Data Processing: Building a Machine Learning Pipeline

William S. Ventura
wventur1@jhu.edu
571-606-4514
Department of Computer Science
Johns Hopkins University
Baltimore, VA 21218-2682, USA

Editor: William S. Ventura

1. Introduction

Machine learning is an exciting and rapidly growing field, especially in recent years. The goal of machine learning is to be able to use existing data in conjunction with algorithms to build models that are able to give accurate predictions, classifications and recognize patterns. While this can be a really useful tool, data is scarce and there are necessary steps to take in order to get the most out of it, resulting in better performance.

The process of prepping the data so it can be used for the learning algorithms is often the most time intensive but the most important. The objective of paper is to present a pipeline that will decrease the time needed to pre-process that data. This will be significantly useful when examining data sets that contain thousands, if not millions of entries.

This paper will use and examine data sets provided by the UCI Machine Learning repository, and cover loading the data, separating the data into testing and training sets. It will also cover dealing with data sets that are incomplete, need to account for categorical variables, or need to be transformed. Once the data is prepared and ready it will be presented to the algorithms.

This pipeline uses two types of naïve learning algorithms. One is a majority class predictor and the other is a naïve regression algorithm. The majority class predictor will return the most common class label in a classification tasks, according to the feature that it is given. The naïve regression algorithm will return the average of the outputs per feature that is given. It is important to note that these naïve algorithms and for the purpose of this paper are only used as a place holder to verify our evaluation metrics. That being said, high accuracy and low error are not to be expected from these algorithms.

2. Algorithms and Experimental Methods

2.1

The initial step is to load the data, for this a *load* function was created with parameters *path*, and *header*. The *path* parameters calls the path of the file in question and the *header* parameter calls the list of header names for the respective data set. This function works by creating a temporary csv file that is used to write the column names in the first row, followed by the copy the rows from the data file into the temporary file. Finally the *load* function will use the temporary file and load it as dictionary with the column names as keys. It is important to note that from Python 3.6 and later the standard dictionary type preserves the order of the rows. Since the values in the dictionary are processed as string types, the *floatconvert* and *intconvert* functions were created to convert the values from string types into float or integer types respectively.

2.2

Sometimes data sets have missing values and while it is possible to remove the rows with missing values, it does also remove a row with information. The *impute_missing* function was created as an alternative to removing the rows with missing data. This function works by taking the parameter *dataset*, which looks at a particular feature that has missing values. The *impute_missing* function goes through the whole list and if a value is a float it will append it to a new list, if it is not of float type it will be skipped. This new list will then be used to calculate the mean of the feature column. The function will then go back to the feature column and for every entry it skipped before, it will replace the missing value with the mean calculated from the list of all known values.

2.3

In order to account for and handle categorical data, two functions were created in the pipeline. The *ordinal_encode* and *nominal_encode* functions serve as two methods to handle categorical data. The *nominal_encode* function was used to perform one-hot encoding on a certain feature. The two parameters used in this function were *data*, and *categories*. This function worked by creating a mapping of the category input, then for all the values in the data it would iterate throughout it and if the value matched the value in the mapping, then it would be a 1, if not then it will stay a 0 in the list and that list was later appended to another list, *one_hot*. This would continue for length of the category data, creating a n x m matrix where n is the length of the data and m is the length of the categories. The function, *ordinal_encode* works similarly except it does not create a matrix. The *ordinal_encode* function works by using the parameters *dataset*, and *value_map*, and parses through the whole dictionary, if the entry was found within the dictionary that was also in the value map, then entry was updated to correspond with the value mapping.

2.4

Among these functions, another created tool in this machine learning pipeline was the *discretize* function. This function worked by discretizing values in the dataset. The parameters for this function were *data*, *bins*, *type*. When using the *discretize* function, the data of interest, number of bins, and type of discretization is entered. This function allows for both equal-width and equal-frequency discretization, by enter 0 or 1 in the *type* parameter, respectively.

2.5

The *standardize* function is important when wanting to calculate the population z-scores from the training data, and then returns the standardized z-scores on the sample data (testing data).

2.6

The most important part of this pipeline tool kit would be out *cross_validation* function which is built on other functions created such as *merge*, and *fold_split*. The *merge* function is used to merge the feature lists into a tuple. The reasoning behind choosing a tuple was that when the data is split into folds in the *fold_split* function, the data was randomized and the output for each feature input would be out of order, a tuple was used to preserve that order. The *cross_validation* function uses the *fold_split* function to create a list with n-numbered folds. The parameters of the *cross_validation* function also include *algorithm*, and *evaluation_metric*, which allows the user to input which algorithm and metrics to be used.

2.7

In order to evaluate our algorithms, this pipeline included some metric functions for regression problems called *regression_metrics* and *fold_accuracy* and *total_accuracy* for classification problems. The regression metrics output the mean squared error, MSE and the mean absolute error, MAE. The fold accuracy for this pipeline is quite simple, the actual and predicted output lists are parsed. Whenever the predicted matches the actual, the correct count will increment by 1. The accuracy is calculated by final count and dividing it by the length of the actual output list and then multiplied by 100. The *total_accuracy* function is after performing the cross validation, which evaluates the accuracy for all the number of k-folds.

2.8

In order to test this pipeline, as mentioned in the introduction two types of naïve learning algorithms are used. One was a majority class predictor, *class_predictor*, and the other created is a naïve regression algorithm, *naive_regression*. The naïve regression algorithm will return the average of the outputs for the feature, while the majority class predictor, returns the most often class label for a classification task.

2.9 Data

This paper examines two different sets of data from the UCI Machine Learning Repository. The abalone data set, which a multivariate dataset that contains continuous, nominal and integer features; and the voting data set that is also a binary class and is used in this paper for a classification example.

2.9.1

In this paper the abalone data was used to evaluate the naive regression algorithm and test the data pre-processing pipeline. It was first loaded, then had the values converted into float types, for this paper the features of interest were 'Shell Weight' and 'Rings'. Using the *merge* function, they were merged into a tuple. A 5-fold cross-validation was then performed on the merged tuple list, using the *naive_regression* as the algorithm and *regression_metrics* as the evaluation metrics.

2.9.2

For the classification task, the voting dataset was loaded, and then an ordinal encoding using *ordinal_encode* was performed on the 'Class'. The features that were of interest for this classification problem were the 'physician-fee-freeze' and 'Class'. Using the *merge* function, these two features were turned into a tuple. When testing the majority class predictor a 5-fold cross validation was performed. The evaluation metrics used was the *fold_accuracy* function, and outside the parameters of the cross-validation function a *total_accuracy* evaluation function was used.

3. Results

3.1 Naive Regression Task

The mean-squared and mean-absolute error for the naive regression task on the abalone dataset was:

C:/Users/wsven/PycharmProjects/Project_1_Ventura_2022/Project_1_Pipeline.py

The is the MSE and MAE for the Cross Validation:

(MSE,MAE)

[(11.689652640826067, 2.4577478575782576), (9.234824124206641, 2.299247014952132), (10.12820681989318, 2.3769070242748067), (11.270310516691152, 2.4070701710351776), (9.660085427946486, 2.2701742622539345)]

3.2 Majority Class Predictor

The classification accuracy for each iteration in the 5-fold cross validation was:

C:/Users/wsven/PycharmProjects/Project_1_Ventura_2022/Project_1_Pipeline.py

The Classification Accuracy in this iteration is 56.32183908045977 %
The Classification Accuracy in this iteration is 62.06896551724138 %
The Classification Accuracy in this iteration is 62.06896551724138 %
The Classification Accuracy in this iteration is 64.36781609195403 %
The Classification Accuracy in this iteration is 62.06896551724138 %
The Total Accuracy is 61.379310344827594 %

4. Discussion

As mentioned before these two naive algorithms use to assess the pipeline and are very simple and not expected to have high accuracy and low error. The reason being is that these algorithms do not perform any complex calculations to reach a predicted value. In the naive regression algorithm, the mean of the feature column was calculated and that same mean was returned a prediction. This simplicity is show to affect our results where for all 5 folds in the cross-validation testing, are unacceptable values for the mean squared error and the mean absolute error.

In the case of the majority class predictor, the most often or *mode* was returned as our prediction value for this algorithm. While the classification accuracy for each iteration in the 5-fold cross-validation testing was higher than 50%, it is merely a sort of representation of how the 'Class' feature is distributed, where there could be around 60/40 split between the binary classes.

5. Conclusions

While the field of machine learning is exciting and continually growing, there are many steps as shown in this paper to prepare the data for the learning algorithms. Data pre-processing is the most time-consuming and most important part before using the data on the learning algorithms. Data is limited, and data that is already used can not be used again to retrain. It is the goal of this paper to create a pipeline that minimizes the time required to prep the data and maximize our usage of said data through cross-validation methods, so that more focus can be placed on optimizing the learning algorithms.