CS 335 Autumn 2023 | Midsem Exam

Instructor: Preethi Jyothi Date/Time: Sep 22, 2023, 10:30 am to 1:00 pm

Total points: **20**

Name: _____

Roll number: _____

Instructions

- This is a closed book/closed internet exam which should be completed individually. No form of collaboration or discussion is allowed.
- Documentation to necessary libraries such as `numpy`, `matplotlib` will be available to you.
- Use of laptops or cell phones are not allowed during the exam.
- Write your name and roll number on the top of this page.
- This exam consists of 2 problems and the maximum possible score is 20.
- For your exam submission, create [rollnumber].tgz with the following internal directory structure:

```
[rollnumber]/
  +- README.txt
  +- CE.py
  +- votedperceptron.py
  +- plotdata.pdf
  +- extra_credit.py [OPTIONAL]
```

  Submit [rollnumber].tgz on Moodle.

- Good luck!

Problem 1: Loss Computation (7 points)

Cross-Entropy Loss. Write a function that computes the cross-entropy loss of a model with respect to a given data set. More precisely, for a feature vector $\mathbf{x}$ and a label $y \in \{1, \ldots, K\}$, let $S(y, \mathbf{x})$ denote the logit value (a real valued score) output by a model. Probability values $P(y, \mathbf{x})$ are defined using the softmax function applied to these logit values. Given a K-dimensional vector of real values $\mathbf{s}$ and an index y, $\text{softmax}(\mathbf{s}, y) = \frac{\exp(\mathbf{s}_y)}{\sum_{z \in \{1,\ldots,K\}} \exp(\mathbf{s}_z)}$. Then, w.r.t. a batch of n training instances $(\mathbf{x}_i, y_i)$, the cross-entropy loss (L_{CE}) of this model is defined as:

$$L_{\text{CE}} = -\frac{1}{n} \sum_{i=1}^{n} \log P(y_i, \mathbf{x}_i).$$

(A) Fill in the function definitions in `CE.py`. Your code should work for any values of `num_classes` and `batch_size`. **[5 pts]**

(B) Note that for logits $\mathbf{s}_y$ with very large magnitudes, computing $\exp(\mathbf{s}_y)$ for the softmax probabilities could lead to numerical overflow or underflow. Protect against this in your code. Explain how you did this in `README.txt`. **[2 pts]**

Problem 2: Voted Perceptron (13 points)

For this problem, you will implement a voted perceptron and compare its performance with that of a standard/vanilla perceptron classifier.

The algorithm for a voted perceptron is as follows:

Inputs: Training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$, number of epochs T, learning rate η, initial weight vector $\mathbf{w}_0$
Output: List of weighted perceptrons, $\{(\mathbf{w}_1, c_1), \ldots, (\mathbf{w}_k, c_k)\}$
Set $k \coloneqq 1$, $\mathbf{w}_1 \coloneqq \mathbf{w}_0$ and $c_1 \coloneqq 0$ ▷ Note: $\mathbf{w}_0$ also contains an extra intercept/bias dimension.
Repeat T times
begin
    for i = 1 ... n **do**
        Compute $\hat{y}_i \coloneqq \text{sign}(\mathbf{w}_k^T \mathbf{x}_i)$
        if $\hat{y}_i = y_i$ **then**
            $c_k \coloneqq c_k + 1$
        else
            $\mathbf{w}_{k+1} \coloneqq \mathbf{w}_k + \eta y_i \mathbf{x}_i$
            $c_{k+1} \coloneqq 1$
            $k \coloneqq k + 1$
        end if
    end for
end

Prediction. At test time, for an instance $\mathbf{x}$, the voted perceptron uses a weighted majority vote across $\{(\mathbf{w}_1, c_1), \ldots, (\mathbf{w}_k, c_k)\}$. That is, the predicted label would be $\hat{y} = \text{sign}\left(\sum_{i=1}^{k} c_i \, \text{sign}(\mathbf{w}_i^T \mathbf{x})\right)$.

The template code in `votedperceptron.py` is fairly self-explanatory and contains many comments elaborating on the functionality of various lines of code.

(A) Within the class `VotedPerceptron`, complete the function definitions of `__init__`, `fit` and `predict`. **[2+4+3=9 pts]**

- Let the initial weight vector $\mathbf{w}_0$ be an all-zeros vector.
- DO NOT change the values of `eta`, `max_epochs` or any other hyperparameters from their default values.
- Note that in `predict`, setting the flag `vanilla` to `True` or `False` determines whether or not the standard/vanilla perceptron or the voted perceptron is invoked, respectively. The standard/vanilla perceptron will correspond to the final weight vector $\mathbf{w}_k$ in the above-mentioned algorithm.

(B) Save a scatter plot of the dataset, along with the decision boundary learned by the standard/vanilla perceptron, in `plotdata.pdf`. Complete the definitions of `get_decision_boundary` and `plot`. **[2 pts]**

(C) Write down the test accuracy from using both the voted perceptron and the standard/vanilla perceptron in `README.txt` without changing any of the default hyperparameter values. In `votedperceptron.py`, we also predict labels for the instances in `blind_data.csv` and write them into `output.csv`. We will match these predictions, along with the test accuracies, with the outputs from our autograder. **[2 pts]**

(D) **EXTRA CREDIT:** Try to improve the test accuracy using any modifications you like. Save this code in a separate file `extra_credit.py` that we will run as `python extra_credit.py` on the command line. The output predictions for the data in `blind_data.csv` using your modifications should be written to `output.csv`. Write down your modifications in `README.txt`. Extra credit points will only be given out if you exceed the test accuracies with the default settings. Additionally, the number of extra credit points granted will depend on the kinds of modifications you've made. **[3 pts]**