Programming Project #01 – Constructing Machine Learning Pipeline

Jacob M. Lundeen JLUNDEE1@JHU.EDU

Department of Data Science
Johns Hopkins University
Baltimore, MD 21218

Abstract

This paper describes the process and functions used to construct a machine learning pipeline for both regression and classification data. The functions were built from scratch in the Python language, including normalization, discretization, and k-fold cross validation. These functions are generalized to handle N number of classes and regressors. A Null Model learning algorithm is used to validate that the pipeline is fully functional. Several metrics are then calculated at the end: Precision, Recall, and F_1 for classification and Mean Absolute Error (MSE), R^2, and Pearson's Correlation for regression.

1. Introduction

Learning algorithms can be very sensitive to the data that is being passed to the algorithm. Real-world data is not clean, there are missing values or categorical variables that some algorithms cannot handle. To deal with these issues, data processing pipelines are created to get the data to a usable point by rectifying most, if not all, of these issues. This project does exactly that, a data processing pipeline is built in Python to get six data sets ready for use in Machine-Learning (ML) algorithms. To verify that the pipeline is functioning correctly, the data is run through a naïve Null Model with the standard metrics output to an external file.

Problem Statement

Along with verifying that the pipeline created is functioning properly, we also want to see how well a naïve Null Model will perform for prediction. This will provide a starting point with which to compare more robust ML algorithms against in the future. My hypothesis is that the Null Model will not perform well due to its naïve structure.

Pre-Processing

First, a brief rundown on the six data sets that are used for the experiment. These data sets are all widely used in ML training and are available on the UCI website. They breakdown as follows:

- Abalone:
    - Nine attributes
    - Eight numerical, one categorical
- Breast Cancer:
    - Ten attributes

    - One class
- Car Evaluation:
    - Six categorical attributes
    - All six contain strings
- Forest Fire:
    - Thirteen attributes
    - Two are string ordinals
- House Votes
    - Sixteen attributes, one class
    - All Boolean
    - '?' are not missing values
- Machine
    - Ten attributes
    - Eight numerical
    - Two nominal with strings

Of the six data sets, all but one of the data sets did not have column headers. So, when the data set is being read in, the column names had to be hard coded. As well, only one data set had missing values (from reading the accompanying information files). To ease handling of the missing values, the NA values were automatically changed to "?" when read in. Once read in, to handle the missing values, a function was written, utilizing Pandas fillna() function, to impute the missing values with the mean value of the column.

Next up is the categorical data. ML algorithms cannot handle anything but numerical data. So any categorical data that were strings needed to be transformed. For ordinal data, they are encoded with integers that preserves the ordered nature. For nominal data, one-hot encoding (or dummy variables) is used. The nominal variables were handled with Pandas get_dummies() function, with the original variable name passed as the prefix for the one hot encoding. The cars and forest data sets, with their ordinal data, is done using dictionaries as mappers and the Pandas replace() function. For the cars data set, since every variable needed the encoding, all the mappers are put into a single list and then looped over the columns.

A discretization function was written to provide the ability to discretize data if needed. The function provides the ability to do equal width or equal frequency discretization. This is accomplished with Pandas cut() and qcut() functions. It can also discretize a single variable, or the entire data set if the user does not pass a variable name to it. It defaults to equal width and twenty bins. For the project, this function is not used but an example of it functioning is provided.

A z-score standardization function is provided for use with regression algorithms. The function is able to standardize a single variable, or an entire data set. When dealing with testing and training data sets, both data sets are standardized off the mean and standard deviation from the training set.

Lastly, a k-fold cross validation function was written. This function will divide the given data set into k-folds and then loop through the folds for the given learning algorithm. If desired, a validation set will be produced, and classification data will be stratified (to include the validation set).

Experimental Approach

The decision to use the Null Model (mean for regression and most plurality for classification) was made primarily to show that the pre-processing steps functioned as intended since the Null Model is simple to implement. A benefit of using the Null Model is that it can be used as a comparison for future algorithms, as a null hypothesis of sorts. Being that it is a Null Model, specific assumptions were made for the algorithm.

2. Results

For classification, the Abalone and Cancer data sets reached an $F_1 = 0.4$, with Precision and Recall not getting above 0.5. For regression, the MSE is large for both models, and the R^2 and correlation showed almost no statistical significance or correlation. Table 1 shows all the results from the experiments.

Table 1 - Results

	Class	**R^2**	**Pearson**	**MSE**	**Precision**	**Recall**	**F_1**
Abalone	sex_I				0.34	0.5	0.4
Breast Cancer	Class				0.32	0.5	0.4
Cars	Class				0.18	0.25	0.21
Forest	Area	0.02	-0.15	1.0			
House	Class				0.31	0.5	0.38
Machine	ERP	0.004	0.07	1.0			

From a time and resource perspective, the Null Model performed well. When doing 5-fold CV with stratification, it ran very quickly.

3. Conclusion

The Null Model, as expected, does a poor job of learning the training data and accurately predicting the test data. However, the Null Model does provide a baseline with which to compare more complicated learning algorithms to in the future.

For the pre-processing functions, overall they were not difficult to code. The cross-validation, in theory, is straightforward. But for classification, it took some trial and error to figure out how to correctly stratify the data. The reason for this was because I was getting a list of lists of data-frames and figuring out the loop logic for a general case of N classes was difficult.

Acknowledgements

I would like to acknowledge the love and support of my wife, Dr. Jordan S. Lundeen, and our wonderful family: Thea, Logan, Sydney, Mavis, Hermione, Ron, and Ginny. Without their unyielding support of my endeavors, I am not sure I would have made it this far. I would also like to thank Johns Hopkins for taking a chance on me.