605.649 — Introduction to Machine Learning

Programming Project #1

1 Project Description

The purpose of this assignment is to give you an introduction to the basics steps data processing while developing a small toolkit of functions to support your tasks in future programming assignments. In this project you will pre-process six datasets and build the skeleton of a basic machine learning pipeline that you will reuse in the later projects.

In this course, you will use six datasets that you will download from the UCI Machine Learning Repository[1]. Details about these datasets can be found on the last page of these instructions. Although these datasets all differ, you will build a general set of tools that can handle any dataset. Test your functions well; your future self will thank you.

1.1 Loading Data

First determine the data structure that you intend to use to store your datasets. Next, build a function that allows you to load the data from the `csv` file on disk and store in your representation. In this function, you should perform any tasks that are uniquely specific to any one particular dataset, such as renaming column headers, performing log transformations, or substituting one value name for another.

1.2 Handling Missing Values

Real data is messy and incomplete. Luckily, our datasets are mostly complete. However, in a few cases you need to handle missing data, most commonly labeled as `?` or `NaN`. Before giving a dataset to a machine learning algorithm for learning, you must first handle this problem. Apply data imputation to impute the missing data with mean of that particular feature. Write a function that, given a dataset, imputes missing values with the feature (column) mean. Be careful, however, that sometimes it may appear there is data missing when in fact it is not. Be sure to review the `.names` file for details.

1.3 Handling Categorical Data

Categorical data represents types of data that can be subdivided into groups (e.g., color, education level). Although the majority of features in the datasets are numeric, there are many instances of categorical data. These are most often represented as strings in the dataset. Many machine learning learning algorithms cannot handle non-numeric data. For those algorithms, we first must convert this data into numbers or tokens, while preserving the information. You will encounter two types of categorical data: ordinal and nominal.

Ordinal Data: Ordinal data is data on a scale in which order matters (e.g., education level). For an ordinal relationship, it is often sufficient to map them to an integer series. For the example *education level*, the values `high school`, `bachelors`, and `masters` can be encoded as `0`, `1`, and `2`, which preserves the natural relationships between these values. Write a function that, given a dataset, encodes ordinal data as integers.

Nominal Data: Nominal data is categorical data in which the categories do not have a natural ordering (e.g., color). For this reason, it is not sufficient to convert nominal data to integers. For example, if we mapped *colors* `red`, `blue`, and `green` to `0`, `1`, and `2`, a machine learning algorithm might accidentally learn that `red < green`, which is not true since colors have no relative value to each other.

One-hot encoding is a technique to convert categorical data to numeric data. In the first step of one-hot encoding, convert the categorical values to integers. Next transform these values to a set of new binary

[1] Note that all of the data sets are also available in the content area within Blackboard

features, known as *dummies*, such that each new binary feature corresponds to a single value of integer encoding.

For example, in an encoding of *colors* `red=0`, `blue=1`, and `green=2`, a one-hot encoded representation would contain three binary features in which the value `red` corresponds to `1 0 0`, the value `green` to `0 1 0`, and `blue` to `0 0 1`. In this example, the one-hot encoding produces three binary features because there were three categorical values in the original feature.

Write a function that, given a dataset, performs one-hot encoding on nominal features. After creating the new set of dummies, make sure to drop the original nominal feature from the dataset.

1.4 Discretization

In some circumstances, it is desirable to transform real-valued data into a series of discretized values. This process of discretization uses binning to group together data within a certain range and replace it with a single-value. This results in a loss of information. Nevertheless, this step may be desirable for certain algorithms and less useful for other algorithms. You will implement discretization so that you have the option to use it (or not) in later assignments.

There are two approaches to discretization. In equal-width discretization, the range of the feature value is split into numerically equal sized bins. In equal-frequency discretization, the size of the bin may vary, but each bin contains approximately the same number of data points.

The number of bins needed may vary by dataset or even by feature. Make sure your implementation is flexible to support any arbitrary number of bins as specified by a provided argument and to support either equal-width or equal-frequency discretization, as selected by an argument. In fact, this is tunable and should be considered as such in future projects.

Write a function that, given a dataset, discretizes real-valued features into discretized features.

1.5 Standardization

Some machine learning algorithms require or prefer that feature values be on the same scale. In statistics, *z-score* standardization is used to normalize data around the feature mean. Specifically, for the feature x to be standardized, compute

$$z(x) = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of the feature value in the *training* set. It is important to remember that the statistics for z-score standardization must be learned from the training set. These are applied to both the training and test set. The test examples play no part in the calculation of μ and σ.

Write a function that, given a training and test set, performs z-score standardization.

1.6 Cross-validation

In any machine learning experiment, we must partition a portion of our data $\mathcal{X}$ for use in training the model (training set, $\mathbf{X}_{train} \subset \mathcal{X}$) and hold out a separate portion to test the model (test set, $\mathbf{X}_{test} \subset \mathcal{X}$).

Since our datasets are relatively small, you want to maximize the amount of data you have for training. In this course we employ a technique known as k-fold cross-validation. In cross-validation the entire dataset is split into k subsets or folds. In this paradigm, each of k folds is treated as the test set for a model trained on the other $k - 1$ folds. This process is repeated k times rotating which fold serves as the test set. At the end of the iteration, each example has been predicted exactly once allowing you to compare the set of predicted labels with the set of true labels for the entire dataset.

Cross validation also needs to be set up in a way to support hyperparameter tuning. This is usually handled via a third dataset (validation set, $\mathbf{X}_{val} \subset \mathcal{X}$), which is used to tune a model during the training process. $\mathbf{X}_{val}$ is a small set extracted from the data set. The basic process requires you to separate out that subset of the data (i.e., the 20%) and then performing cross validation over that subset while testing different hyperparameter values. The actual experiment is then performed via k-fold cross validation on the remaining 80% using the selected parameter values.

When splitting into folds and partitioning into train and test sets for classification tasks, you need to stratify by the y class label. This means you need to approximately equally distribute the class label among

the test sets of k folds. For example, if you have a data set of 100 points where 1/3 of the data is in one class and 2/3 of the data is in another class, and you with to perform 5-fold stratified cross validation, you do the following. First, you will create five partitions of 20 examples each. Then for each of these partitions, 1/3 of the examples (around 6 or 7 points) should be from the one class, and the remaining points should be in the other class.

Write a function that, given a dataset, partitions the data into k folds in which each fold contains a train, test, and (optionally) validation sets. For the classification data sets, also stratify the folds according to the distribution of classes defined over y.

1.7 Evaluation Metrics

In order to evaluate the efficacy of a machine learning algorithm on a particular dataset, you must compare the set of predicted values against the true output. In this course, you will report a classification score for the classification tasks and mean squared error for the regression tasks.

However, you should also consider implementing other common evaluation metrics. These include precision, recall, and the F_1 score for classification tasks, and mean absolute error, r^2 coefficient, and Pearson's correlation for regression tasks.

Write a function that, given a set of true and predicted values, calculates the appropriate, chosen evaluation metric.

1.8 Learning Algorithms

Although the purpose of this first lab is to implement the various data processing tasks and create a machine learning pipeline, you need a way to test your approach to cross-validation and to verify your evaluation metrics. For this purpose, you will create a very simple Majority predictor. This very naïve algorithm will simply return the plurality (most common) class label in classification tasks, and the average of the outputs for regression tasks. Because this algorithm does not consider any of the feature values in its decision, you should not expect high accuracy nor low error results. But this simple algorithm will enable you to test your pipeline and serve as a placeholder to be replaced by more complicated algorithms in later assignments.

Write a function that, given a train and test set, calculates the majority class output for classification and the average output for regression.

2 Project Requirements

For this project, the following steps are required:

- Download the six (6) data sets from the UCI Machine Learning repository. You can find this repository at `http://archive.ics.uci.edu/ml/`. All of the specific URLs are also provided below. All of the data sets are also available in Blackboard itself.
- Develop a toolkit of functions for pre-processing datasets, including the tasks of dropping non-feature columns, handling missing values, and encoding categorical data, discretizing real-valued features.
- Expand your toolkit of functions to include those needed to run a machine learning pipeline, including splitting data into train and test sets, k-fold cross-validation, standardization, the experiment (classification or regression), and the evaluation of your results. For cross validation, test your function with $k = 5$, but enable other values of k to be specified.
- Implement a naïve Majority predictor for both classification and regression to use in your experiments.
- Write a very brief paper that incorporates the following elements, summarizing the results of your experiments. Your paper is required to be at least 5 pages and no more than 10 pages using the JMLR format You can find templates for this format at `http://www.jmlr.org/format/format.html`. The format is also available within Overleaf. The following represents the format expected for all assignment reports in this class. Although some of these aspects may seem trivial given our naïve Majority predictor algorithm, use this opportunity to practice writing a report to these specifications

so that you can receive feedback and better understand the expectations before you undertake the more complex future assignments.

1. Title and author name
2. Problem statement, including hypothesis, projecting how you expect the algorithm to perform
3. Brief description of your pre-processing steps.
4. Brief description of your experimental approach, including any assumptions made with your algorithms
5. Presentation of the results of your experiments
6. Discussion of the behavior of your algorithms, combined with any conclusions you can draw
7. Conclusion
8. References (only required if you use a resource other than the course content.)

- Submit your fully documented code, the outputs from running your programs, and your paper.
- Create a short video that adheres to the following minimal requirements:
  - The video is to be no longer than 5 minutes long.
  - The video should be provided in mp4 format. Alternatively, it can be uploaded to a streaming service such as YouTube with a link provided.
  - Fast forwarding is permitted through long computational cycles. Fast forwarding is *not permitted* whenever there is a voice-over or when results are being presented.
  - Be sure to provide verbal commentary or explanation on all of the elements you are demonstrating.
  - Demonstrate the imputation of missing values.
  - Demonstrate the mapping of a non-binary categorical variable to one-hot coding.
  - Demonstrate discretization of the real-valued features using the method of your choice.
  - Demonstrate standardization of the training set and subsequent application to the test set.
  - Demonstrate your k-fold cross-validation procedure by displaying the size of train and test fold and the accuracy or error, as appropriate, for each individual fold.
  - Demonstrate your evaluation metrics for classification and regression by averaging your predictions over all of the folds of the dataset.

3 Project Evaluation

Your grade will be broken down as follows:

- Code structure – 10%
- Code documentation/commenting – 10%
- Proper functioning of your code, as illustrated by a 5 minute video – 30%
- Summary paper – 50%

4 List of Data Sets

1. Breast Cancer [Classification]

   Overview: This data describes characteristics of cell nuclei present in benign and malignant tumors.

   Predictor: `Diagnosis`: `M` or `B`

   Source: University of Wisconsin, 1993

   URL: `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29`

2. Car Evaluation [Classification]

   Overview: The data is on evaluations of car acceptability based on price, comfort, and technical specifications.

   Predictor: CAR: `unacc`, `acc`, `good`, `vgood`

   Source: Jozef Stefan Institute, Yugoslavia (Slovenia), 1988

   URL: `https://archive.ics.uci.edu/ml/datasets/Car+Evaluation`

3. Congressional Vote [Classification]

   Overview: This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes identified by the Congressional Quarterly Almanac.

   Predictor: Class: `democrat`, `republican`

   Source: University of California, Irvine, 1987

   URL: `https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`

   Notes: Be careful with this data set since "?" does not indicate a missing attribute value. It actually means "abstain."

4. Abalone [Regression]

   Overview: The data describes the physical measurements of abalone and the associated age.

   Predictor: Rings (`int`)

   Source: Marine Research Laboratories, Tasmania, 1995

   URL: `https://archive.ics.uci.edu/ml/datasets/Abalone`

5. Computer Hardware [Regression]

   Overview: The data describes relative CPU performance described by features such as cycle time, memory size, etc.

   Predictor: PRP (`int`)

   Source: Tel Aviv University, Israel, 1987

   URL: `https://archive.ics.uci.edu/ml/datasets/Computer+Hardware`

   Notes: The estimated relative performance `ERP` values were estimated by the authors using a linear regression method. This **cannot** be used as a feature. You should remove it from the feature set, but save it elsewhere. In a later lab, you will have a chance to see how well you can replicate the results with these two models `ERP` and `PRP`.

6. Forest Fires [Regression]

   Overview: This is a difficult regression task, where the aim is to predict the burned area of forest fires by using meteorological and other data.

   Predictor: area (`float`)

   Source: University of Minho, Portugal, 2007

   URL: `https://archive.ics.uci.edu/ml/datasets/Forest+Fires`

   Notes: The output `area` is very skewed toward 0.0. The authors recommend a log transform.