CSCI1410 Fall 2018
Assignment 6: Supervised Learning

Code Due Monday, November 19
Writeup Due Thursday, November 26

1 Introduction

George has gathered a lot of information about who has been stealing his Dr Pepper. He needs your help with classifying this information.

In this assignment, you will implement k-nearest-neighbors (KNN) and parts of linear regression in abstract. Then, you will use linear regression to make predictions on a real dataset.

2 Code Walkthrough

You are given 5 code files. Below is a high-level overview of each. View the docstrings and comments of each file for more detailed information.

- `supervisedlearner.py` contains `SupervisedLearner`, the abstract superclass of the other classes that you will implement for this assignment, `KNNClassifier` and `RegressionLearner`.

  In the `SupervisedLearner` class, you will implement `compute_features`.

- `knn.py` contains the `KNNClassifier` class, for which you will implement `train`, `predict`, and `evaluate`.

- `regression.py` contains the `RegressionLearner` class, for which you will implement `predict` and `evaluate`.

- `bicycleregression.py` contains support code that will be useful when you apply linear regression to a real dataset. In this file, you will fill in the stencil code for `produce_regression_model`.

- `perceptron.py` contains the `Perceptron` class, for which you will implement `train`, `predict` and `evaluate`.

3 Part 1: Generic Supervised Learning

3.1 Compute Features

First, you should implement `compute_features`, which converts a data point into a feature vector. This method will be useful for both the KNN and regression portions of the assignment and, more generally, in any machine learning approach that involves extracting features from data using hand-picked feature functions. You can find `compute_features` and its specifications in `supervisedlearning.py`. We advise you to unit test your implementation of the `compute_features` function, and ensure that its runtime is less than a minute.

3.2 K Nearest Neighbors (KNN)

In `knn.py`, you will write code that can be used to apply a KNN approach to any classification problem. You should implement `train`, `predict`, and `evaluate`. The docstrings for these methods give I/O specifications and some instructions on how to implement them. You may add any other code or instance variables to the class as you see fit.
We advise you to unit test your implementation of `train`, `predict`, and `evaluate` functions, and ensure that the runtime for each of these functions is less than a minute.

Note: In the stencil code, we use the phrase **anchor point** to refer to one of the points that is stored in a KNN approach. A KNN classifier classifies a data point x by taking a plurality vote among the K anchor points that are closest in feature space to x.

3.3 Linear Regression

In `regression.py`, you will write generic code that can be used for any regression problem. We have already written `train` for you. It uses the matrix-based approach that you learned in lecture. It is up to you to write `predict` and `evaluate`.
We advise you to unit test your implementation of `predict`, and `evaluate` functions, and ensure that the runtime for each of these functions is less than a minute.

3.4 Perceptron

In `perceptron.py` you will write code to `train`, `predict` and `evaluate` a perceptron. In general, a perceptron is an algorithm for learning a binary classifier, using the following function:

$$f(x) = \begin{cases} 1 & \text{if } \boldsymbol{w}.\boldsymbol{x} + \boldsymbol{b} > \mathbf{0} \\ 0 & \text{otherwise} \end{cases}$$

where **x** is the datapoint, **w** is the set of weights that the perceptron learns and **b** is a bias term.You can either use a bias term in the training equation or could append a constant bias term in the dataset. The dataset that we will use to test your code has a bias term included in it, so feel free to consider $\boldsymbol{b} = \mathbf{0}$.

3.5 Testing Your Code

You will find that it is nontrivial to verify that your code is correct. We recommend that you test your code by creating small, phony datasets and making sure that your code behaves as you expect it to on them.

4 Part 2: Regression on Bicycle Dataset

4.1 Problem

Now that you have written a basic library for performing linear regression, it's time to put it to use! You are given a portion of a real dataset about bike rentals. Each data point contains the following information, in order:

1. **hour**: the hour of a day in military time, divided by 24.
2. **workingday**: this will be 0 on a weekend or holiday, 1 on a working day.
3. **atemp**: the normalized feeling temperature in Celsius

Each data point is labeled with the number of bikes sold at that hour. You will create a regression model that predicts the number of bikes sold, given those three factors.

You are given the dataset in the form of a saved 2D numpy array, `student_data.npy`. Each row of this array contains a single data point, followed by its correct label (the number of bikes sold in that hour) in the last column. In `bicycleregression.py`, we have loaded the data array for you and split it up into datapoints and associated labels.

Your goal is to fill in the `produce_regression_model` function in `bicycleregression.py` such that it returns a trained `RegressionLearner` that predicts as accurately as possible on novel datapoints.

4.2 Support Code

You will find `create_monomial_feature_func` and `all_monomials_with_maximum_degrees` helpful for producing monomial feature functions. You are encouraged to use them. View their documentation for details.

4.3 Grading

We have given you a fraction of the original dataset and withheld the rest. Your grade will be a function of the average squared error you achieve on the withheld portion of the dataset. Perfect scores will be awarded to regression models that achieve an average squared error of less than 11,000. Partial credit will be given for higher error rates, on a curve. In addition, submissions with error rates below 16,000 are guaranteed at least 25 of the 40 points.

4.4 Advice

Although in principle you may use any feature functions you like for this assignment, and it may be fun to experiment, you can earn full points using only monomial feature functions.

The main challenge in this assignment is to choose features that strike the right balance between overfitting and underfitting to the given subset of the data, such that your model generalizes well to the withheld subset of the data. The more features you include, the more likely you are to overfit, and vice versa.

A good way to prevent overfitting is to use regularization during training. You **do not** need to use regularization to succeed on this assignment.

5 Written Questions

Please turn in the written portion via Gradescope with each section clearly distinguished.

5.1 Question 1

Consider a dataset that has two binary features. The label belonging to each datapoint is a boolean function of the binary features. You should provide us with **valid sets of weights and bias terms** or **justify the failure** in using a perceptron for perfectly classifying the datapoints when:

1. The boolean function is AND.
2. The boolean function is OR.
3. The boolean function is XOR.

5.2 Question 2

Consider a binary classification problem in the 2D input space having X and Y coordinates. The plane consists of a circle of radius 1, in such a way that all points inside the circle are labelled 'A' and all points that lie outside are labelled 'B'. Answer the following questions based on the given information:

1. Are the classes linearly separable? Justify your answer.
2. Consider that along with the coordinates, their squares and product are also provided as features. Does addition of these features make the classes linearly separable?
3. Consider a scenario where you are allowed to choose a single feature in order to classify the inputs. Which feature would make the classes linearly separable?
   Hint: Consider scaling the problem from two dimentions to three dimensions for these questions)

6 Restrictions

You may not import any packages besides numpy in the version of your code that you hand in.

7 Install and Handin

- **Install**: `cs1410_install SL`
- **Handin**: `cs1410_handin SL`