CS440/ECE448 Spring 2020

Assignment 5: Classify

Due date: Wednesday April 8th, 11:59pm

Created 2018: Justin Lizama and Medhini Narasimhan
Updated 2020: Daniel Gonzales and Kuangxiao Gu

In this assignment, we are going to see if we can teach a computer to distinguish living things from non-living things. More precisely, you will implement the perceptron and logistic regression algorithms to detect whether or not an image contains an animal or not.

Contents

General guidelines and submission

Basic instructions are the same as in MP 1. To summarize:

- For general instructions, see the main MP page and the course policies.

You should submit on Gradescope:

- A copy of **classify.py** containing all your new code

Problem Statement

You are given a dataset consisting of images, that either contain pictures of animals or not. Your task is to write two classification algorithms to classify which images have animals in them. Using the training set, you will learn a perceptron and logistic regression classifier that will predict the right class label given an unseen image. Use the development set to test the accuracy of your learned models. We will have a separate (unseen) test set that we will use to run your code after you turn it in. You may use NumPy in this MP to program your solution. Aside from that library, no other outside non-standard libraries can be used.

Dataset

This dataset consists of 10000 32x32 colored images total. We have split this data set for you into 2500 development examples and 7500 training examples. The images have their RGB values scaled to range 0-1. This is a subset of the CIFAR-10 dataset, provided by Alex Krizhevsky.

The data set can be downloaded here: data (gzip) or data (zip). When you uncompress this, you'll find a binary object that our reader code will unpack for you.

Perceptron Model

The perceptron model is a linear function that tries to separate data into two or more classes. It does this by learning a set of weight coefficients w_i and then adding a bias b. Suppose you have features $x_1, \ldots, x_n$ then this can be expressed in the following fashion:

$$f_{w,b}(x) = \sum_{i=1}^{n} w_i x_i + b$$

You will use the perceptron learning algorithm to find good weight parameters w_i and b such that $\text{sign}(f_{w,b}(x)) > 0$ when there is an animal in the image and $\text{sign}(f_{w,b}(x)) \leq 0$ when there is a no animal in the image. Note that in our case, we have 3072 features because each image is 32x32 and they each have RGB color channels yielding 32*32*3 = 3072.

Training and Development

Please see the textbook and lecture notes for the perceptron algorithm. You will be using a single classical perceptron whose output is either +1 or -1 (i.e. sign/step activation function).

- **Training:** To train the perceptron you are going to need to implement the perceptron learning algorithm on the training set. Each pixel of the image is a feature in this case. **Be sure to initialize weights and biases to zero. Note:** To get full points on the autograder, use a constant learning rate (no decay) and do not shuffle the training data.
- **Development:** After you have trained your perceptron classifier, you will have your model decide whether or not images in the development set contain animals in them or not. In order to do this take the sign of the function $f_{w,b}(x)$. If it is negative then classify as 0. If it is positive then classify as 1.

Use only the training set to learn the weights.

Logistic Regression Model

Logistic regression is another linear model that tries to separate data into two or more classes. It does this by learning a set of weight coefficients w_i and then adding a bias b. Suppose you have features $x_1, \ldots, x_n$ then this can be expressed in the following fashion:

$$f_{w,b}(x) = \sigma\Big(\sum_{i=1}^{n} w_i x_i + b\Big), \quad \sigma(x) = \frac{1}{1+e^{-x}}$$

The sigmoid function σ is applied to the linear function to give an output between 0 and 1. You will use the gradient descent learning algorithm to find good weight parameters w_i and b such that $round(f_{w,b}(x)) = 1$ when there is an animal in the image and $round(f_{w,b}(x)) = 0$ when there is a no animal in the image.

Training and Development

Please see the lecture notes for training a logistic regression model.

- **Training:** To train the model you are going to need to implement gradient descent on the training set. The loss used in this MP is the binary cross entropy loss:

  $$L = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} \ln(f_{w,b}(x^{(i)})) + (1-y^{(i)}) \ln(1-f_{w,b}(x^{(i)}))$$

  Where $x^{(i)}$ and $y^{(i)}$ are i^{th} data-label pair in the training set. **Be sure to initialize weights and biases to zero. Note:** To get full points on the autograder, use a constant learning rate (no decay) and do not shuffle the training data.
- **Development:** After you have trained your logistic regression classifier, you will have your model decide whether or not images in the development set contain animals in them or not. In order to do this round the function $f_{w,b}(x)$ to the nearest integer. If it is < 0.5 then classify as 0. If it is ≥ 0.5 then classify as 1.

Use only the training set to learn the weights.

Extra Credit

Perceptron and Logistic Regression are simple linear models, and although this is sufficient in a lot of cases, they have their limits. Implement K-Nearest Neighbors using Euclidean distance. To break ties, use the negative label (no animal class). Try and see what's the highest accuracy you can get, and find some justification (just for fun no need to submit) for why your choice of model is superior to linear models for this particular task. You must implement this algorithm on your own with only standard libraries and NumPy. The choice of K should be chosen based on experimentation. **Note:** To prevent memory errors due to the autograder's limitations, iterate through each sample in your algorithm instead of using a vectorized approach.

Provided Code Skeleton

We have provided (tar zip) all the code to get you started on your MP, which means you will only have to implement the logic behind perceptron.

- **reader.py** - This file is responsible for reading in the data set. It makes a giant NumPy array of feature vectors corresponding with each image.
- **mp5.py** - This is the main file that starts the program, and computes the accuracy, precision, recall, and F1-score using your implementation of the classifers.
- **classify.py** This is the file where you will be doing all of your work.

Inside the code ...

- The function classifyPerceptron() and classifyLR() take as input the training data, training labels, development set, learning rate, and maximum number of iterations.
- The training data provided is the output from **reader.py**.
- The training labels is the list of labels corresponding to each image in the training data.
- The development set is the NumPy array of images that you are going to test your implementation on.
- The learning rate is the hyperparameter you specified with --lrate (it is 1e-2 by default). Please do not reset this value inside your code.
- The maximium number of iterations is the number you specified with --max_iter (it is 10 by default). Please do not reset this value inside your code.
- You will have each classify() output the predicted labels for the development set from your models.
- The function classifyEC() is a function where you can implement the extra credit, if you decide to attempt it. If you use the --extra flag then classifyEC() will be ran instead of classify().

NOTE: In classify() only implement the models on the raw data. Do NOT do any extra credit or anything extra in classify(). Do not modify the provided code. You will only have to modify classify.py.

To understand more about how to run the MP, run **python3 mp5.py -h** in your terminal.