1

Project: Linear Regression

University of Oregon, DSCI/CIS 372

This project will take a lot of time and work. Start early.

Project Description

In this project you will:

1. Implement linear models.
   (a) You will write code which reads in data about Abalones from a CSV file.
   (b) You will do some preprocessing of the Abalone data.
   (c) You will construct a linear model which predicts the age of an Abalone.

   The goal of this task is to get you very familiar with SciKit-Learn, a popular library for data science.
2. You will learn about the MNIST dataset, a classic benchmark dataset in the field of machine learning.
   (a) You will use the machinery you developed in the linear regression project to classify images in the MNIST dataset.
   The goal is to (1) provide a baseline of accuracy with a method you are familiar with and (2) ensure that your pipeline for reading in training and testing data is working so that we can use MNIST in our next project.

Task 0 | Set Up

This section is only for setting up your environment and does not have a deliverable.

- Your submitted code will be run in a terminal. You may use whatever development environment which works for you (e.g., PyCharm), but make sure your code runs from a regular terminal before submission.
- Make sure you are running Python 3. You can verify this by executing the command in the terminal:

  `python -version`

  Another approach is to always run the command `python3`.

  WARNING: If you use the command `python3`, do so consistently. You must also install libraries used by `python3` with the command `pip3`.

  DO NOT CONTINUE UNTIL YOU ARE RUNNING PYTHON 3
- Install the required libraries for this project by running the following commands:

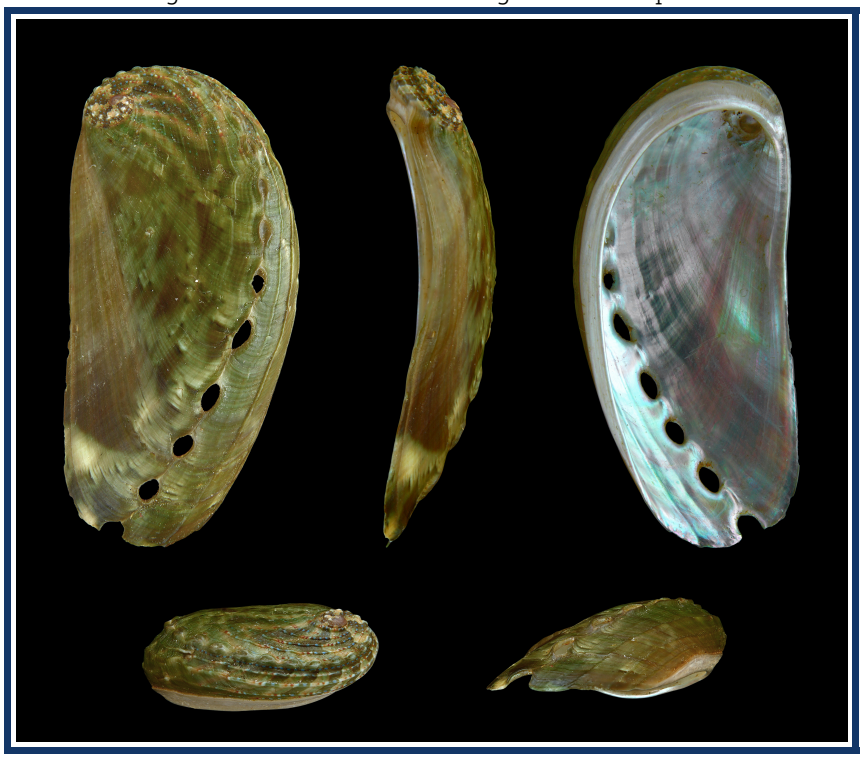
  ```
  pip install scikit-learn
  pip install scipy
  pip install numpy
  pip install matplotlib
  ```

  Or, if you are running python via Anaconda:

  ```
  conda install scikit-learn
  conda install scipy
  conda install numpy
  conda install matplotlib
  ```

Task 1 | Implement linear models: Abalones

Figure 1.1: Abalone shell. Image from Wikipedia.



1.1 About Abalones

An abalone is a type of mollusk. The shell of one can be seen in the Figure 1.1.

You will be working with a dataset initially sourced from `https://archive.ics.uci.edu/ml/datasets/Abalone`. From the description:

> [This dataset is about] predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope – a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age.

The collected data has the attributes explained in Table 1.1.

Table 1.1: Description of measured Abalone characteristics.

Name	Data Type	Measurement Unit	Description
Sex	nominal	–	F (Female), M (Male), and I (Infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	Perpendicular to length
Height	continuous	mm	With meat in shell
Whole weight	continuous	grams	Whole abalone
Shucked weight	continuous	grams	Weight of meat
Viscera weight	continuous	grams	Gut weight (after bleeding)
Shell weight	continuous	grams	After being dried
Rings	integer	–	+1.5 gives the age in years

Some preprocessing of the data has already been done:

> From the original data examples with missing values were removed (the majority having the predicted value missing), and the ranges of the continuous values have been scaled for use with an ANN (by dividing by 200).

1.2 Preprocessing

You will do some additional preprocessing of this data before we begin. Please do the following steps in-order.

1. Write a simple python program to convert nominal data into numeric 0 or 1 values based on mutually exclusive categories. These values are called "dummy values."

   Truncated versions of "before" and "after" should look like this (first line header is optional):

   Before

```
Sex,Length,Diameter,Height,WholeWt,ShuckedWt,VisceraWt,ShellWt,Rings
M,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15
M,0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,7
F,0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,9
M,0.44,0.365,0.125,0.516,0.2155,0.114,0.155,10
I,0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,7
I,0.425,0.3,0.095,0.3515,0.141,0.0775,0.12,8
F,0.53,0.415,0.15,0.7775,0.237,0.1415,0.33,20
F,0.545,0.425,0.125,0.768,0.294,0.1495,0.26,16
M,0.475,0.37,0.125,0.5095,0.2165,0.1125,0.165,9
F,0.55,0.44,0.15,0.8945,0.3145,0.151,0.32,19
...
```

   After

```
IsFemale,IsMale,IsInfant,Length,Diameter,Height,WholeWt,ShuckedWt,VisceraWt,ShellWt,Rings
0,1,0,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15
0,1,0,0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,7
1,0,0,0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,9
0,1,0,0.44,0.365,0.125,0.516,0.2155,0.114,0.155,10
0,0,1,0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,7
0,0,1,0.425,0.3,0.095,0.3515,0.141,0.0775,0.12,8
1,0,0,0.53,0.415,0.15,0.7775,0.237,0.1415,0.33,20
1,0,0,0.545,0.425,0.125,0.768,0.294,0.1495,0.26,16
0,1,0,0.475,0.37,0.125,0.5095,0.2165,0.1125,0.165,9
1,0,0,0.55,0.44,0.15,0.8945,0.3145,0.151,0.32,19
...
```

Sidenote: There are other coding schemes. Another commonly seen coding scheme is effect coding. In effect coding we would keep "Sex" as one column and map each nominal value to a specific number. For example, we might assign F=0, M=1, I=2; or F=-1, M=1, I=0; or F=-300, M=π, I=$\sqrt{2}$; or any other somewhat arbitrarily chosen values. There are mathematical/statistical tradeoffs to doing effect coding vs dummy coding, as well as which specific values to map to in effect coding. You can choose to explore effect coding more fully in Project 4.

2. Once you have applied dummy coding to the "Sex" variable, randomly assign each row of data in `abalone.csv` into either a training data set or a testing data sets. 90% of the rows in `abalone.csv` should go into the training data set (~3760 lines) and 10% of the rows should go into the testing dataset (~417 lines). Save these data sets as `abalone_train.csv` and `abalone_test.csv`, respectively.
3. Save the script you used to do this preprocessing as `preprocess00.py`

1.3 Write Linear Regression Code

There are two files with starter code in them, `main.py` and `exp00.py`.

The file `main.py` is brief: it simply is the launching point for your work. Namely, it runs "Experiment" files.

The file `exp00.py` is the first of several "Experiment" files you will be working with. Each experiment file might be analogized to a Jupyter notebook: it contains the code which loads in data, processes the data, runs some sort of model or analysis on the data, and reports the outcome.

`exp00.py` contains 5 methods:

1. load_train_test_data Loads the train and test CSV files you created in § 1.2.
2. compute_mean_absolute_error Computes the MAE
3. compute_mean_absolute_percentage_error Computes the MAPE
4. print_error_report Prints to the console an error report when given a trained model, and data to predict on. (Calls 2 and 3.)
5. run This is the most important method. It contains the following parts:
   (a) Print statement with the start time and name of the experiment being run.
   (b) Loads the data. (Calls 1.)
   (c) Trains a model. **You will write this code (about 2 lines).**

(d) Evaluates the model. (Calls 4.)
(e) Print statement with the end time and duration of the experiment.

Experiment files will all follow this basic format: auxiliary methods which all center around a single reproducible run method. Within the run method the same pattern of:

(a) load data (and any more preprocessing), (b) load and train a model, and (c) evaluate the model

will be consistent in this class.

> It is a software engineering best practice that, if you are copying and pasting boilerplate code within a project, you're doing something wrong. Namely, the code should be made its own method/utility class and then you should invoke the code once.
>
> This guidance is generally true. However, in model creation and development there is a tinkering process which (I find) requires this rule to be broken. Namely, you will discover that you get to process X, decide that X should be changed to Y, and then later changed to Z. You will want to undergo this process dozens of times for a given new model: preprocessing changes; model changes; evaluation changes; etc.
>
> The challenge is, if code is modularized at this point in the development process, you might decide to circle back to X and discover that it no longer works (or "works" but gives different output than the first time) due to the changes you made along the way.
>
> While version control software like git might be helpful, I find the easiest and most conceptually simple way to evolve while still maintaining ease of reproducibility for old code is to create a series of experiments which are quite similar to each other. Experiments which are as self-contained as possible and subsequently copied at each iteration while still following the same conceptual pattern.
>
> Once a model has been finalized (after anywhere from 1 to 100+ experiments), then one can apply the regular rules around coupling+cohesion and other software design best practices.

Your task is twofold:

1. **Part A** Search in `exp00.py` for the comments with the text #Fix This Line. Modify the line(s) around the comments to instantiate and train a linear regression model from the library sci-kit learn.
2. **Part B** Report the evaluation metrics in a report called `report.pdf` (You can use any document creation software like Word/Google Docs/LaTeX/etc., just make sure it's converted to a PDF on submission.)

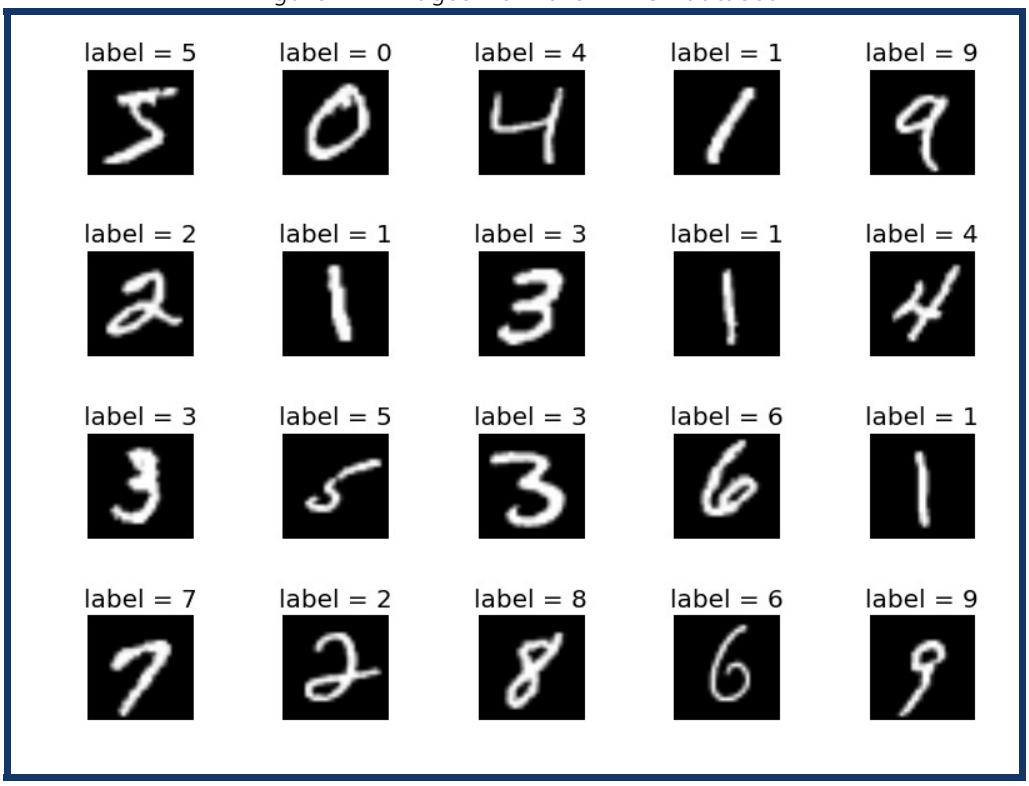
Task 2 | Implement linear models: MNIST

Edit `main.py`. Modify all references to class `Experiment00` in `main.py` to now refer to the class `Experiment01` in file `exp01.py`.

2.1 Part 1: About the MNIST Dataset

The MNIST dataset is a collection of handwritten digits and their associated values. The handwritten digits have been converted to pure grayscale, and the colors inverted so that the background is in black (coded as 0 values) while the strokes are in white (coded as non-0 values).

Figure 2.2: Images from the MNIST dataset.



We've provided the data for you in two CSV files: `mnist_training.csv` and `mnist_testing.csv`.

2.1.1 Visualizing a digit of the dataset

Your first task is to create a working pipeline which reads in the data for a digit, reshapes the data, and visualizes it using matplotlib.

Specifically, you will be working with two starter code files: `main.py` and `exp01.py`

For this task, the experiment file contains starter code. You will build on the starter code. The file `exp01.py`, once it's finished, will do the following:

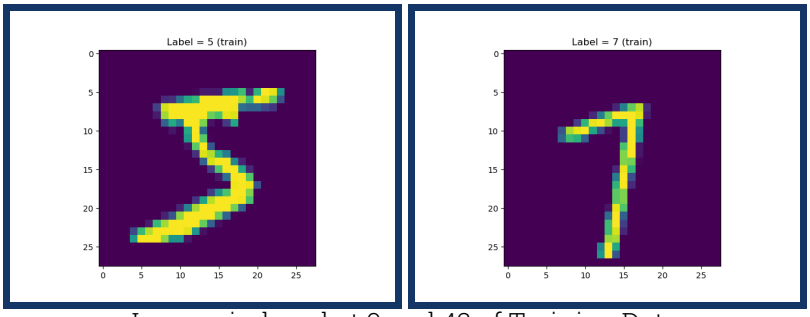
- Loads MNIST data from the provided CSV files.
- Reshape the data. Each line of the CSV file is currently of the form: $sample_i =$ `digit_the_image_represents, pixel1, pixel2, ···, pixel784`.

  You will reshape this data so that each line is transformed (conceptually) to an ordered pair:

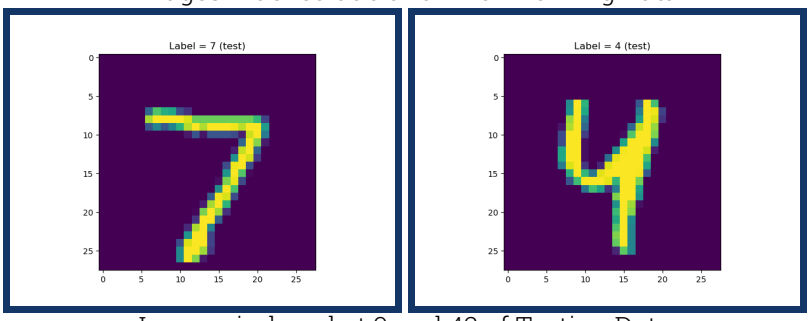
$$sample_i = (\texttt{digit\_the\_image\_represents}, \begin{bmatrix} pixel1 & pixel2 & \cdots & pixel28 \\ pixel33 & \cdots & & \\ \vdots & & \ddots & \\ & & & pixel784 \end{bmatrix})$$

  We've used 1-based indexing here for clarity, but 0-based indexing is how you'll want to implement it in code.

- Visualize samples loaded from the training data at indexes $0, 42, 156$ and samples $0, 42, 542$ from the testing data using matplotlib. The label associated with the data should be included in the image. You can see an example here for images 0 and 42 of the training data, and 0 and 42 of the testing data:



Images indexed at 0 and 42 of Training Data



Images indexed at 0 and 42 of Testing Data

You are expected to fill out the unwritten components of `exp01.py`. Search for the text # Fix & Complete me and # Complete me to find the methods which still need work.

You should run your code by executing `python main.py` on the command line.

Once you have visualized samples $0, 35, 56$ of the training data and samples $0, 36, 42$ of the testing data, save images 56 of the training data and 42 of the testing data. These images are to be included in a report called "report.pdf." The report can be created in Microsoft Word, LaTeX, Google Docs, or whatever document processing software you prefer.

2.2 Part 2: Use linear regression on MNIST

In this task, you will use the machinery you developed in the linear regression project to classify images in the MNIST dataset. The goal is to (1) provide a baseline of accuracy with a method you are familiar with and (2) ensure that your pipeline for reading in training and testing data (which should have mostly been accomplished in §2.1.1) is working.

Edit `main.py`. Modify all references to class `Experiment01` in `main.py` to now refer to the class `Experiment02` in file `exp02.py`.

`Experiment02` does the following:

- It load the data (You will implement this. While we often want to avoid boilerplate and copy-pasting code, in this instance we suggest doing just that: copy-paste the code you wrote to load the training and testing data in §2.1.1). Search for the text # Fix & Complete me (copy from exp01.py) to find the place to implement this.
- It trains a linear model using the scikit-learn library (This is done for you, and should serve as a model for how you should structure your code when you implement neural networks later on.) Search for the text # Fix this line (and note that there are two instances of lines to fix, both of which can largely be copy-pasted from exp00.py).
- It outputs predictions when given data.
- It reports the mean error rate of the trained model on both training and testing data.

  Mean error rate for a set of pairs $(trueVal_1, predVal_1), \cdots, (trueVal_n, predVal_n)$ is defined as

$$\frac{1}{n}\sum_{i=0}^{n} trueVal_i \quad != \quad int(round(predVal_i))$$

  You will implement this, too. Search for the text # Complete me!.

Report the mean error rate on both the training and testing data for this model in "report.pdf."

Task 3 | What to Submit

You will submit two files:

`runnable_code.zip` and `reports_and_logs.zip`

3.1 Code

Put any executable code into a zip file called `runnable_code.zip`. The goal is that another human can unzip this folder and successfully run your code. You may put any special instructions or notes about how to run your code in a `readme.txt` file, also placed within `runnable_code.zip`.

This zip folder should contain at least the following:

- preprocess00.py
- exp00.py
- exp01.py
- exp02.py
- main.py

3.2 Reports and Logs

Put any reports, logs, images, and output into a zip file called `reports_and_logs.zip`. This zip folder should contain at least the following:

- report.pdf, which includes the error metrics for all models as well as the requested MNIST images.

3.3 Upload to Canvas

Upload two separate files to Canvas: `runnable_code.zip` and `reports_and_logs.zip`.

Appendix A | Grading Criteria

This project will be primarily assessed along three axes:

- **Validity**

  Programs should conform to specifications stated in the project description.

  Some components of the project will have example output provided. Your program's output must match the provided output for full points. If your output does not match identically for deterministic code (or within a reasonable margin for non-deterministic/random-based code) then expect a heavy penalty.

- **Style, Documentation, and Readability**

  Programs should be easy to read and understand. Coding well is writing well. Writing well is thinking well.

  To help us write quality code, we will use Pylint. Pylint is used in industry (Google, Amazon, etc.) to enforce Python's coding standard. Pylint also provides messages about style / readability / documentation flaws. You should run Pylint before submitting your work. Pylint assigns a score out of 10 to the code it processes. It can be very, very difficult to get a 10. We will consider 6/10 to be "full credit" for most work.

- **Design and Fluency**

  When one is learning a natural language like English, Spanish, Chinese, etc., we often go through a phase where our language is "technically correct," but may not seem right to a fluent speaker of the target language – in fact, our spoken or written speech as an (advanced) learner may still seem jarring or just plain weird at times to a fluent speaker. The same principle holds true for code.

  This lack of fluency we all sometimes experience when learning new topics isn't to be confused with creativity or pushing the limits in new directions. Rather, it's just part of acclimating to the norms of a new subject area and undergoing the learning process of becoming an expert in a particular skill.

  Coding is a highly creative and highly controlled processes. In making decisions in how to tackle the objectives outlined in the project tasks, you should keep in mind the following principles in order to maximize fluency:

  - Programs should be written with regard to efficiency of both human development time and code space.
  - Programs should be "elegantly" written. The "keep it simple" principle comes to mind.
  - Programs should consist of small, coherent, independent methods.
    I follow the two inches rule-of-thumb: if a method is longer than two inches on the screen, it should probably be broken up into more methods.

  Design and fluency is a category which is difficult to automatically grade, yet incredibly important to get feedback on. Come and talk in office hours if you are routinely missing points in this category.

Rubric

Abalones (100 points possible)

Preprocessing

preprocess00.py code validity: _____ of 35

preprocess00.py code readability: _____ of 6

preprocess00.py code fluency: _____ of 4

Linear Regression

exp00.py code validity: _____ of 35

exp00.py code readability: _____ of 6

exp00.py code fluency: _____ of 4

Reporting

Metrics reported in report.pdf: _____ of 10

Subtotal: _____ of 100

MNIST (100 points possible)

Part 1 - Visualization (50 pts)

exp01.py code validity: _____ of 30

exp01.py code readability: _____ of 6

exp01.py code fluency: _____ of 4

Images included in report.pdf: _____ of 10

Part 2 - Linear regression (50 pts)

exp02.py code validity: _____ of 30

exp02.py code readability: _____ of 6

exp02.py code fluency: _____ of 4

Metrics included in report.pdf: _____ of 10

Subtotal: _____ of 100

Total: _____ of 200