This lab will be discussion-based, but please write something down. We intend for you to begin thinking about these concepts, but **please do not stress if you don't understand everything perfectly by the end.** Use the checkoffs to clarify your understanding and clear up confusion, and come to office hours if you want to discuss lab topics further. The staff are here to help you learn!

Evaluating learning methods

In HW 2, we will implement a very simple learning algorithm that, when given a set of possible hyperplanes and a data set $\mathcal{D}_{train}$ (consisting of a data array and label vector), returns the hyperplane that minimizes the number of errors on the training set. In the lecture, we have also considered the Perceptron algorithm (notes) and will implement it in homework this week. We will go on to study a number of other algorithms, all of which take in data as input and return a classification hypothesis as output.

In this lab, we will explore how to evaluate learning methods. You will have a checkoff conversation with a staff member at the end of the lab. After that, you're welcome to leave or stay to work on homework.

Note the following notation and definitions, used throughout this lab:

- A generator $\mathcal{G}$ is a function that takes as input n, the number of samples desired, and returns an $(\mathcal{X}, y)$ pair where $\mathcal{X}$ is a d by n array of randomly sampled data points and y is a 1 by n array of their corresponding labels $\{+1, -1\}$.
- A training dataset $\mathcal{D}_{train}$ is a set of labelled samples generated by $\mathcal{G}$, $\mathcal{X}, y$, where x^i represents the features of an object to be classified (vector of real and/or discrete values), and y^i represents the label of x^i.
- A binary classifier h is a function that takes an example $x \in R^d$ as input and returns $+1$ or -1 as output.

1) Evaluating a classifier

Imagine that you have a generator $\mathcal{G}$ that pulls from a finite dataset of millions of points.

Let's assume that $\mathcal{D}_{train}$ is one such output of the generator $\mathcal{G}$.

Consider the situation in which you have run a machine learning algorithm on some training dataset $\mathcal{D}_{train}$, and it has returned to you a specific h. Your job is to design (but not implement yet!) a procedure for evaluating h's effectiveness as a classifier. (Want more on classifiers? Check the notes)

Assume we have a `score` function that takes a classifier h, dataset D - a tuple of data and labels: (X, y) - and returns the percentage of correctly classified examples as a decimal between 0 and 1. We'll package it as follows:

```
def eval_classifier(h, D):
    test_X, test_y = D
    return score(h, test_X, test_y)
```

A) Percy Eptron suggests reusing the training data to assess h:

```
eval_classifier(h, D_train)
```

Explain why Percy's strategy might not be so good.

B) Now write down a better approach for evaluating h, which may use h, $\mathcal{G}$, and $\mathcal{D}_{train}$, and computes a score for h. The syntax is not important, but do write something down. What does this score measure and what is the range of possible outputs?

C) Explain why your method might be more desirable than Percy's. What problem does it fix?

D) How would your method from **B** score the classifier h, if $\mathcal{D}_{test}$ came from a different distribution than $\mathcal{G}$, but $\mathcal{D}_{train}$ was unchanged?

2) Evaluating a learning algorithm

A learning algorithm is a function L that takes as input

- data set $\mathcal{D}_{train}$ as training data

and returns

- a classifier h.

A) Would running the learning algorithm L on two different training datasets $\mathcal{D}_{train_1}$ and $\mathcal{D}_{train_2}$ produce the same classifier? In other words, would $h_1 = L(\mathcal{D}_{train_1})$ be the same classifier as $h_2 = L(\mathcal{D}_{train_2})$? What if those training datasets were pulled from the **same** distribution?

Now, consider a situation in which someone is trying to sell you a new learning algorithm, and you want to know how good it is. There is an interesting result that says that without any assumptions about your data, *There is no learning algorithm that, for all data sources, is better than every other learning algorithm*. So, you'll need to assess the learning algorithm's performance in the context of a particular data source.

Check Yourself: *What is the difference between a classifier and a learning algorithm? Understanding the distinction will help you when thinking about this question.* (Stuck? Check the notes)

Assume that you have a generator of labeled data, $\mathcal{G}$, which will be suitable for your application. The learning algorithm's performance on $\mathcal{G}$-generated data will be a good predictor of the learning algorithm's performance on data from your application. (You can review how to evaluate learning algorithms in the notes)

B) Linnea Separatorix wants to evaluate a **learning algorithm**, and suggests the following procedure:

```
def eval_learning_alg(L, G, n):
    # draw a set of n training examples (points and labels)
    train_X, train_y = G(n)
    # run L
    h = L(train_X, train_y)
    # evaluate using your classifier scoring procedure, on some new labeled data
    test_data = G(n) # draw new set of test data
```

```
    return eval_classifier(h, test_data)
```

Check Yourself: *What are G and n in the code above?*

Explain why Linnea's strategy might not be so good.

C) Next, Linnea decides to generate one classifier h but evaluate that classifier with multiple (10) test sets in her `eval_learning_alg`. More specifically, Linnea changed her code above into:

```
def eval_learning_alg(L, G, n):
    # draw a set of n training examples (points and labels)
    train_X, train_y = G(n)
    # run L
    h = L(train_X, train_y)
    # evaluate using your classifier scoring procedure, on some new labeled data
    score = 0
    for i in range(10):
        test_data = G(n) # draw new set of test data
        score += eval_classifier(h, test_data)
    return score/10
```

Is Linnea's strategy good now? Explain why or why not.

Check Yourself: *How many classifiers is Linea generating and testing from the learning algorithm?*

D) Now design a better procedure for evaluating L. Write pseudocode for a procedure that takes L, $\mathcal{G}$ and n and returns a score. Say what the output score measures and what the best and worst values are.

```
def better_eval_learning_alg(L, G, n):
    # your procedure
```

E) Explain why your method might be more desirable than Linnea's.

3) Evaluating a learning algorithm with a small amount of data

In reality, it's almost never possible to have a generator of all the data you want; in fact, in some domains data is very expensive to collect, and so you are given a fixed, small set of samples. Now assume that you only have 100 labeled data points to use for training and testing/evaluation.

A) In the last section, you thought about how to evaluate a learning algorithm. Now that you are given only 100 labeled data points in total, how would you evaluate a learning algorithm? Specifically, how would you implement `better_eval_learning_alg` from **2C)** without $\mathcal{G}$ but instead with your 100 labeled data? (You don't need to write out new pseudocode for `better_eval_learning_alg` but still think about how you would implement it.)