Homework 9

Submit Assignment

Due May 15 by 11:59pm **Points** 8 **Submitting** a file upload **File Types** py
Available May 2 at 2:45pm - May 17 at 11:59pm 15 days

Introduction

In this project, you will implement a multiclass perceptron classifier as well as a classifier for a Pacman apprentice. You will test the first classifier on a set of scanned handwritten digit images, and the last on sets of recorded pacman games from various agents. Even with simple features, your classifiers will be able to do quite well on these tasks when given enough training data.

As in the previous projects, an autograder is provided for you to grade your answers on your machine.

The code for this project consists of several Python files. You can download all the code and data as a zip archive file: homework9.zip.

File you'll edit and submit: perceptron.py and perceptron_pacman.py

You will fill in portions of perceptron.py, and perceptron_pacman.py and submit them. Please do not change the other files in this distribution or submit any of the original files other than these two files.

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation -- not the autograder's judgements -- will be the final judge of your score.

Academic Dishonesty: Your code will be checked for plagiarism. If you copy someone else's code and submit it with minor changes, it will be easy to detect and it will lead to an automatic 0 on the assignment. I trust you all to submit your own work only; please don't let me down.

Getting Help: You are not alone! If you find yourself stuck on something, please ask for help. Office hours and the discussion forum are there for your support; please use them. I want these projects to be rewarding and instructional, not frustrating and demoralizing. But, I don't know when or how to help unless you ask.

Discussion: Please be careful not to post spoilers.

Question 1 (4 points): Perceptron

A skeleton implementation of a multiclass perceptron classifier is provided for you in perceptron.py.

A multiclass perceptron classifier keeps a weight vector w_y for each label y.

Given a feature vector f (in our case, a map from pixel locations to indicators of whether they are on), the perceptron computes the class y whose weight vector is most similar to the feature vector f.

Formally, given a feature vector f, we score each class with:

score(y) = w_y . f

Then we choose the class with the highest score as the predicted label for that data instance.

In the code, we will represent w_y as a Counter (the Counter class is defined in util.py).

The *classify* method has been written for you. Take a look at it for inspiration. Your task is to fill in the *train* method.

Using the addition, subtraction, and multiplication functionality of the Counter class in util.py, the perceptron updates should be relatively easy to code. Certain implementation issues have been taken care of for you in perceptron.py, such as handling iterations over the training data and ordering the update trials. Furthermore, the code sets up the weights data structure for you. Each legal label needs its own Counter full of weights.

Run your code with:

```
python dataClassifier.py
```

The command above should yield validation accuracies in the range between 40% to 70% and test accuracy between 40% and 70% (with the default 3 iterations and default 100 training examples).
One of the problems with the perceptron is that its performance is sensitive to several practical details, such as how many iterations you train it for. The current code uses a default value of 3 training iterations. You can change the number of iterations for the perceptron with the -i iterations option.

```
python dataClassifier.py -i 6
```

Try different numbers of iterations and see how it influences the performance. In practice, you would use the performance on the validation set to figure out when to stop training, but you don't need to implement this stopping criterion for this assignment.

You can also try using more training examples and see the effect on accuracy:

```
python dataClassifier.py -t 500
```

```
python dataClassifier.py -t 1000
```

Question 2 (4 points): Pacman Apprentice

You will now use a modified version of perceptron in order to learn from pacman agents. In this question, you will fill in the train method in perceptron_pacman.py. This code should be similar to the method you've written in perceptron.py.

For this application of classifiers, the data will be states, and the labels for a state will be all legal actions possible from that state. Unlike perceptron for digits, **all of the labels share a single weight vector w**, and **the features extracted are a function of both the state and possible label.**

For each action, we calculate the score as follows:

$$\text{score} = w * f(a)$$

Then the classifier assigns whichever label receives the highest score:

a' = argmax(score)

Here again, the *classify* method has been written for you. Take a look at it for inspiration.

Training updates occur in much the same way that they do for the standard classifiers. Instead of modifying two separate weight vectors on each update, the weights for the actual and predicted labels, both updates occur on the shared weights as follows:

w = w + f(a) # Correct action

w = w - f(a') # Guessed action

Run your code with:

```
python dataClassifier.py  -d pacman
```

This command should yield validation and test accuracy of over 70%.