Machine Learning Project 1

Anthony Vasquez
Ph:313-920-8877
Email: avasque1@jh.edu

Editor: Anthony Vasquez

Abstract

The topic of this document is to experiment with simplistic models using a modular end-to-end approach to loading, preparing, testing, training, and evaluating a model. The two models that are developed are a naïve regressor and classifier. Using the accuracy metric, it is predicted that both models will not perform well at either regression or classification but will likely perform better than a randomized output given any test input.

Keywords: Naive, Classifier, Regressor, Preprocessing, Categorical, Standardization, K-fold, Stratified, Cross-validation, Accuracy

1 Introduction

A trained and tested model is the finished product of a machine learning pipeline, but there are many stages of the pipeline that are equally important. A common saying with regards to model training is, "garbage in garbage out," where "garbage in" is data not rigorously curated and can show up in the form of a lack of feature correlation, etc... A model learns from input which may offer little to no use for the data scientist. For the purposes of this document, it is assumed that all the data was useful and topics like feature correlation were of no concern, but data preprocessing proved to be an essential step to the implementation of the naïve algorithms used in this experiment.

The start of the experiment was to implement a data loader on six datasets given in csv format. Therefore, a brief section on the contents of the data is discussed followed by a section on preprocessing data. Preprocessing is defined here as the execution of a data curation plan in preparation for the successful training and testing of a model. Examples of data curation are dropping rows that have missing data or replacing that data with other data. Other preprocessing steps discussed are how to handle categorical data, data standardization, and cross validation. It should be mentioned that the latter topic is handled in a separate section.

k-fold validation assumes the data has been purged of inconsistencies, missing values, or any other defect that can lead to strange results. It falls in the data split section, because this is where the data is divided into multiple validation and train sets. These details will be handled more thoroughly in its respective section, followed by the learning algorithms used.

On account of their naivety, the two algorithms used are covered to a short extent. These algorithms are essentially place holders used to run the data through the pipeline during the training testing, and evaluation which will be a part of every future project of this course. Analyzing model performance, is crucial step to understanding the learner's ability to generalize to new data, therefore, model evaluation is included, since it is the looking glass from which the scientist can evaluate what was learned by the model and determine if the model needs fine-tuning, new features, more data, or different data altogether.

Finally, a simple experiment is done to test the pipeline and naïve algorithms. It is expected that the naïve classifier and naïve regressors, though extremely simple, should perform better then a model that takes any input and output a randomly selected class.

2 Data

This section discusses the data and preprocessing techniques used in this project.

2.1 Data Used

The were six datasets in csv format used. The datasets included a data file, where all the feature and target information were located. A names file was included that included meta data such as background, statistics and attribute names. The names of the datasets and a brief description include:

- Abalone is a dataset used to predict the age of a gastropod species by counting the number rings on the inner shell of the sample. There are 4177 samples with eight attributes to include, sex, length, diameter, height, whole weight, shucked weight, viscera weight, shell weight, ring number (Waugh, 1995). There are no missing values in this dataset.

- Breast-cancer-wisconsin is a dataset that includes 699 samples with 10 attributes and is used to classify either benign or malignant cancer types (Wolberg, 1992). This dataset is missing 16 values that are denoted by a "?."

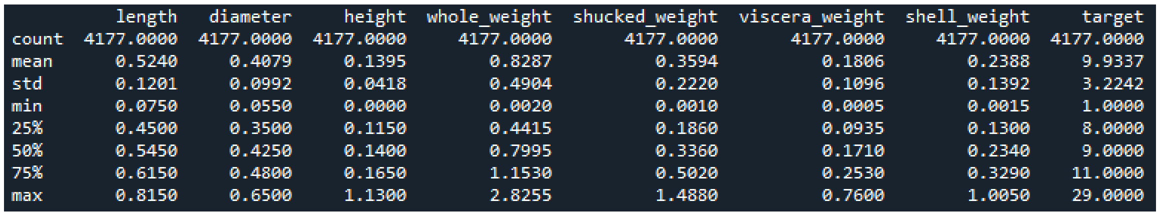
- Car is a dataset used to evaluate hierarchy induction tools. There are 1728 samples with six attributes and four classes including, unacc, acc, good, and v-good (Marko Bohenec, 1997). This dataset has no missing values.

- Forestfires is a dataset used to predict the burn area of a forest fire based on 517 samples, and >12. The attributes include special coordinates X and Y of a park map, month, data, indices from the Fire Warning Information system, temperature, relative humidity, wind speed, rain, and burn area (Paulo Cortez, 2007). This dataset has no missing values.

- House-votes-84 is a dataset used to predict whether a congressman from the house of representatives belongs to the republican or democrat parties based whether they voted on 16 key votes: handicapped-infants, water-project-cost-sharing, adoption-of-the-budget-resolution, physician-fee-freeze, el-salvador-aid, religious-groups-in-school, anti-satellite-test-ban, aid-to-nicaraguan-contras, mx-missile, immigration, synfuels-corporation-cutback, education spending, superfund-right-to-sue, crime, duty-free-exports, and export-administration-act-south-africa (Schlimmer, 1984). The attributes are either a "y" or a "no," and missing values are represented as a "?."

- Machine is a dataset that is used to predict relative cpu performance and includes 209 samples with 10 attributes. Attributes include, vendor name, model name, machine, cycle time, minimum main memory, maximum main memory, cache memory, minimum channels, maximum channels, published relative performance, and estimated relative performance (Phillip Ein-Dor, 1987). This dataset has no missing values.

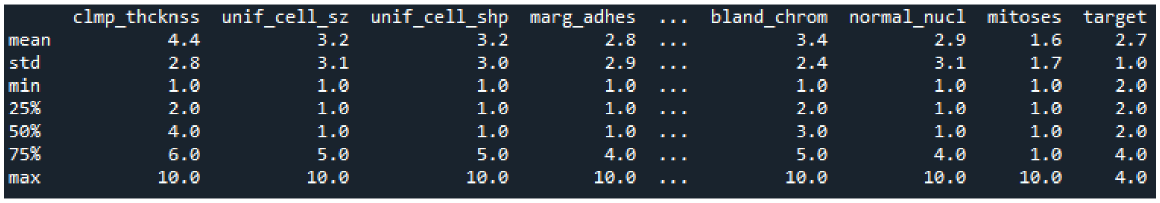
Each of the above datasets were used to develop the machine learning pipeline to ensure that different datatypes can successfully undergo preprocessing prior to the train, test, evaluate tasks. It However, because of the limited time and space, not all the datasets were used for experimentation and so will not be represented in the results. For this reason, only two datasets were used, that is, abalone because it can be used for both classification and regression, and breast-cancer-Wisconsin. Abalone is a dataset used for both classification and regression problems in machine learning. The age of abalone is found by staining the inner shell and counting the rings with a microscope. This dataset has 4177 samples with eight attributes and no missing data (Waugh, 1995).



```
         length   diameter     height  whole_weight  shucked_weight  viscera_weight  shell_weight     target
count 4177.0000  4177.0000  4177.0000     4177.0000       4177.0000       4177.0000     4177.0000  4177.0000
mean     0.5240     0.4079     0.1395        0.8287          0.3594          0.1806        0.2388     9.9337
std      0.1201     0.0992     0.0418        0.4904          0.2220          0.1096        0.1392     3.2242
min      0.0750     0.0550     0.0000        0.0020          0.0010          0.0005        0.0015     1.0000
25%      0.4500     0.3500     0.1150        0.4415          0.1860          0.0935        0.1300     8.0000
50%      0.5450     0.4250     0.1400        0.7995          0.3360          0.1710        0.2340     9.0000
75%      0.6150     0.4800     0.1650        1.1530          0.5020          0.2530        0.3290    11.0000
max      0.8150     0.6500     1.1300        2.8255          1.4880          0.7600        1.0050    29.0000
```

Table 1. This table shows a basic pandas statistical analysis of the abalone dataset prior to standardization.

The breast-cancer wisconsin dataset was gathered in the early 1990s by a physician at the university of Wisconsin Hospital. It has eight attributes and two classes[1] with missing values. More on attributes and classes can be found in section 2.1. and statistical analysis is shown in



```
      clmp_thcknss  unif_cell_sz  unif_cell_shp  marg_adhes  ...  bland_chrom  normal_nucl  mitoses  target
mean           4.4           3.2            3.2         2.8  ...          3.4          2.9      1.6     2.7
std            2.8           3.1            3.0         2.9  ...          2.4          3.1      1.7     1.0
min            1.0           1.0            1.0         1.0  ...          1.0          1.0      1.0     2.0
25%            2.0           1.0            1.0         1.0  ...          2.0          1.0      1.0     2.0
50%            4.0           1.0            1.0         1.0  ...          3.0          1.0      1.0     2.0
75%            6.0           5.0            5.0         4.0  ...          5.0          4.0      1.0     4.0
max           10.0          10.0           10.0        10.0  ...         10.0         10.0     10.0     4.0
```

Table 2. This table shows a basic pandas statistical analysis of the breast-cancer-wisconsin dataset prior to standardization.

2.2 Data Preprocessing

Both the abalone and breast cancer datasets were standardized by changing all values to range from -1 to 1. Standardization was achieved by using the equation:
$z(x) = \frac{x-\mu}{\sigma}$, where x is the data value, μ is the mean, and σ is the attribute standard deviation. The sex attribute in abalone was encoded using pandas, while the breast-cancer-wisconsin did not need to be encoded, as all the data was numerical. The breast-cancer-wisconsin dataset did have missing values which were replaced by μ.

3 Data Split

This section discusses the data split method of stratified k-fold cross validation.

3.1 Stratified K-fold Cross Validation

The data was separated into several stratified train and validation sets for k-fold cross validation, where k = 5. The full data was stored as a pandas dataframe but was subsequently broken into smaller dictionaries for the randomized sampling of the data and each dictionary was appended to a list that was iterated through to obtain the five train and validation sets (Figure 1).

[1] The two classes are benign and malignant.

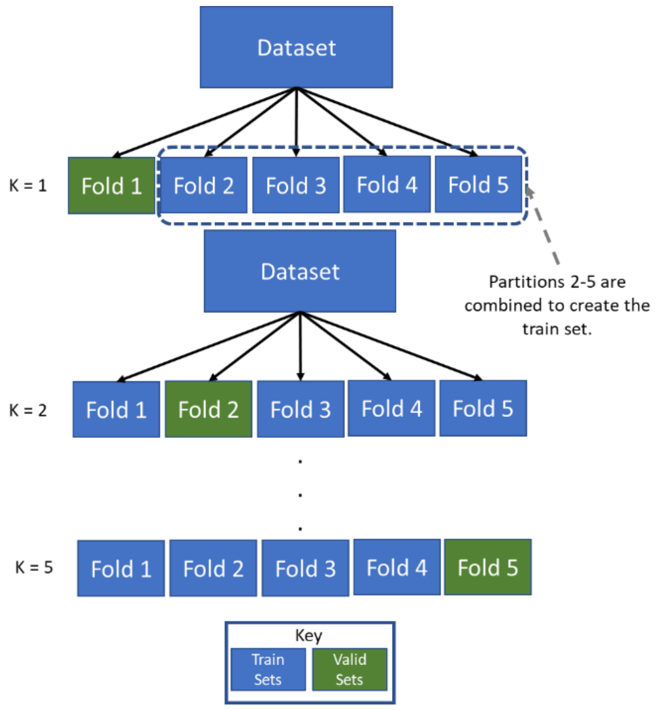


Figure 1: K-fold cross validation was used for the experiments of this document. The green represents the validation set, where blue are the combined trainsets for that iteration.

4 Models

This section discusses the naïve models used for this experiment.

4.1 Naïve Classifier

As mentioned in an earlier section, the two models used were naïve. The first was a naïve classifier which would calculate the majority class for each train set. To do this, the train set was iterated through and counted for each class which were appended to a 'class' and 'count' list of a dictionary. This was a relatively easy task using python's flexible dictionary data structure and its built-in list functionality. First the max value of the count dictionary was found using $\max\_count = \max(count\_dictionary['count'])$. Once the maximum value was found, the index could be retrieved using $\max\_index = count\_dictionary['count']index(\max\_value)$. Finally, the majority class was retrieved using $\max\_class = count\_dictionary['class'][\max\_index]$. This process was done k times, and the average of the majority class for each train set was recorded. The majority class was set as the output given any input from the test set.

4.2 Naïve Regressor

The Naïve regressor implementation was simpler, because the mean of a column of data can be found using the pandas mean method $column_{mean} = df['target']mean()$. The column mean was calculated for all train sets and was used as the output for any test set input.

5 Results

This section discusses the accuracy results of using the two naïve methods mentioned in 4.1 and 4.2 vs the accuracy of randomly sampling a class instead of the naïve approach.

5.1 Naïve Classifier Results

The results from using the naïve classifier were puzzling. Not only was the accuracy calculation a low (5.88%), but the accuracy was the same for each test set (Figure 2). It took a while to realize why that was. After some digging, it was discovered that a k-fold stratified split guarantees that each class will have the same number of samples. How then was a maximum class count obtained, if all classes had the same count? To answer this question, results from python list methods needed to be analyzed.

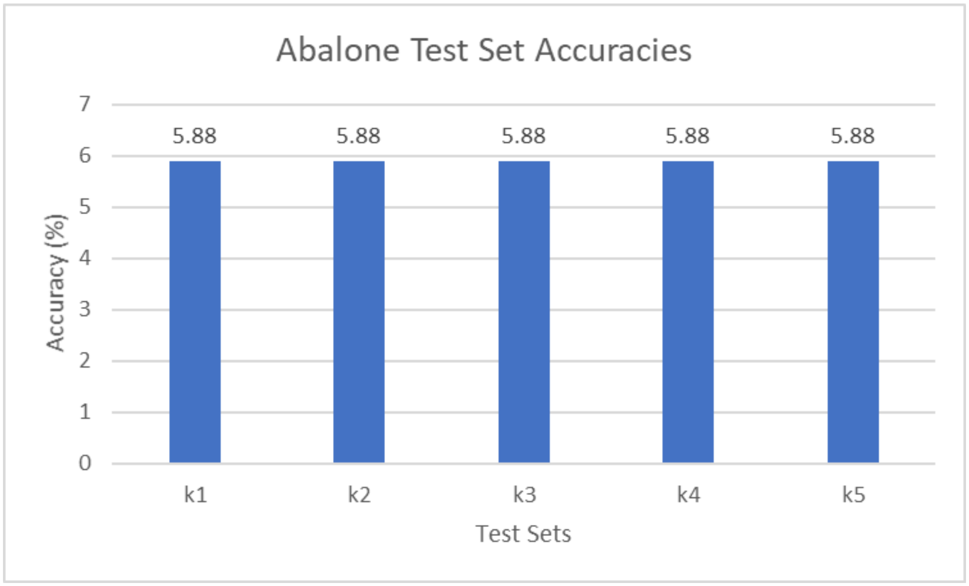


Figure 2. Each of the test sets scored the same accuracy for the naïve classifier

Recall section 4.1, that by iterating through the dataset, each class was counted on each train set, which was the same for all counts, since stratification was used during the k-fold split. For example, the count for each class for the abalone dataset was 20, which makes the max count 20. Therefore, python's list.index() functionality returned the index of the maximum count which defaulted to zero. Inserting the max_index into the class count dictionary's "class" key returned the first class in the list, which was four. There were the same number of class fours in each test set[2], and the same number of fours in each prediction set[3]. Accuracy is defined as $accuracy = \frac{correct\ predictions}{total\ predictions}\ X\ 100$. Using this equation results in $accuracy = \frac{5}{85}\ x\ 100 = 5.88\%$ which is statistically what is expected from a randomly guessing. Randomly generating values was not pursued in python code since the result was obvious. The results for breast-cancer-wisconsin was the same in that all the test sets scored the same accuracy, but there were only two classes so the accuracies scored higher at 50%.

[2] There were five instances of class four.

[3] There were 85 instances of class four in the prediction set, because the naïve classifier outputs the majority class for all input.

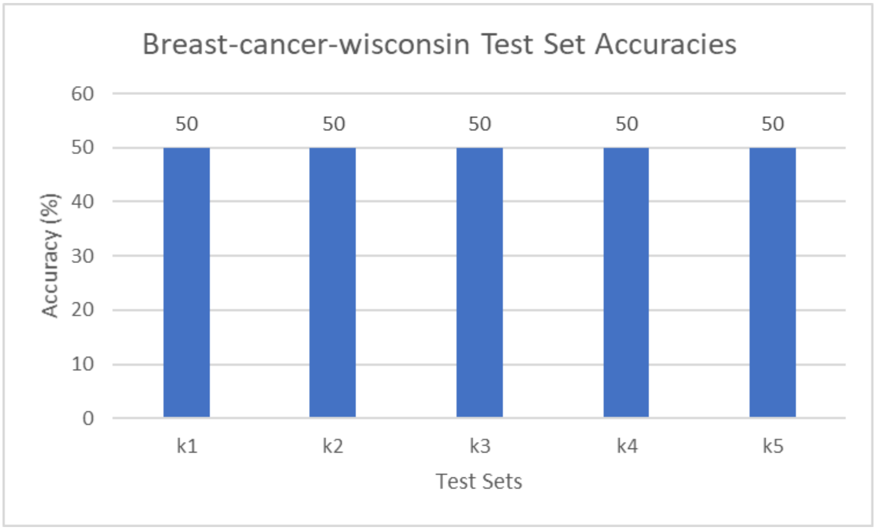


Figure 3. Similar results were calculated from the breast-cancer-wisconsin test sets with higher accuracy from only having two classes.

5.2 Naïve Regressor results

The naïve regressor had similar results to the naïve classifier for the abalone dataset. The mean value for each train set was 12. The mean was used as output for any test set input value. The mean-squared-error was calculated using

$mse = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$, where N is the number of test samples, y_i is the predicted value, and $\hat{y_i}$ is the truth value.

The result was a high mean-squared-error of 88.0 for each test set (Figure 4), indicating a high bias or variance output by the naïve regressor. To avoid getting the same mse value for each test, a non-stratified approach is necessary, since the stratification ensures that the classes are even in each train and test set.

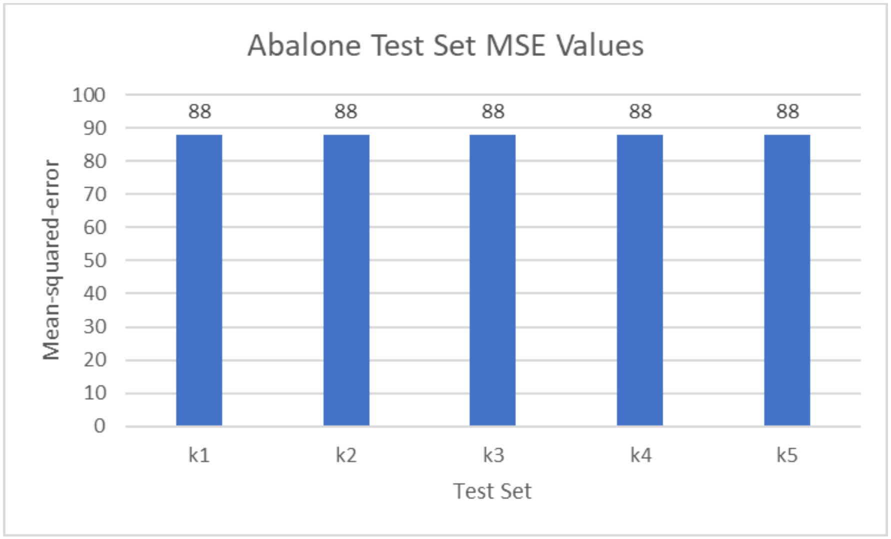


Figure 4. The MSE calculation were once again the same per test set. This could be avoided by using a non-stratified approach.

A conclusion can be made however, that the regressor has a high bias towards the mean value of the dataset. To improve the prediction output of the naïve model and lower the mean-squared-error, a new more sophisticated model needs to be developed. Perhaps an alternative to a completely new model is to add some sort of optimization as an addition to the the naïve regressor model.

The selection of a randomized prediction values in the range of all possible train target values, resulted in better results with an average mse value of 45.63 (Figure 5). Again, this is a high mse value, but it can be assumed that since the prediction values were randomly generated from all possible train set target values, the variance must be high.

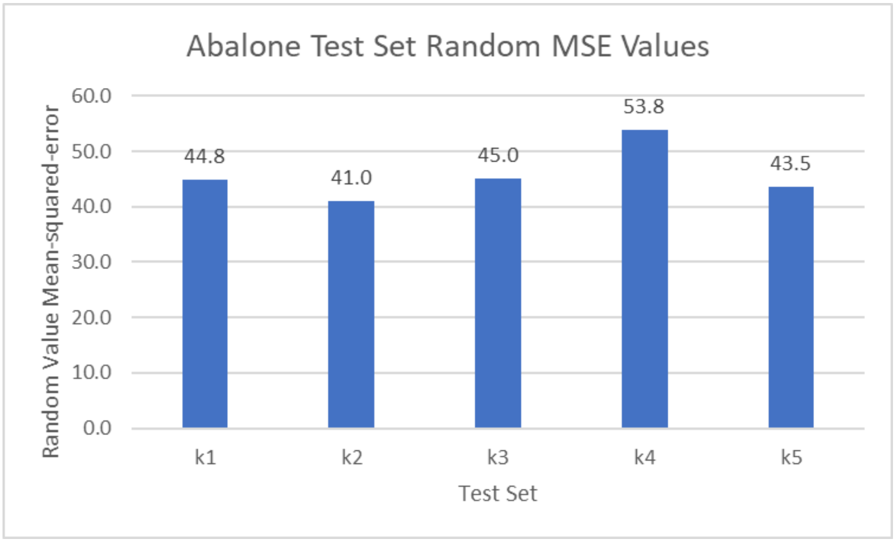


Figure 5. If the y value is randomized based on the train set target values, the MSE value goes down but is still high. It can be concluded that the bias went down, but the variance likely increased.

6 Conclusion

This project was time consuming, but worth every minute. I learned how to create a machine learning pipeline to load, preprocess, train, test, and evaluate trained models. I learned how to discretize data, standardize data, encode data, and how to create stratified k-fold train test sets. Some of these things were learned in the past using machine learning libraries like scikit learn, but this is the first time that I went through this type of machine learning pipeline approach nearly from scratch.

Aside from the implementation of the pipeline, an equally important lesson here is to think more thoroughly the experiment to help shape the hypothesis. The hypothesis was that the naïve classifier and regressor would score higher then would a randomized prediction generator. Both guesses were wrong, however. The classifier would score the same as the majority class for any perfectly stratified train/test set. The MSE score what also hurt by the high bias of outputting the average target value for any input value. Randomizing the prediction values decreased the MSE value by about half, because the bias went down, but the variance went up. If I would have thought about the experiment more carefully, I could have had a more realistic hypothesis, and instead of assuming that the code was not working correctly, I could have saved time and directed my attention elsewhere.

References

Marko Bohenec, B. Z. (1997, June). Car Evaluation Database. Avignon, France.

Paulo Cortez, A. M. (2007, December). Forest Fires. Guimaraes, Portugal.

Phillip Ein-Dor, J. F. (1987, October). Relative CPU Performance Data. Tel Aviv, Israel.

Schlimmer, J. (1984). 1984 United States Congressional Voting Records Database. Washington D.C., USA.

Waugh, S. (1995). Extending and benchmarking Cascade-Correlation. Hobart, Tasmania, Australia.

Wolberg, W. H. (1992, July 15). Wisconsin Breast Cancer Database. Madison, Wisconsin, USA.