highest values in that column (in this case the highest value for each data set) and the numbers with asterisks beside them are the values that according to a t-test calculation have p-values greater than a threshold of p=0.05 and are therefore not significantly lower than the bolded value in that column. For each cell each of the three metrics are averaged for one algorithm/data set combination. For instance, in the SVM/AVILA cell the accuracy, AUC, and F1 score for all trials run on the AVILA data set with the SVM learning algorithm are averaged and is outputted as one score seen in the table.

Table 3. Testing Set Performance
(normalized scores for each learning algorithm by problem averaged over three metrics)

MODEL	AVILA	CALHOUSING	ELECTRIC	LETTER	MEAN
SVM	0.848	0.680	0.996	0.918	0.860*
LR	0.674	0.852	0.993*	0.756	0.819*
KNN	0.809	0.725	0.775	**0.965**	0.818*
RF	**0.975**	**0.929**	**0.999**	0.961	**0.966**

From this table we can see that random forests is the best performing learning algorithm for three out of the four data sets. For AVILA after random forests, the second best performing model are SVMs, followed closely by KNNs, and then finally logistic regression. The CALHOUSING data set actually had logistic regression as its second best performing model with KNNs as the third and lastly SVMs. Numbers for the ELECTRIC data set were exceedingly close to each other with SVMs coming in second, then logistic regression, and ultimately KNNs. The logistic regression result for ELECTRIC was also the only value besides some in the MEAN column that after performing a t-test and collecting p-values was considered significantly closer to the highest value. Finally in the LETTER data set the KNN learning algorithm actually outperformed the other algorithms with random forest coming second, then SVM and lastly logistic regression. This is interesting because LETTER was the most balanced of the data sets and yet the one data set where random forests did not perform the best. However, this circumstance may just be an anomaly as in this study there are only five trials being performed for every algorithm here and the values are so close that with a different set of trails the numbers could very well change. The

corresponding t-statistics as well as the p-values for all of the figures in Table 3 can be found in Tables 7 and 8 in the Appendix.

4. Discussion

In CNM06 their highest performing learning algorithms include bagged trees, neural nets, and random forest trees, and this is all prior to calibration. After calibration, SVMs performance improved significantly to be nearly as well performing as random forests. For the most part, single decision trees and logistic regression are two learning algorithms that were not performing at a high level. However, in both the CNM06 results as well as in my replication study, the generalizations do waver as some learning algorithms fluctuate between trials or have a better performance with a specific data set.

Ultimately, the results from my smaller scale replication are aligned with CNM06. Random forests consistently performed the best out of the four algorithms. KNNs and SVMs both were also steadily produced high results with logistic regression coming in last. However, logistic regression still performed fairly well as all four learning algorithms generated high results. As previously stated, the general trends in the performance of the four learning algorithms remained constant. However, between different trials and various computations, the algorithms did produce varying results.

In order to get more comprehensive and generalizable results I would have to include more trials, learning algorithms, and performance metrics and eventually reach a similar scale as CNM06. One downfall with comparing significant differences with an independent t-test is that there were such few trials and a higher threshold of $p=0.05$. I do believe that more extensive testing and a lower threshold would result in fewer false positives and more valid results. Because of the small scale of the study many of the values would fluctuate between trials and although there is a consistent pattern it does not mean that they results never varied.

Ultimately, the small scale reproduction was predominantly successful at replicating the study performed in CNM06. Accordingly, the original goal of substantiating the results in Caruana and Niculescu-Mizil's paper by emulating their study was achieved.

Bonus

To extend the base requirements of the project I included a fourth learning algorithm into my code. I also started with the data sets given in the paper but branched out to pick up two more data sets. The extra data cleaning done for the ADULT and COVTYPE data sets that were originally done but then replaced is shown as Figures 5 and 6 in the Appendix. The reason behind this was that I wanted to get the experience of working with using one hot encoding to turn nominal data into usable variables and that was done in both ADULT and COVTYPE. I also did some testing on LETTER V1 where the target label was split very unevenly into classes just to confirm that the performance metrics were outputting expected results. Cleaning for this data set is listed as Figure 7 in the Appendix. I also wanted to create visualizations of my data so as to not blindly work on

data sets so I went ahead and graphed the AVILA, CALHOUSING, ELECTRIC, and LETTER data sets. Screenshots of these graphs are included in the Appendix.

Acknowledgements

I would like to take a moment and acknowledge support for this project by Professor Jason Fleischer as well as TA Abdullah Al-Battal and the rest of the COGS 118A TAs and IAs. They were all endlessly patient during office hours and discussion sections throughout the entire quarter, but especially during this project. The lecture slides, github notebooks, and homework assignments were also greatly appreciated as they helped me break down class concepts and made everything understandable. I am also extremely grateful to my peers who were eager to help answer questions on Piazza and further helped clarify my understanding on numerous machine learning topics.

References

Caruana, R., & Niculescu-Mizil, A. (2006). An Empirical Comparison of Supervised Learning Algorithms. *Proceedings of the 23rd International Conference on Machine Learning*.

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Fleischer, Jason, "Jasongfleischer/UCSD_COGS118A." *GitHub,* github.com/jasongfleischer/UCSD_COGS118A.

Hernández, Pablo. "Mine Is Better: Metrics for Evaluating Your (and Others) Machine Learning Models." *DataScienceAero*, 5 Apr. 2019.

King, R., Feng, C., & Sutherland, A. (1995). Statlog: comparison of classification algorithms on large real world problems. *Applied Artificial Intelligence, 9.*

Mishra, Aditya. "Metrics to Evaluate Your Machine Learning Algorithm." *Medium*, Towards Data Science, 24 Feb. 2018.

Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions, Statistics and Probability Letters, 33 (1997) 291-297.

Perlich, C., Provost, F., & Simonoff, J. S. (2003). Tree induction vs. logistic regression: a learning-curve analysis. J. Mach. Learn. Res., 4, 211–255.

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

"Scipy.stats.ttest_ind." *Scipy.stats.ttest_ind - SciPy v1.6.1 Reference Guide*, SciPy, 18 Feb. 2021, docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html.

Appendix

Table 4.

Training Set Performance

(normalized scores for each learning algorithm by metric averaged over four problems)

MODEL	ACC	AUC	FSC	MEAN
SVM	0.977	0.740	0.972	0.896
LR	0.806	0.863	0.793	0.820
KNN	0.986	1	1	0.995
RF	1	1	1	1

Table 5 (Supplemental for Table 2).

P-values for mean test set performance across trials for each algorithm/data set

MODEL	ACC	AUC	FSC	MEAN
SVM	0.203	0.332	0.142	0.0003
LR	0.112	0.090	0.157	0.014
KNN	0.070	0.119	0.069	0.994
RF	1	1	1	1

Table 6 (Supplemental for Table 2).

T-statistics for mean test set performance across trials for each algorithm/data set

MODEL	ACC	AUC	FSC	MEAN
SVM	1.537	1.149	1.853	9.784
LR	2.096	2.390	1.846	6.408
KNN	2.472	2.129	2.621	0.006
RF	0	0	0	0

Table 7 (Supplemental for Table 3).

P-values for mean test set performance across trials for each algorithm/metric

MODEL	AVILA	CALHOUSING	ELECTRIC	LETTER	MEAN
SVM	1.529e-12	3.271e-16	0.009	0.012	0.215
LR	4.526e-16	3.142e-06	0.065	1.862e-12	0.121
KNN	4.516e-11	1.976e-10	7.931e-08	1	0.060
RF	1	1	1	0.026	1

Table 8 (Supplemental for Table 3).
T-statistic for mean test set performance across trials for each algorithm/metric

MODEL	AVILA	CALHOUSING	ELECTRIC	LETTER	MEAN
SVM	77.201	210.040	4.528	4.279	1.531
LR	209.803	25.530	2.509	151.445	2.083
KNN	79.845	229.184	91.446	0	2.740
RF	0	0	0	2.852	0

Table 9.
Raw Test Scores

			AVILA	CALHOUSING	ELECTRIC	LETTER
SVM	TRIAL 1	ACC	0.835	0.712	0.996	0.843
		AUC	0.906	0.489	0.999	0.987
		FSC	0.798	0.832	0.994	0.815
	TRIAL 2	ACC	0.835	0.713	0.995	0.929
		AUC	0.910	0.499	0.999	0.987
		FSC	0.795	0.833	0.993	0.932
	TRIAL 3	ACC	0.831	0.711	0.996	0.886
		AUC	0.911	0.498	0.999	0.987
		FSC	0.792	0.831	0.995	0.895
	TRIAL 4	ACC	0.840	0.715	0.991	0.893
		AUC	0.913	0.498	0.999	0.987
		FSC	0.804	0.834	0.987	0.881
	TRIAL 5	ACC	0.835	0.712	0.996	0.890
		AUC	0.910	0.498	0.999	0.987
		FSC	0.797	0.831	0.994	0.877
LR	TRIAL 1	ACC	0.685	0.808	0.981	0.725
		AUC	0.755	0.864	0.998	0.812
		FSC	0.575	0.872	0.973	0.731
	TRIAL 2	ACC	0.685	0.808	0.996	0.726
		AUC	0.764	0.868	0.999	0.812
		FSC	0.571	0.871	0.994	0.727
	TRIAL 3	ACC	0.683	0.828	0.996	0.723
		AUC	0.758	0.888	0.999	0.810
		FSC	0.574	0.873	0.994	0.724

	TRIAL 4	ACC	0.690	0.813	0.994	0.728
		AUC	0.770	0.868	0.999	0.815
		FSC	0.569	0.874	0.991	0.728
	TRIAL 5	ACC	0.687	0.808	0.994	0.730
		AUC	0.762	0.865	0.999	0.817
		FSC	0.578	0.871	0.992	0.733
KNN	TRIAL 1	ACC	0.793	0.700	0.7922	0.957
		AUC	0.871	0.662	0.855	0.980
		FSC	0.749	0.812	0.684	0.957
	TRIAL 2	ACC	0.801	0.697	0.789	0.957
		AUC	0.881	0.665	0.857	0.980
		FSC	0.758	0.810	0.681	0.957
	TRIAL 3	ACC	0.794	0.698	0.796	0.955
		AUC	0.874	0.665	0.856	0.980
		FSC	0.750	0.811	0.689	0.955
	TRIAL 4	ACC	0.799	0.700	0.786	0.960
		AUC	0.878	0.665	0.846	0.982
		FSC	0.757	0.8105	0.665	0.959
	TRIAL 5	ACC	0.800	0.700	0.794	0.955
		AUC	0.875	0.661	0.854	0.980
		FSC	0.759	0.815	0.681	0.954
RF	TRIAL 1	ACC	0.965	0.900	0.999	0.943
		AUC	0.995	0.958	1	0.989
		FSC	0.956	0.930	0.998	0.940
	TRIAL 2	ACC	0.972	0.898	1	0.947
		AUC	0.996	0.956	1	0.990
		FSC	0.966	0.929	1	0.948
	TRIAL 3	ACC	0.967	0.895	1	0.949
		AUC	0.994	0.956	1	0.991
		FSC	0.961	0.927	1	0.949
	TRIAL 4	ACC	0.968	0.903	0.999	0.943
		AUC	0.995	0.958	1	0.990
		FSC	0.960	0.932	0.999	0.945
	TRIAL 5	ACC	0.972	0.901	0.999	0.949
		AUC	0.995	0.956	1	0.991
		FSC	0.964	0.931	0.999	0.949

Figure 1.
Avila Data Set

Figure 2.
California Housing Data Set

Figure 3.
Electrical Grid Stability Simulated Data Data Set

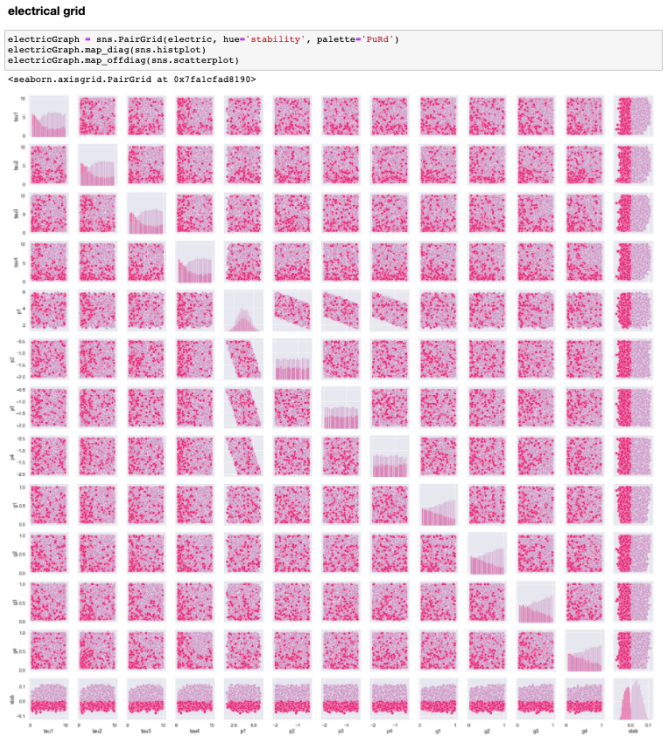

Figure 4.
Letter Data Set

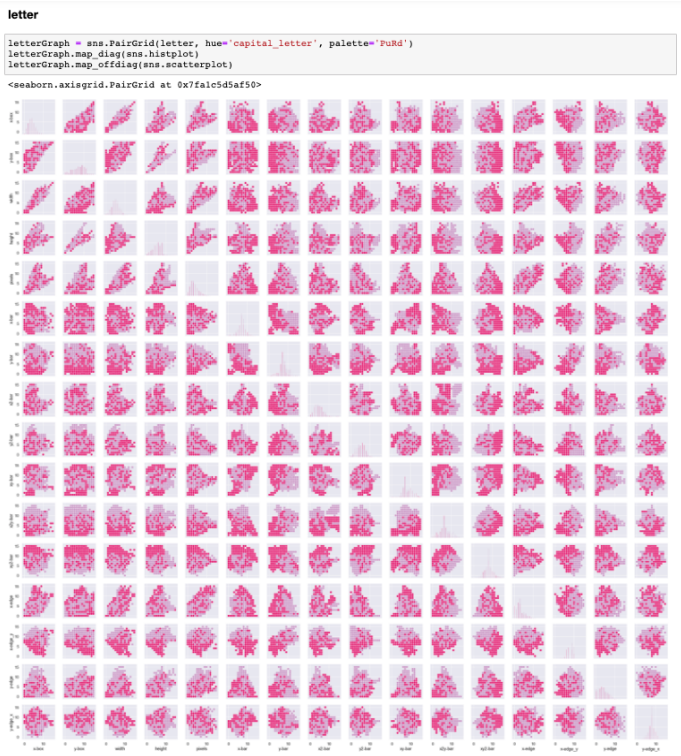

Figure 5.
ADULT Data Set Cleaning

adult data set

```
# load in data
adult = pd.read_csv('data/adult.data', header=None)
```

```
# rename columns
adult.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
                 'occupation' ,'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
                 'hours-per-week', 'native-country', 'income']
```

```
# turning nominal data columns into one hot encoded values
# (workclass, education, marital status, occupation, relationship, race, native country)

workclass_dummies = pd.get_dummies(adult.workclass, prefix='workclass')
adult = pd.concat([adult, workclass_dummies], axis=1)

education_dummies = pd.get_dummies(adult.education, prefix='education')
adult = pd.concat([adult, education_dummies], axis=1)

maritalstatus_dummies = pd.get_dummies(adult['marital-status'], prefix='marital-status')
adult = pd.concat([adult, maritalstatus_dummies], axis=1)

occupation_dummies = pd.get_dummies(adult.occupation, prefix='occupation')
adult = pd.concat([adult, occupation_dummies], axis=1)

relationship_dummies = pd.get_dummies(adult.relationship, prefix='relationship')
adult = pd.concat([adult, relationship_dummies], axis=1)

race_dummies = pd.get_dummies(adult.race, prefix='race')
adult = pd.concat([adult, race_dummies], axis=1)

nativecountry_dummies = pd.get_dummies(adult['native-country'], prefix='native-country')
adult = pd.concat([adult, nativecountry_dummies], axis=1)

# making sex and income columns binary
adult.sex.replace([' Female', ' Male'], [0, 1], inplace=True)
adult.income.replace([' <=50K', ' >50K'], [0, 1], inplace=True)
```

```
# dropping nominal data columns that became one hot encoded
adult = adult.drop(['workclass', 'education', 'marital-status', 'occupation', 'relationship',
                    'race', 'native-country'], axis=1)

# moving income column to beginning
tempCols = adult.columns.tolist()
tempCols.insert(0, tempCols.pop(tempCols.index('income')))
adult = adult.reindex(columns = tempCols)
```

```
print(adult.income.value_counts())
adult.head()
```

```
0    24720
1     7841
Name: income, dtype: int64
```

	income	age	fnlwgt	education-num	sex	capital-gain	capital-loss	hours-per-week	workclass_?	workclass_Federal-gov	...	native-country_Portugal	native-country_Puerto-Rico	native-country_Scotland	native-country_South	native-country_Taiwan	native-country_Thailand
0	0	39	77516	13	1	2174	0	40	0	0	...	0	0	0	0	0	0
1	0	50	83311	13	1	0	0	13	0	0	...	0	0	0	0	0	0
2	0	38	215646	9	1	0	0	40	0	0	...	0	0	0	0	0	0
3	0	53	234721	7	1	0	0	40	0	0	...	0	0	0	0	0	0
4	0	28	338409	13	0	0	0	40	0	0	...	0	0	0	0	0	0

5 rows × 108 columns

Figure 6.
CALHOUSING Data Set Cleaning

covertype data set

```
# load in data
covertype = pd.read_csv('data/covtype.data', header=None)
```

```
# rename columns
covertype = covertype.rename(columns={0: 'elevation', 1: 'aspect', 2: 'slope',
                                      3: 'horizontal_dist_to_hydrology', 4: 'vertical_dist_to_hydrology',
                                      5: 'horizontal_dist_to_roadways', 6: 'hillshade_9am', 7: 'hillshade_noon',
                                      8: 'hillshade_3pm', 9: 'horizontal_dist_to_fire_points',
                                      10: 'wilderness_area_1', 11: 'wilderness_area_2', 12: 'wilderness_area_3',
                                      13: 'wilderness_area_4', 54: 'cover_type'})

# renaming soil type columns
count = 1
for i in range(14, 54):
    temp = ('soil_type_%s' % (count))
    covertype = covertype.rename(columns={i: temp})
    count = count + 1
```

```
# change cover type to binary
covertype.cover_type = np.where(covertype.cover_type == 2, 1, 0)
```

```
print(covertype.cover_type.value_counts())
covertype.head()
```

```
0    297711
1    283301
Name: cover_type, dtype: int64
```

	elevation	aspect	slope	horizontal_dist_to_hydrology	vertical_dist_to_hydrology	horizontal_dist_to_roadways	hillshade_9am	hillshade_noon	hillshade_3pm	h
0	2596	51	3	258	0	510	221	232	148	
1	2590	56	2	212	-6	390	220	235	151	
2	2804	139	9	268	65	3180	234	238	135	
3	2785	155	18	242	118	3090	238	238	122	
4	2595	45	2	153	-1	391	220	234	150	

5 rows × 55 columns

Figure 7.
LETTER V1 Data Set Cleaning

letterV1 recognition data set

```
# load in data
letterV1 = pd.read_csv('data/letter-recognition.data', header=None)
```

```
# rename columns
letterV1.columns = ['capital_letter', 'x-box', 'y-box', 'width', 'height', 'pixels',
                    'x-bar' ,'y-bar', 'x2-bar', 'y2-bar', 'xy-bar', 'x2y-bar', 'xy2-bar',
                    'x-edge', 'x-edge_y', 'y-edge', 'y-edge_x']
```

```
# turn letter O to positive (1) and the rest to negative (0)
letterV1.capital_letter = letterV1.capital_letter.replace(['O'], 1)
letterV1.capital_letter = letterV1.capital_letter.replace(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
                                                           'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S',
                                                           'T', 'U', 'V', 'W', 'X', 'Y', 'Z'], 0)
```

```
print(letterV1.capital_letter.value_counts())
letterV1.head()
```

```
0    19247
1      753
Name: capital_letter, dtype: int64
```

	capital_letter	x-box	y-box	width	height	pixels	x-bar	y-bar	x2-bar	y2-bar	xy-bar	x2y-bar	xy2-bar	x-edge	x-edge_y	y-edge	y-edge_x
0	0	2	8	3	5	1	8	13	0	6	6	10	8	0	8	0	8
1	0	5	12	3	7	2	10	5	5	4	13	3	9	2	8	4	10
2	0	4	11	6	8	6	10	6	2	6	10	3	7	3	7	3	9
3	0	7	11	6	6	3	5	9	4	6	4	4	10	6	10	2	8
4	0	2	1	3	1	1	8	6	6	6	6	5	9	1	7	5	10