605.649 — Introduction to Machine Learning

Programming Project #1

1 Introduction

The purpose of this assignment is to give you some hands-on experience implementing a nonparametric algorithm to perform classification and regression. Basically, you will be implementing a k-nearest neighbor classifier and regressor. For this assignment, you may *not* use any libraries to implement the learning algorithms themselves (or any subsequent models implemented in this course). This includes tasks for distance calculations, editing or compressing, application of the kernel, management of the neighbors, etc. Basic data processing, however, may be done using libraries such as NumPy or Pandas.

Throughout 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 toward the end of these instructions. Although these datasets all differ, you should plan to build a general set of tools that can handle any dataset. Test your functions well; your future self will thank you.

2 Null Model Predictors

Although the purpose of this first lab is to experiment with nonparametric, nearest neighbor machine learning, you need a way to test your approach against some kind of baseline approach. For this purpose, you will create very simple "Null Model" predictors. These are very naïve algorithms that will simply return the plurality (most common) class label in classification tasks, and the average of the outputs for regression tasks. Because null models do not consider any of the feature values in its decision, you should not expect high accuracy nor low error results. But these simple models will enable you to test whatever process or pipeline you put in place and serve as a baseline for comparison. In future assignments, you may even find you can adapt them for use as a part of the more complicated algorithm being implemented.

3 Nonparametric Algorithms

In completing this assignment, be very careful with how the features are handled. Nearest neighbor methods work best with numeric features, so some care will need to be taken to handle categorical (i.e., discrete) features. The typical approach to handling numeric features is to first normalize them (either using min-max normalization or z-score normalization) and then, when finding neighbors, use a Minkowski metric of the form

$$D_{\text{num}}(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{d} |x_k - y_k|^p \right)^{1/p}$$

where classic values for p are $p = 1$ (Manhattan distance), $p = 2$ (Euclidean distance) or $p = \infty$ (max-coordinate distance).

One way of handling categorical features is with the Value Difference Metric (VDM). Note that you are not required to use VDM, but you may find this to be a useful method. Note further that VDM only works with classification. If you have categorical features in regression, you will need to apply a different approach (such as one hot coding).

For VDM, let $f(\mathbf{x})$ be a feature of data instance $\mathbf{x}$, where $f(\mathbf{x}) \in \{v_1, \ldots, v_k\}$. Let $C_i = \#\{f(\mathbf{x}) = v_i\}$ (i.e.,, the number of times feature f takes on value v_i in the training set) and $C_{i,a} = \#\{f(\mathbf{x}) = v_i \wedge \text{class} = a\}$ (i.e., the number of examples in the training set belong to class a that also have feature value v_i for feature f). We then define the "distance" between feature value v_i and feature value v_j as

$$\delta(v_i, v_j) = \sum_{a=1}^{\text{num classes}} \left| \frac{C_{i,a}}{C_i} - \frac{C_{j,a}}{C_j} \right|^p .$$

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

where p is the exponent that defines the associated norm similar to an L_p norm. Typically, you will set $p = 1$ or $p = 2$ (tune this). Then the distance between two examples $\mathbf{x}$ and $\mathbf{y}$ is

$$D_{\text{cat}}(\mathbf{x}, \mathbf{y}) = \left(\sum_{k=1}^{d} \delta(\mathbf{x}_k, \mathbf{y}_k) \right)^{1/p}.$$

A question that is often asked is with respect to how to combine numerical and categorical distance. To address this, we assume that we do not take the pth root. We justify this by observing that the pth root does not change the ordering of the neighbors, so we can still pick the k nearest correctly. In this case, we calculate distances based on the relevant feature type while using the same value of p for each and then just add the two distances together:

$$D(\mathbf{x}, \mathbf{y}) = D_{\mathbf{num}}(\mathbf{x}, \mathbf{y}) + D_{\mathbf{cat}}(\mathbf{x}, \mathbf{y})$$

In this project, you will also be implementing your choice of either the edited nearest-neighbor or the condensed nearest-neighbor algorithms. The basic approach to edited nearest-neighbor starts with the complete training set, and the editing process can be summarized as follows:

1. Consider each data point individually.
2. For each data point, use its single nearest neighbor to make a prediction.
3. If the prediction is (your choice: correct or incorrect, but don't do both in the same algorithm), mark the data point for deletion.
4. Stop editing once performance on the held out 20% starts to degrade.

The basic approach to condensed nearest-neighbor starts with an empty condensed training set, and the condensing process can be summarized as follows:

1. Add the first data point from the training set into the condensed set.
2. Consider the remaining data points in the training set individually.
3. For each data point, attempt to predict its value using the condensed set via 1-nn.
4. If the prediction is incorrect, add the data point to the condensed set. Otherwise, move on to the next data point.
5. Make multiple passes through the data in the training set that has not been added until the condensed set stops changing.

As a final comment, it's important to note that, when editing or condensing for the regression data sets, a "correct" prediction is determined by whether or not the prediction falls within some ϵ of the ground truth value. You need to tune this ϵ. Note that the best way to consider possible values is by examining the range of target values possible in the training set and decide on an appropriate range of error thresholds to test from there. It is also important to point out that this ϵ is used *only* for the editing or condensing process. It is *not* used for determining regression performance. For that, you would use the appropriate loss function, which for regression is mean squared error.

Also in this project (and all future projects), the experimental design we will use 5×2 cross-validation. Note that stratification is not expected for the regression data sets, but you should be sure to sample uniformly across all of the response values (i.e. targets) when creating your folds. One approach for doing that (that's not particularly random) is to sort the data on the response variable and take every other point for a given fold.

4 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 cross-validation. In cross-validation the entire dataset is split into subsets or folds. There are two versions of cross-validation that are generally used. The first is called k-fold cross-validation. Here, you subdivide the data into k equally-sized subsets of the data. You train on $k-1$ of the folds and test on the remaining, held out fold. You then rotate through the data, conducting k experiments where each experiment holds out a different one of the folds. It is customary to do 10-fold cross-validation.

The other method is called $k \times 2$ cross-validation. Here, you split the data in half at random, creating two folds. For one experiment, you train on the first half and test on the second half. For a second experiment, you train on the second half and test on the first half. You repeat this k times, yielding $2k$ experiments. It is customary to do 5×2 cross-validation, and that is what you will be asked to do in these assignments.

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 distribute the class label approximately equally 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 wish to perform 10-fold stratified cross-validation, you do the following. First, you will create ten partitions of 10 examples each. Then for each of these partitions, $1/3$ of the examples (around 3 points) should be from the one class, and the remaining points should be in the other class.

To make sure the overall experimental design is clear, we outline the process as follows. Here we focus on 5×2 cross-validation, since that is the type of cross-validation you are being asked to implement. First, cross-validation needs to be set up in a way to support hyperparameter tuning. This is usually handled via a third dataset (tuning set, $\mathbf{X}_{tun} \subset \mathcal{X}$), which is used to tune a model during the training process. $\mathbf{X}_{tun}$ is a small set extracted from the data set. The basic process requires you to separate out that subset of the data (in particular, 20%) and then evaluating a configuration of hyperparameters using that subset while testing different hyperparameter values. The actual experiment is then performed via cross-validation on the remaining 80% using the selected hyperparameter values.

1. Divide your data into two parts: 80% will be used for training and 20% will be used for tuning, pruning, or other types of "validation."
2. Do the following five times:
   (a) Divide the 80% into two equally-sized partitions. For classification, make sure you stratify so that the class distributions are the same in the two partitions.
   (b) For hyperparameter tuning, construct two models using a candidate set of parameters. Each model will be trained with one of the halves and tested on the held-out 20%.
3. Average the results of these ten experiments for each of the parameter settings and pick the parameter settings with the best performance.
4. Do the following five times.
   (a) Divide the 80% again into two equally-sized partitions (stratified).
   (b) Train a model using the tuned hyperparameters on the first half and test on the second half.
   (c) Train a model using the tuned hyperparameters on the second half and test on the first half.
5. Average the results of these ten experiments and report that average.

5 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 Canvas itself.
- Implement what you need for some kind of pipeline, such as z-score normalization, the management of categorical features, and a cross-validated experimental design.
- Implement an experiment to test different nonparametric learning methods.
  - Implement k-nearest neighbor and be prepared to find the best k value for your experiments. To do this, you must tune k and explain in your report how you did the tuning.
  - Implement either edited or condensed k-nearest neighbor. See above with respect to tuning k; however use $k = 1$ during the editing/condensing process. Only use the tuned k when doing a prediction using your reduced data set. On the regression problems, you should define an error threshold ϵ to determine if a prediction is correct or not. This ϵ will also need to be tuned.
  - For classification, employ a plurality vote to determine the class (i.e.,, return the class with the highest number of votes among the neighbors; if there's a tie, choose randomly). For regression, apply a Gaussian (radial basis function) kernel to make your prediction. In particular, you should use the kernel $K(\mathbf{x}, \mathbf{x}_q) = \exp(-\gamma||\mathbf{x} - \mathbf{x}_q||_2)$, where $||\cdot||_2$ indicates the L2 norm (i.e. Euclidean distance in this case). You will need to tune the bandwidth $\gamma = 1/\sigma$.
  - Run your algorithms on each of the data sets. These runs shall be done with 5×2 cross-validation so you can compare your results statistically. You should use classification error to evaluate your classifier and mean squared error to evaluate your regressor. Other metrics are permitted, but you must explain and justify your choice if you don't use the above.
- Write a lab report that incorporates the following elements, summarizing the results of your experiments. Your report 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 organization expected for all assignment reports in this class.
  1. Title and author name
  2. Brief, one paragraph abstract summarizing the results of the experiments
  3. Problem statement, including a testable hypothesis (or hypotheses), projecting how you expect the algorithm to perform (make sure to specify how it will be tested)
  4. Brief description of any pre-processing steps you needed to complete
  5. Brief description of your experimental approach (basically the 5×2-cv design), including any assumptions made with your algorithms
  6. Presentation of the results of your experiments on all data sets using tables or graphs as appropriate
  7. Discussion of the behavior of your algorithms, combined with any conclusions you can draw
  8. Conclusions you can draw from the experiments
  9. References (only required if you use a resource other than the course content or textbook.)
- Create a short video that is no more than 5 minutes long, provided in mp4 format (or uploaded to a streaming service such as YouTube with the link provided), and adheres to the following minimal requirements: 1) 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. The splicing together of segments covering the necessary elements is encouraged. 2) Be sure to provide verbal commentary or explanation on all of the elements you are demonstrating, but don't waste time describing the code. Demonstrate the following functionality in your video:

  - Numerical data normalization through z-score standardization, showing the derivation of μ and σ^2.
  - Categorical data transformed either to be one-hot coded or to incorporate some kind of label (integer) coding.
  - Construction of separate subsets of the data to support 5×2 cross validation (i.e., the held-out 20% and then two folds of 40% each).
  - The calculation of your distance function(s).
  - The calculation of your kernel function.
  - An example of a point being classified using k-nn. Show the neighbors returned as well as the point being classified.
  - An example of a point being regressed using k-nn. Show the neighbors returned as well as the point being predicted.
  - An example being edited from the training set if you implemented edited nearest neighbor, or an example being added to the training set if you implemented condensed nearest neighbor.

- Submit your fully documented code, the video demonstrating the required functionality, and your paper.

6 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%

A List of Data Sets

Use the following information on all of the assignments except the Reinforcement Learning assignment. Note that you are expected to read and understand all of the associated ".names" files for these 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`

   Notes: The classes are indicated with the values "2" and "4", but these are just two discrete classes and should be treated as such. The features are all values in the range $[1, 10]$ and should be treated either as numeric or ordinal features. The Sample code number should be discarded. Missing values may be imputed, or the examples with missing features may be dropped from the data set.

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`

Notes: All of the features should be treated as if they are nominal even though, technically, they are ordinal. This is a four-class problem.

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." This is a two-class problem.

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`

Notes: Observe that, even though the ".names" file lists 29 classes, this is not to be treated like a classification problem. You are trying to predict (regress) the number of rings. All of the features are real-valued except for sex. You may choose between one-hot coding this feature, or discarding it.

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 features for vendor name and model name are not useful fo regression, so these features should be discarded. All of the remaining features are real-valued (numeric). 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. It might be interesting to test your regressors against this value as part of your experimental results. Even though a class distribution is provided, this is not a classification problem and should not be treated as such.

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 of the form $\hat{r}(\mathbf{x}) = \ln(r(\mathbf{x}) + 1)$, where $r(\mathbf{x})$ is the target value associated with example $\mathbf{x}$. The spatial coordinates should be treated as numeric features. The month and day features are ordinal and should use label encoding; however, be careful in computing distance since these are "cycle" features (e.g., in the absence of date information, the distance between December and January is only 1.0, not 11.0).

B Machine Learning Pipeline (Optional)

The following outlines components that you may regard as useful to include in any data processing and machine learning pipeline you wish to implement. Ultimately, much of the functionality listed below will be needed for completing these projects; however, we decided to leave things up to you, the programmer, to determine when and how you needed to incorporate these elements. An example of the pieces of such a pipeline is shown in Figure 1.

B.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.

B.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` (Note, however, that the Congressional Voting dataset assigns a meaning to `?`, instead of using it to indicate missing values). 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 the mean (continuous feature) or mode (discrete feature) of that particular feature.

B.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 still several 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 must first 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*,

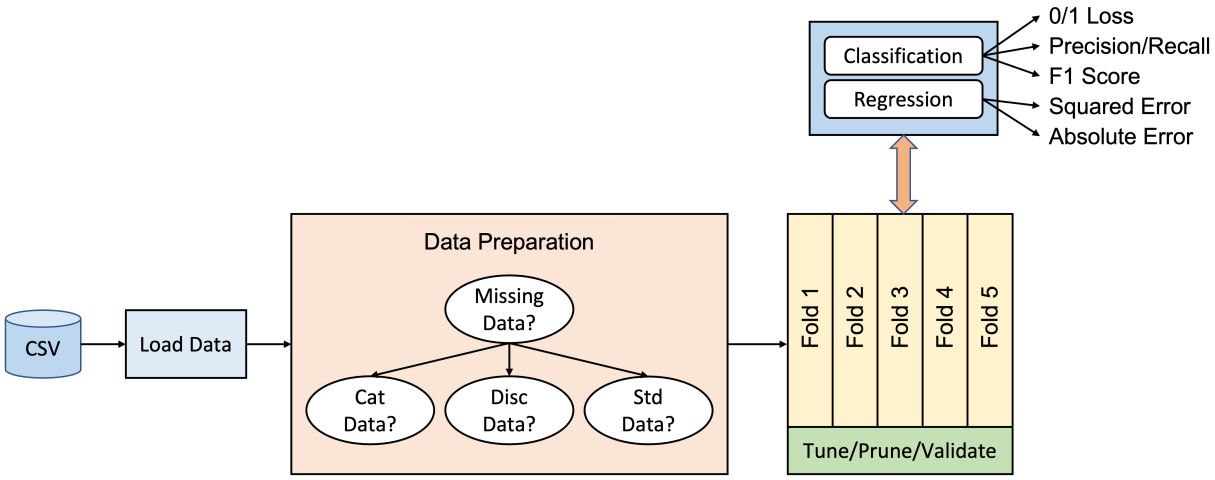

Figure 1: Machine Learning Processing Pipeline

the values `high school`, `bachelors`, and `masters` can be encoded using label encoding as `1`, `2`, and `3`, which preserves the natural relationships between these values.

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 (called label encoding). 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 nominal/categorical data to numeric data. As the first step of one-hot encoding, convert the categorical values to integers. Next transform these values to a set of new binary 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. Note that you are permitted to use dummy encoding, which is available through Pandas.

B.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.

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.

B.5 Normalization

Some machine learning algorithms require or prefer that feature values be on the same scale. There are two different ways to accomplish this: z-score standardization and min-max normalization.

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 subsequently applied to both the training and test set. The test examples play no part in the calculation of μ and σ.

Min-max normalization is the process whereby the feature values are rescaled to fit within the range of zero (0) to one (1). To do this, find the minimum and maximum value for the feature in question. Then rescale as follows:

$$\hat{x}_i = \frac{x_i - \min_i}{\max_i - \min_i}.$$

where $\min_i$ is the minimun value for feature x_i in the training set and $\max_i$ is the maximumn value for feature x_i in the training set.

B.6 Evaluation Metrics

In order to evaluate the efficacy of a machine learning algorithm on a particular dataset, you need to compare the set of predicted values against the true output. In this course, you will report a classification score such as accuracy 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.