Winter 2026 DSC 240: Introduction to Machine Learning

Machine Problem 2

Due: **Monday, Feb 9th, 11:59 pm PST**

Notes:

1. **This assignment is to be done individually.** You may discuss the problems at a general level with others in the class (e.g., about the concepts underlying the question, or what lecture or reading material may be relevant), but the work you turn in must be solely your own.

2. Be aware of the late policy in the course syllabus – i.e., *no late days for all the assignments,* so **it is your responsibility to turn in your assignment to Gradescope by the due time.**

3. Any updates or corrections will be posted on Piazza, so check there occasionally.

4. You can refer to online resources and cite exact references. **No copying code**.

5. You should write the code from scratch, and you **are not allowed** to directly use any third-party tools such as `sklearn`. You can use any default Python libraries and the imported packages at the beginning of `mp2.py`

6. Be sure to re-read the **"Academic Integrity"** on the course syllabus. You must complete the section below. If you answered Yes to either of the following two questions, give corresponding full details.

Did you receive any help whatsoever from anyone in solving this assignment?
Did you give any help whatsoever to anyone in solving this assignment?

In MP1, you learned how to implement a simple hand-crafted classifier. We will make things more interesting this time. In MP2, you will **start from scratch** and implement your favorite learning algorithm from the class that outputs one linear classifier. You can choose to implement any one (or more) of the learning algorithms, e.g., the Perceptron algorithm, Gradient Descent, or Stochastic Gradient Decent. For GD / SGD, you may use any surrogate loss. The **only constraint** is that your algorithm needs to fit into the time/memory requirements of the autograder hosted on Gradescope.

Remember to keep the function name and argument of `run_train_test(training_data, testing_data)` unchanged.

At the end of the day, you can submit the best-performing training algorithm.

You can use `Pandas` library to deal with the data and use `NumPy` and `SciPy` for fast linear algebraic operations.

Problem Overview

Change the Python 3 program `mp2.py` that creates a linear classifier for the binary classification problem. The training and testing data sets are available in the starter package. For each data point, there are 5 real-value features (`x1`, `x2`, `x3`, `x4`, `x5`). The target is 0/1, indicating which class it belongs to.

- `data/*`: include `train.csv` for training and `dev.csv` for development. We will have an extra private test set for the final grading.
- `mp2.py`: the Python file you need to work on and submit.

Code Instructions

In this project, you need to implement one learning algorithm (for example: perceptron) to obtain a linear classifier for binary classification. You may try out different algorithms we covered in class and tweak them so they work. At the end of the day, you should submit only one learning algorithm to the *Gradescope*. Your algorithm should be implemented from scratch. Please **do not use** any external implementation or off-the-shelf packages. Specifically, for this assignment, a good template would be to write an algorithm that iteratively updates the weight by following an update rule (SGD, GD, or Perceptron) for learning until it converges or reaches the maximum number of iterations you set (or when certain other conditions for convergence are met).

`run_train_test(training_data, testing_data)` function is called to update the parameters in the linear classifier and get the prediction on testing_data via the trained classifier.

Evaluation Instructions

You can test your program with the `train.csv` and `dev.csv` provided in this starter package. You can directly run `mp2.py` for checking:

```
python mp2.py
```

This will output the predictions you get on the dev dataset. We will use the **F1 score** to evaluate the results. The final score will be graded on another private test set (although the same training dataset will be used for training the classifier).

Submission Instructions

You should upload `mp2.py` to Gradescope for grading.

Grading Instructions

The score on Gradescope is computed based on the F1 score using the following rules:

- **Manual Grader (50%)**: If your code can run successfully, you will get full points. Otherwise, your score will be based on completeness.
- **AutoGrader (50%)**: Your score depends on the F1 score on the private test data:
  - 15% weightage: If your F1 score is equal to 0.80 or more. You get 15% for just matching 0.80.
  - 35% weightage: Additional prorated marks when your F1 score is above 0.80:

$$Score = 35\% \times \frac{(Your\ F1\ Score - 0.80)}{(0.95 - 0.80)}$$

  - If your F1 score is below 0.80:

$$Score = 0$$

  - **Bonus(10%)**: If your score beats the TA baseline, you can get bonus marks! For reference, TA baseline F1 score is 0.95

Late submissions are **not accepted**.

Additional Important Notes

As mentioned in the evaluation instructions, the `run_train_test(training_data, testing_data)` function in the `mp2.py` file should return your classifier's predictions. We will use the **F1 Score** calculated on a hidden test dataset in Autograder for scoring.

Note: While we provided an example metric for accuracy in `mp2.py` template code, we have **not provided** any code for the F1 score and **strongly suggest** that you implement it yourself for a better understanding of concepts. There is **no need** to include any code for F1 score evaluation in your final submission. We ask you to implement it locally so that you can benchmark your solutions and correctly estimate your classifier's performance before submitting it to Gradescope.