Homework 2

Due: March 2, 2018 at 7:00PM

Written Questions

Problem 1: Estimator

(5 points)

Let $x_1, x_2, ..., x_m$ be an i.i.d. (independent and identically distributed) sample drawn from distribution $B(p)$ where $B(p)$ denotes a Bernoulli distribution. Specifically,

$$\mathbb{P}(B = 1) = p, \quad \mathbb{P}(B = 0) = 1 - p.$$

Suppose p is an unknown parameter and we estimate it via:

$$\widehat{p} = \frac{1}{m} \sum_{i=1}^{m} (x_i).$$

Show that $\widehat{p}$ is an unbiased estimator for p. Recall that an estimator is unbiased if its expected value over all possible samples is equal to the parameter it is estimating. *Note:* In lecture, we showed this estimator is unbiased for a sample size of 3. For this question, we are asking you to generalize the argument to a sample size of m.

Problem 2: Agnostic PAC Learning

(25 points)

Previously, we looked at PAC learning under the assumption that the true hypothesis was a function within our set of hypotheses H—the *realizable case*. However, this assumption does not always hold. In some cases, the true hypothesis is a function $f \notin H$—the *unrealizable case*. Learning in the unrealizable case is also called **agnostic learning**. We will examine how PAC learning differs in this scenario.

Hoeffding's inequality can give us a bound on sample size for the unrealizable case in which we have a finite set of hypotheses H. The true function for labeling data is f. Suppose we are labelling data x generated from distribution D. Define the *expected error* of a hypothesis, $\mathrm{err}_D(h)$, as the expected proportion of data incorrectly labelled by the hypothesis h, written as:

$$\mathrm{err}_D(h) = \mathbb{E}_{x \sim D}[f(x) \neq h(x)].$$

We have a sample S. Define the *sampling error* of a hypothesis, $\mathrm{err}_S(h)$, as the proportion of data from the sample S incorrectly labelled by hypothesis h. It may be written as:

$$\mathrm{err}_S(h) = \frac{1}{|S|} \sum_{(x,y) \in S} [y \neq h(x)].$$

Define $\mathrm{ERM}(H) = \mathrm{argmin}_{h \in H}\, \mathrm{err}_S(h)$ and $\mathrm{RM}(H) = \mathrm{argmin}_{h \in H}\, \mathrm{err}_D(h)$ as the empirical risk minimizing hypothesis and the risk minimizing hypothesis, respectively. In the PAC setting, we content ourselves with an algorithm that, with probability at least $1 - \delta$, returns a hypothesis $\widehat{h}$ whose error over the distribution D is within ϵ of that of the risk minimizing solution, or:

$$|\mathrm{err}_D(\widehat{h}) - \mathrm{err}_D(\mathrm{RM}(H))| \leq \epsilon.$$

Note: We are considering PAC learning on a binary dataset.

a. First, consider the realizable case, $f \in H$. What is the PAC learning bound for $\widehat{h} = \text{ERM}(H)$? Express the sample size bound in terms of ϵ, δ, and $|H|$.

b. Suppose we're stuck with agnostic learning such that $f \notin H$. Our best hypothesis is $\widehat{h} = \text{ERM}(H)$. Use Hoeffding's inequality to show that, given ϵ_1 and δ_1, there's an m such that sampling $S = \{x_1, ..., x_m\} \sim D$ gives us $|\text{err}_S(h) - \text{err}_D(h)| \leq \epsilon_1$, with probability at least $1 - \delta_1$, for some h.

c. What happens if we use a sample of size m to evaluate *all* the hypotheses in H? In particular, we want it to be simultaneously true that, for all $h \in H$, $|\text{err}_S(h) - \text{err}_D(h)| \leq \epsilon_1$ with probability $1 - \delta$. Write an upper bound for δ in terms of δ_1 using the union bound.

d. Now we know that our error estimates for all $h \in H$ are within ϵ_1 of their true errors (with high probability). If we pick $\widehat{h} = \text{ERM}(H)$, how far might $\widehat{h}$ be from $\text{RM}(H)$? Call that ϵ and write ϵ in terms of ϵ_1. That is, find ϵ in terms of ϵ_1 such that $|\text{err}_D(\widehat{h}) - \text{err}_D(\text{RM}(H))| \leq \epsilon$.

e. Putting it all together, define m in terms of ϵ and δ so that, with probability at least $1 - \delta$ (using the previously computed bound), $|\text{err}_D(\text{ERM}(H)) - \text{err}_D(\text{RM}(H))| \leq \epsilon$.

Problem 3: Naive Bayes Maximum Likelihood

(12 points)

Consider binary dataset S with observations in the form $\{(x_j^1, ..., x_j^n), y_j)\}$. Define $c(y)$ as a function that counts the number of observations such that the label is y.

$$c(y) = \sum_{(x_j, y_j) \in S} [y_j = y]$$

Define $c(i, y)$ as a function that counts the number of observations such that the label is y and $x^i = 1$.

$$c(i, y) = \sum_{(x_j, y_j) \in S} [y_j = y, x_j^i = 1]$$

Define b as $\mathbb{P}(Y = 1)$, and b^{iy} as $\mathbb{P}(X^i = 1 | Y = y)$. Prove that the following estimators are MLE for these parameters:

$$\widehat{b}_{MLE} = \frac{c(1)}{|S|} \quad \text{and} \quad \widehat{b^{iy}}_{MLE} = \frac{c(i, y)}{c(y)}$$

Problem 4: Gradient Descent

(18 points)

We have a convex function f over the closed interval $[-b, b]$ (for some positive number b). Let f' be the derivative of f. Let α be some positive number, which will represent a learning rate parameter.

Consider using gradient descent to find the minimum of f: We start at $x_0 = 0$. Then, at each step, we set $x_{t+1} = x_t - \alpha f'(x_t)$. If x_{t+1} falls below $-b$, we set it to $-b$, and if it goes above b, we set it to b.

We say that an optimization algorithm (such as gradient descent) ϵ-*converges* if, at some point, x_t stays within ϵ of the true minimum. Formally, we have ϵ-convergence at time t if

$$|x_{t'} - x_{\min}| \leq \epsilon, \qquad \text{where } x_{\min} = \operatorname*{argmin}_{x \in [-b, b]} f(x)$$

for all $t' \geq t$.

a. For $\alpha = 0.1$, $b = 1$, and $\epsilon = 0.001$, find a convex function f so that running gradient descent does not ϵ-converge. Specifically, make it so that $x_0 = 0$, $x_1 = b$, $x_2 = -b$, $x_3 = b$, $x_4 = -b$, etc.

b. For $\alpha = 0.1$, $b = 1$, and $\epsilon = 0.001$, find a convex function f so that gradient descent does ϵ-converge, but only after at least 10,000 steps.

c. Construct a different optimization algorithm that has the property that it will always ϵ-converge (for any convex f) within $\log_2(2b/\epsilon)$ steps.

d. Unfortunately, even if x_t is within ϵ of $x_{\min}$, $f(x_t)$ can be arbitrarily greater than $f(x_{\min})$. However, consider the case where the derivative of f is always between $-r$ and r. ($\forall x \in [-b, b]$, $f'(x) \in [-r, r]$.) In this case, we can make a guarantee about the difference between $f(x_t)$ and $f(x_{\min})$.

   Given that $|x_t - x_{\min}| \leq \epsilon$ and that $-r \leq f'(x) \leq r$, find a bound on $|f(x_t) - f(x_{\min})|$ in terms of ϵ and r.

Programming Assignment

Introduction

In this assignment, you'll implement both Naive Bayes and Logistic Regression and use these algorithms to classify handwritten digits.

Stencil Code & Data

You can find the stencil code for this assignment on the course website. We have provided the following two files:

- `main.py` is the entry point of your program which will read in the data, run the classifiers and print the results.
- `models.py` contains the `NaiveBayes` model and the `LogisticRegression` model which you will be implementing.

You should *not* modify any code in the `main.py`. All the functions you need to fill in reside in `models.py`, marked by `TODO`s. You can see a full description of them in the section below.

To feed the data into the program successfully, please do *not* rename the data files and also make sure all the data files are in a directory named `data` located in the same directory as `main.py`. To run the program, run `python main.py` in a terminal.

MNIST Dataset

You will be using the well-known MNIST Dataset, which includes 60,000 examples for training and 10,000 for testing. All images are size normalized to fit in a 20×20 pixel box and this box is centered in a 28×28 image based on the center of mass of its pixels. The full description of this dataset is available at `http://yann.lecun.com/exdb/mnist/`. There are four files:

- `train-images-idx3-ubyte.gz`: images of 60,000 training examples,
- `train-labels-idx1-ubyte.gz`: labels of 60,000 training examples,
- `t10k-images-idx3-ubyte.gz`: images of 10,000 testing examples,
- `t10k-labels-idx1-ubyte.gz`: labels of 10,000 testing examples.

On a department machine, you can copy the files to your working directory by running the command

```
cp /course/cs1420/data/hw2/* <DEST DIRECTORY>
```

where `<DEST DIRECTORY>` is the directory where you would like to copy the four data files. If you are working locally, you will need to use the `scp` command to copy the files to your computer over `ssh`:

```
scp <login>@ssh.cs.brown.edu:/course/cs1420/data/hw2/* <DEST DIRECTORY>
```

Data Format

The original feature values in this dataset are integers ranging from 0 to 255 (encoding grey scale values), but we have thresholded those features to make them binary (1 for values higher than 99, 0 otherwise). The labels are the same as the original dataset, which are integers in the range $[0, 9]$.

As always, we have written all the preprocessing code for you. The final dataset is represented by a `namedtuple` with two fields:

- `data.inputs` is a $m \times 784$ `NumPy` array that contains the binary features of the m examples.
- `data.labels` is a m dimensional `NumPy` array that contains the labels of the m examples.

You can find more infomation on `namedtuple` **here**

The Assignment

In `models.py`, there are six functions you will implement, three for each model. They are:

- `NaiveBayes`:
  - **train()** uses maximum likelihood estimation to learn the parameters. Because all the features are binary values, you should use the Bernoulli distribution (as described in lecture) for the features.
  - **predict()** predicts the labels using the inputs of test data.
  - **accuracy()** computes the percentage of the correctly predicted labels over a dataset.
- `LogisticRegression`:
  - **train()** uses stochastic gradient descent to optimize the parameters of the logistic regression model.
  - **predict()** predicts the labels using the inputs of test data.
  - **accuracy()** computes the percentage of the correctly predicted labels over a dataset.

Note: A softmax function has already been implemented. What softmax does is normalize input values from the range $[-\infty, \infty]$ into the range $[0, 1]$ so that they add up to 1. You can think of the output of the softmax function as a probability distribution over n classes. The shape of l, p and $\nabla\mathcal{L}(w; l)$ are all $[1, n]$ where n is the number of classes and l denotes the logits, which are used as inputs of softmax function.

Note: You are not allowed to use any off-the-shelf packages that have already implemented these models, such as scikit-learn. We're asking you to implement them yourself.

Gradient Descent

You will be using Stochastic Gradient Descent (SGD) to train your `LogisticRegression` model. Below, we have provided pseudocode for SGD on a sample S:

```
initialize parameters $w$ and learning rate $\alpha$
repeat until $w$ converges:
    shuffle training examples
    for each training example $(x, y) \in S$:
        $l = \langle x, w \rangle$
        $p = \text{softmax}(l)$
        for $i \in$ the set of classes
            if $i = y$: $\nabla \mathcal{L}(w; l_i) = p_i - 1$
            else: $\nabla \mathcal{L}(w; l_i) = p_i$
        $\nabla \mathcal{L}(w; x) = \langle x, \nabla \mathcal{L}(w; l) \rangle$
        $w = w - \alpha \nabla \mathcal{L}(w; x)$
```

Project Report

As introduced in HW1, each programming assignment in this course will be accompanied by a short report in which you will answer a few guiding questions about the results of your algorithm. To answer these questions, you may need to write new code that generates some output (potentially a value or graph) that you will want to include in your writeup. For example, you may be asked to contrast the results of running the same algorithm on different datasets or to explore the effect of changing a certain hyperparameter in your algorithm. By the end of this course you will not only be able to implement common machine-learning algorithms but also develop intuition as to how the results of a given algorithm should be interpreted.

The next section outlines some guiding questions that you should answer in your report. Please leave any code that you use in your final handin but make sure that it is **not** run by default when your program is run. We ask that your final report be a PDF file named `report.pdf`, which must be handed in along with your code. You may use any program to create the PDF file, but we highly recommend using LaTeX. We have provided an example report available on our course website to get you started.

Guiding Questions

- Report the training and testing error of both classifiers. Discuss the results in a short paragraph.
- When training the `LogisticRegression` model with SGD, graph the loss at each iteration of the algorithm. Produce 2 graphs, each with a different learning rate. In a sentence or two discuss the graphs.

Grading Breakdown

We expect the accuracies that `NaiveBayes` and `LogisticRegression` reach should be above 84% and 86%, respectively. For logistic regression, we will evaluate your implementation on the highest accuracy among five trials since we are using stochastic gradient descent. As always, you will primarily be graded on the correctness of your code and not based on whether it does or does not achieve the accuracy targets.

The grading breakdown for the assignment is as follows:

Written Questions	60%
Naive Bayes	16%
Logistic Regression	16%
Report	8%
Total	100%

Handing in

Written Questions

Answers to the written questions should be printed, stapled and labeled with the date, homework number and your unique course ID. You should place your handin in the CS1420 handin box located on the second floor of the CIT.

Programming Assignment

To hand in the programming component of this assignment, first ensure that your code runs on *Python 3* using our course `virtualenv`. You can activate the `virtualenv` on a department machine by running the following command in a Terminal:

```
source /course/cs1420/cs142_env/bin/activate
```

Once the `virtualenv` is activated, run your program and ensure that there are no errors. We will be using this `virtualenv` to grade all programming assignments in this course so we recommend testing your code on a department machine each time before you hand in. Note that handing in code that does not run may result in a significant loss of credit.

To hand in the coding portion of the assignment, run `cs142_handin hw2` from the directory containing all of your source code and your report in a file named `report.pdf`.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin. Instead, you should use the course ID that you generated when filling out the collaboration policy form. If you do not have a course ID, you should email the HTAs as soon as possible.

Obligatory Note on Academic Integrity

Plagiarism—don't do it.

As outlined in the Brown Academic Code, attempting to pass off another's work as your own can result in failing the assignment, failing this course, or even dismissal or expulsion from Brown. More than that, you will be missing out on the goal of your education, which is the cultivation of your own mind, thoughts, and abilities. Please review this course's collaboration policy and, if you have any questions, please contact a member of the course staff.

Feedback

Many aspects of this course have been radically redesigned compared to previous iterations of the course. To continually improve this class and future iterations of the class, we are offering an online **anonymous feedback form**. If you have comments, we would love to hear your constructive feedback on this assignment and on any other aspects of the course.