CSC246/446 Machine Learning
Project 1: Hunt the Polynomial
Due: Jan 31 1159pm

Overview

This assignment is meant to give you practical experience with machine learning style programming.

At an engineering level, you will need to learn to work with datasets, commandline arguments, file paths, and numerical programming. At an academic level, you will deepen your understanding of vectors, matrices, loss functions, overfitting vs generalization, and more broadly speaking, the fundamental principles of supervised learning.

Your objective is to implement equation 3.28 from PRML – i.e., to find the best fitting regularized weights for a linear model. This part should be straightforward - you *must use python* and either numpy or pytorch. The instructor is more familiar with numpy, so if you choose pytorch, you are in uncharted territory. For the rest of this document anywhere you see numpy feel free to replace with pytorch. In either case, your program must be *vectorized* and reasonably efficient. *You may not do any slow single threaded pure python numerical computation loops*. E.g., you should use the vector manipulation routines from numpy. These are implemented in compiled languages which are much more efficient than python, and if you use python for ML, you need to know how to do it efficiently. You should implement equation 3.28 somewhere in your code as a method.

You will be recreating the results on polynomial fitting from the first chapter. I have created various synthetic datasets, each one being created by choosing a particular polynomial, sampling a subset of the x-axis, evaluating the polynomial at each point, and then adding a small amount of zero-mean Gaussian noise. Your objective is to identify for each dataset the best fitting polynomial degree that does not yield substantial overfitting.

You should write an additional method that sweeps through each polynomial order (up to a given maximum). For each step, you should then find the best weights and evaluate accuracy (via rmse) and estimate the degree of overfitting. You will then have to develop your own heuristics to identify the "best" order as the one with the highest accuracy achievable without significant overfitting. That is the primary goal of the assignment. Additionally, you must include a readme with a brief discussion of your approach and the results of your method. **This is an individual assignment and all work must be your own.**

Project Requirements

This section details the project requirements in terms of the required API, file name conventions, documentation requirements (a report style readme), and data file formats.

API

In order to facilitate automated testing, you must name your program polyhunt.py (or something similar, if you use Java or C) and your readme should be named readme.txt. Since everyone will be having the same filenames, it is critical that you include your names and UR email's in the contents of every file, and implement the "info" option which prints your name and contact information. You must submit a program which can be operated via the command line. No Jupyter notebooks. **You must include a readme which explains how to use your program and enable/disable the various options. You are not allowed to require the TAs to edit your code to change the options.**

Your program must support the following commandline arguments:

- `m` - integer - polynomial order (or maximum in autofit mode)
- `gamma` - float - regularization constant (use a default of 0)
- `trainPath` - string - a filepath to the training data
- `modelOutput` - string - a filepath where the best fit parameters will be saved, if this is not supplied, then you do not have to output any model parameters
- `autofit` - boolean - a flag which when supplied engages the order sweeping loop, when this flag is false (or not supplied) you should simply fit a polynomial of the given order and parameters. In either case, save the best fit model to the file specified by the modelOutput path, and print the RMSE/order information to the screen for the TA to read.
- `info` - boolean - if this flag is set, the program should print your name and contact information

You may define additional optional arguments of your own choosing. Some suggestions:

- `numFolds` - the number of folds to use for cross validation
- `devPath` - a path to a held-out data set

- `debug` - a flag to turn on printing of extra information (for debugging)
- ... plus any others you wish.

Data Format

The datasets will be in CSV format - two columns of floating point numbers, separated by a comma. The model parameters will be stored in numpy format, making use of commented header rows to store the polynomial order and regularization constant. – If you use pytorch, be sure to budget time to figure out an adapter. You may use pandas; however, you may not use libraries to implement the regression portion or cross validation – those must be your own code, from scratch.

Readme Requirements

Your readme should have the following six sections, in order:

1. **Method**. Briefly and specifically describe your autofit method. If you introduced any custom parameters that are necessary for your autofit, describe them here.
2. **Regularization**. Did you use it? If you didn't, probably your code will crash. What values did you try? What value seemed to work the best?
3. **Model Stability**. Run your program a few different times on the data (or on different subsets of the data) - how does the output change? Ideally you want a method that always picks the same correct order every time. In practice, there will be some variance. How stable are your predictions? If we run it a few times, will our results agree with yours?
4. **Results**. Describe the results of your autofit method. Does it match the expected outcome on the labeled datasets? What are the results for the second (unlabeled) group of datasets? Do you believe these results are correct?
5. **Notes**. This section is optional. Include any notes to TAs regarding quirks of your program, additional features, or other comments you believe may be helpful to know when testing and grading.

Academic Honesty

As stated on the syllabus, this course will follow the UofR Academic Honesty policy. Failure to follow the academic honesty policy carries stiff penalties. A typical result is a zero on the assignment with a further reduction of the overall grade. As is required by the policy, all instances of academic dishonesty will be reported. As stated earlier, this is an **individual assignment**.

Grading

Your project will be graded according to these criteria:

- **program correctness** – the weights output are correct for the basic version, all commandline arguments implemented and functional, doesn't crash, writes correct model file, tested by contents of model file at end and response to commandline arguments

  We must be able to check the weights that result from your program!

- **program efficiency** – good use of vector operations; no unnecessarily slow or inefficient use of for loops, list comprehensions, or data structures, tested by execution time and code inspection.
- **method accuracy** – your estimates are reasonably close to the ground truth (i.e., the actual polynomial order), tested on various datasets. There may be grade penalties for significantly overshooting or undershooting the correct degree.
- **method consistency** – your program's autofit results agree with what you claim in your report
- **readme** – everything I asked for is there, clear and concise, consistent and clean formatting

Please note: In order to streamline grading, please ensure that your full name and UR email are listed at the top of the redeem and your source files.