CIS519: Applied Machine Learning	**Fall 2019**
Homework 1	
Handed Out: September 23	*Due: October 7, 11:59 PM*

Version 1

- Feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You should, however, **write down your solution yourself**. Please include at the top of your document the list of people you consulted with in the course of working on the homework.
- While we encourage discussion within and outside the class, cheating and copying code is strictly not allowed. Copied code will result in the entire assignment being discarded at the very least.
- Please use Piazza if you have questions about the homework. Also, please come to the TAs recitations and to the office hours.
- Handwritten solutions are not allowed. All solutions must be typeset in Latex. Consult the class' website if you need guidance on using Latex. If you don't have a lot of experience with Latex (or even if you do), we recommend using Overleaf (`https://www.overleaf.com`) to write your solutions. You will submit your solutions as a single pdf file (in addition to the package with your code; see instructions in the body of the assignment).
- The homework is due at 11:59 PM on the due date. We will be using Gradescope for collecting the homework assignments. You should have been automatically added to Gradescope. If not, please ask a TA for assistance. Please do **not** hand in a hard copy of your write-up. Post on Piazza and contact the TAs if you are having technical difficulties in submitting the assignment.
- Here are some resources you will need for this assignment
  - A zip that includes the datasets, Latex template, and Python template: `https://www.seas.upenn.edu/~cis519/fall2019/assets/HW/HW1/hw1-materials.zip`
  - The Student's t-table: `https://www.seas.upenn.edu/~cis519/fall2019/assets/HW/HW1/t-table.pdf`

Problems

1. Understanding Decision Tree Learning [20 Points]

(a) **[7 points]** In this problem, you are tasked with determining which feature the ID3 decision tree algorithm will pick to be the root of the decision tree. We will be working with the dataset presented in Table 1, described below.

Table 1 contains the aggregated statistics from a dataset that recorded whether a group of school students played outside based on if it was sunny out and if there was snow on the ground. The dataset has two binary features (`Sunny` and `Snow`), a binary label (`Play outside`), and the number of times the feature and label pair appear together in the dataset (`Count`). Although Table 1 does not list the individual instances of the dataset (e.g., you cannot determine how many times `Play outside` = yes, `Sunny` = yes, and `Snow` = no at the same time), it does contain all of the information necessary to run the ID3 algorithm.

Use the dataset in Table 1 to pick the feature at the root of the decision tree created by the ID3 algorithm. Recall that the ID3 algorithm will split on the feature that has

Play outside	Sunny	Count
yes	yes	25
yes	no	5
no	yes	6
no	no	14

Play outside	Snow	Count
yes	yes	14
yes	no	16
no	yes	4
no	no	16

Table 1: The `Play Outside` data set contains 50 instances, 30 of which are positive (`Play outside` = yes), and 20 of which are negative. The `Count` column marks how many times each label and feature pair appeared in the datset. For example, it was sunny and the students did not play outside 6 times.

the largest information gain (or equivalently smallest conditional entropy). Here are the definitions of entropy and information gain that you may find useful (equations use the same notation as in the lectures):

$$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

$$\text{Gain}(S, a) = \text{Entropy}(S) - \sum_{v \in \text{values}} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v)$$

where S_v is the subet of S for which feature a has value v. Show the calculations you performed to come up with your answer.

(b) **[7 points]** Entropy is one of many statistics that can be used to decide which feature to split on in a decision tree. In this problem, you will use a different statistic to build a full decision tree on a small dataset presented in Table 2, a modified version of the Balloons dataset.[1]

Color	Size	Act	Age	Inflated	Count
Blue	Small	Stretch	Adult	**F**	4
Blue	Small	Dip	Adult	**T**	1
Blue	Small	Dip	Child	**F**	2
Blue	Large	Stretch	Child	**T**	2
Blue	Large	Dip	Adult	**F**	1
Blue	Large	Dip	Child	**F**	2
Red	Small	Dip	Child	**F**	2
Red	Large	Stretch	Child	**T**	3

Table 2: The `Balloons` data set

The Balloons dataset has 4 features (`Color`, `Size`, `Act`, `Age`) that each can have two values and a binary label (`Inflated`). The `Count` column indicates how many times that instance appears in the dataset.

Instead of constructing a decision tree using entropy, you will use a new statistic called the minimum error, defined as:

$$\text{MinError}(S) = \min\{p_-, p_+\}$$

[1] You can learn more about this data set at `http://archive.ics.uci.edu/ml/datasets/Balloons`

MinError is computed over all of the instances remaining at the particular node in the decision tree. For example, if there are 10 instances in the node and 8 are positive, MinError $= \min\{0.2, 0.8\} = 0.2$. Like entropy, MinError can be thought of as a way to quantify how impure a node is.

Then, you will use a version of information gain modified to use MinError to select which feature to split on at a node in a decision tree.

$$\text{Gain}_{\text{ME}}(S, a) = \text{MinError}(S) - \sum_{v \in \text{values}} \frac{|S_v|}{|S|} \cdot \text{MinError}(S_v)$$

Using the version of information gain modified to use MinError, compute the full decision tree using the Balloons dataset.[2] You can type your decision tree as a series of if statements using Latex's `verbatim` environment, like below:

```
if Color = Blue:
  if Size = Small:
    Inflated = T
  if Size = Large:
    Inflated = F
if Color = Red:
  if Age = Adult:
    if Act = Stretch:
      Inflated = T
    if Act = Dip:
      Inflated = F
  if Age = Child:
    Inflated = T
```

(c) **[6 points]** Does ID3 guarantee a globally optimal tree? By globally optimal, we mean a decision tree that perfectly fits the training data and also has the minimum depth. Briefly justify your answer.

2. SGD, Decision Trees, and Feature Extraction [80 Points + 10 Points Extra Credit]

This problem involves programming exercises that rely on the Python libraries called NumPy and scikit-learn. Before you begin, ensure that you are working on a computer with these packages installed. See Appendix A for how to test you have the necessary software.

(a) **Training Classifiers [60 Points]:** The goal of this problem is to experiment with training and evaluating several different binary classifiers (all based on SGD and decision trees) on a small dataset with features that have been extracted for you. For

[2] If there is a tie between two features, select the feature which comes first in the table. That is, if `Color` and `Act` both have the same information gain, pick `Color` to split on.

each of the classifiers below, you will estimate how well it will do on testing data using cross-validation and then compare that estimate to the results on a real test set.

The dataset that you will be working with is an artificial dataset called Madelon that was created to test different binary classification models. It has 2,000 training and 600 testing instances, each with 500 features.

Algorithms Below are the details of the different classifiers that you need to train.

(i) **SGD**: Using the `SGDClassifier` scikit-learn class, train a SGD classifier on the training data. The class should be initialized as follows (see Appendix A for a brief example of how to train and predict labels using a scikit-learn model):[3]

```
from sklearn.linear_model import SGDClassifier
model = SGDClassifier(loss='log', max_iter=10000)
```

(ii) **Decision Tree**: Train a decision tree using the ID3 algorithm with the `DecisionTreeClassifier` class from scikit-learn. The model should be initialized as follows:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='entropy')
```

(iii) **Decision Stump of Depth 4**: Train a decision stump (a decision tree where the maximum depth is bounded) of depth 4 using the same `DecisionTreeClassifier` class from scikit-learn. The model should be initialized as follows:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

(iv) **Decision Stumps as Features**: This classifier combines many different decision stump classifiers into one classifier via SGD. At a high level, what you do here can be viewed as learning over a transformation of the feature space. Rather than working in the original feature space of the data (of dimensionality 500 in this case) you will first transform this space to a new feature space, where each feature is a *function of the original features* (It will be a learned function). We chose the new feature space to have dimensionality of 50, and each of the 50 features will be the output of a decision stump – a DT of depth 4. Once you generate these 50 transformation functions you will convert each of the original examples to an example in the new space by evaluating each of the 50 decision stumps on each of the original examples, training and test examples. Once you do that, you have transformed the original data set to a new one, of dimensionality 50, and you will run a learning algorithm (SGD) on the new dataset.

For explaining how to build this model, we will denote the training data to be an $N_{\text{train}} \times k$ matrix $\mathbf{X}_{\text{train}}$, where N_{train} is the number of training examples and

[3]Directly copying and pasting this code into Python may cause problems because the quotes are the wrong character

k is the number features. Similarly, we denote an $N_{\text{eval}} \times k$ matrix $\mathbf{X}_{\text{eval}}$ to be the dataset that you want to use to evaluate your model (either the held-out data in cross-validation (see next section) or the test data).

First, you will need to train 50 *different* decision stumps on the training data. In order to make sure that the decision stumps are different, each stump should be trained on a random 50% of the input features. For example, if there are 30 features in the training data, each time you train a new stump, randomly pick 15 of those features and train the stump on only those 15 features.

Each of your trained decision stumps can be used to predict a binary label for each original k-dimensional instance $\mathbf{x}$. This binary label can be viewed as a new feature that represents the output of that decision tree. If you repeat this for all 50 stumps, the k-dimensional $\mathbf{x}$ can be converted into a 50-dimensional $\mathbf{x}'$ where each dimension is a label predicted by one of the 50 stumps. This process can be applied to all N_{train} instances, producing a new $N_{\text{train}} \times 50$ feature matrix $\mathbf{X}'_{\text{train}}$.

Specifically, the ith row and jth column of $\mathbf{X}'_{\text{train}}$ is the output from the jth stump on the ith original training instance

```
# this is pseudocode
X_prime_train[i, j] = stumps[j].predict(X_train[i])
```

Using this new feature matrix $\mathbf{X}'_{\text{train}}$ and the original labels, train a SGD classifier using the `SGDClassifier`. This is the final model.

In order to evaluate this model on $\mathbf{X}_{\text{eval}}$, you have to repeat the same process above to convert $\mathbf{X}_{\text{eval}}$ into $\mathbf{X}'_{\text{eval}}$. That is, take $\mathbf{X}_{\text{eval}}$, use the *same* stumps that were trained on $\mathbf{X}_{\text{train}}$ and convert it into $\mathbf{X}'_{\text{eval}}$. Then, use the SGD model trained on $\mathbf{X}'_{\text{train}}$ to predict labels for $\mathbf{X}'_{\text{eval}}$.

Evaluation: Cross-Validation In order to evaluate the performance of each of the above classifiers, you will use 5-fold cross-validation. The purpose of cross-validation is to estimate how a model will do on the testing data without repeatedly running the model on the test data. This estimate can be used to select hyperparameters (the learning rate, decision tree depth, etc.) that perform the best without looking at the testing data.

To run 5-fold cross-validation, you will partition the training data into 5 equally-sized disjoint sets (this has already been done for you). Then you will train each model 5 different times, each time training on 4 folds and evaluating on a different held-out fold. For example, here are the training and evaluation folds for each of the 5 iterations of cross-validation:

Training Folds	Evaluation Fold
1 2 3 4	5
1 2 3 5	4
1 2 4 5	3
1 3 4 5	2

2 3 4 5 1

To measure the performance of the classifier, you will compute two different accuracies on each iteration of cross-validation, then average the accuracies across iterations at the end. On the ith iteration of cross-validation, you should compute the accuracy of the classifier when evaluating on the same 4 training folds that were used to train the model, denoted $\text{acc}^i_{\text{train}}$, and the accuracy of the classifier when evaluating on the 1 held-out fold, denoted $\text{acc}^i_{\text{held-out}}$. Finally, the accuracies should be averaged across iterations:

$$\text{acc}_{\text{train}} = \frac{1}{k}\sum_{i=1}^{k} \text{acc}^i_{\text{train}}$$

$$\text{acc}_{\text{held-out}} = \frac{1}{k}\sum_{i=1}^{k} \text{acc}^i_{\text{held-out}}$$

$\text{acc}_{\text{held-out}}$ will be your estimate of how well the model will perform on the true test data and $\text{acc}_{\text{train}}$ is a useful statistic to use for analysis.

Additionally, you should compute a 95% confidence interval for the accuracy estimates using the Student's t-distribution. Confidence intervals require computing the standard deviation of the accuracies

$$\sigma_{\text{train}} = \text{std}\left([\text{acc}^1_{\text{train}}, \text{acc}^2_{\text{train}}, \dots]\right)$$

$$\sigma_{\text{held-out}} = \text{std}\left([\text{acc}^1_{\text{held-out}}, \text{acc}^2_{\text{held-out}}, \dots]\right)$$

The confidence interval is computed by

$$\bar{x} \pm t \cdot \frac{S}{\sqrt{n}}$$

where $\bar{x}$ is the sample mean (either $\text{acc}_{\text{train}}$ or $\text{acc}_{\text{held-out}}$), S is the sample standard deviation (either σ_{train} or $\sigma_{\text{held-out}}$), and n is the number of observations, 5. t is the critical value, which you can look up using the provided t-table.

Our implementation takes about 2-3 minutes to run the full cross-validation for all of the models. When you are developing your code, we suggest reducing the number of training examples that you use, which will speed up training. Then when you need to generate the final numbers, use all of the provided data.

Evaluation: Testing After you have estimated the accuracy of each model via cross-validation by computing $\text{acc}_{\text{held-out}}$, you should compare your estimate to how well the classifier actually does on a real test set. For each model, train it once on *all 5* training folds and compute the accuracy on the testing data (which was never used during cross-validation), denoted acc_{test}.

Then, use the provided Python code to plot $\text{acc}_{\text{train}}$, $\text{acc}_{\text{held-out}}$, and acc_{test} with error bars for the training and held-out values. The plot together with the cross-validation code is 48 points. You should turn in this plot in addition to your answers to the following questions.

1. **[3 Points]** Rank the classifiers by the held-out and testing performances. Is the order the same? Which did the best? The worst? Was your testing performance estimate $\text{acc}_{\text{held-out}}$ similar to the real testing performance acc_{test}? Comment on your observations.
2. **[3 Points]** Which classifier had the highest training accuracy? Why?
3. **[3 Points]** Were the results statistically significant according to the 95% confidence interval?[4] If not, what could you change about the experiment to have a tighter confidence interval?
4. **[3 Points]** Why is it important to be able to estimate the performance of a classifier using a technique like cross-validation without repeatedly running that classifier on the testing data?

(b) **Feature Extraction [20 Points]**: In the previous problem, you trained several different binary classifiers on a dataset that had features that were computed for you. In this problem, you will generate the features yourself on a new dataset then run a subset of the experiments from the previous problem using the features you create.

The dataset you will be working with is (a modified version of) the Badges dataset from class. There are 1,000 training and 1,000 testing names. Each name has been given a 0 or 1 label based on some secret labelling function that you are going to try and learn with the binary classifiers. Since we only provide you with the names, you need to convert them into features using the process described below.

For every name, you will generate 260 features (i.e., the final feature matrix should be of size $N \times 260$). Each feature will be a binary indicator feature which marks whether the letter at a specific position in the name is equal to a predefined letter. Specifically, for each of the 5 first letters of the first and last names, there will be 26 different binary features. Exactly one of these 26 features will be 1 for each position in the name, and that feature is the index of the letter in the alphabet. If a name is shorter than 5 letters long, all of the corresponding 26 features will be 0.

Concretely, assume there are only 2 letters in the alphabet, {`a`, `b`}, you are creating features for the name `ab aa` using only the first two letters of the first and last names. The corresponding features would look like this:

```
Feature ID:    0 1 2 3 4 5 6 7
Feature Value: 1 0 0 1 1 0 1 0
```

Feature 0 indicates whether or not the first letter of the first name is an `a`, feature 1 indicates whether or not the first letter of the first name is a `b`, feature 2 indicates whether or not the second letter of the first name is an `a`, ..., feature 7 indicates whether or not the second letter of the last name a `b`.

You should create this $N \times 260$ feature matrix for the Badges data using the first 5 letters of the first and last names and all 26 letters of the alphabet (26 letters in the alphabet $\times$ (5 letters in first name + 5 letters in last name) = 260 features). **[18 Points]** Implement this in the `compute_features` function and turn it in to Gradescope for automatic testing. You should copy this function (and any necessary python

[4] `https://en.wikipedia.org/wiki/Student%27s_t-test`

imports) into a file called `hw1.py` and upload that to Gradescope. Do not try to upload the Jupyter Notebook.

After you have successfully implemented the featurization, train and test all of the models from the previous problem on the badges data. You do **not** have to run the full cross-validation (of course you can if you want to). You only need to train one model on all of the training data and test that model on all of the testing data. Report the accuracies on the training and testing data and answer the following questions:

1. **[2 Points]** Rank the models by their training and testing accuracies. Is the order the same? Is the model which performed the best on the Madelon test dataset the same as the one which performed the best on the Badges test dataset?

Extra Credit [10 Points] Finally, for extra credit, come up with new features of your own in addition to the 260 above and see how well you can do on the test set. **[5 Points]** Explain what features you used and the impact they had on the model accuracy.

Additionally, we have provided you with an extra set of names that we did not give you the labels for. Use your best model and predict labels on this set of names and upload them to the Gradescope unit test which will compute your model's accuracy. The file with the labels should be called `labels.txt` and there should be one label, either 0 or 1, per line. **[5 Points]** If the accuracy is above a specific threshold, the test will pass and you will get extra credit. You will still receive extra credit for experimenting with new features even if you are unable to pass the accuracy unit test.

Submission Instructions

We will be using Gradescope to turn in both the Python code and writeup pdfs. You should have been automatically added to Gradescope. If you do not have access, please ask the TA staff on Piazza.

For this homework assignment, there are two Gradescope assignments:

- "Homework 1 - Code": This is the assignment where you should upload your implementation of the `compute_features` function in a file called `hw1.py` and optional name labels in a file called `labels.txt`. Additionally, you should submit all of your code when you have finished the assignment (upload the individual Python files, zip them together, or upload the Jupyter Notebook as a pdf).
- "Homework 1 - PDF": This is the assignment where you should upload your writeup as a PDF.

A Required Software

Before starting the programming exercises, you will need to make certain that you are working on a computer with the following software:

1. python 3.6

2. numpy (`http://www.numpy.org/`)
3. scikit-lean (`http://scikit-learn.org/stable`)

To make sure that you have these libraries, run the following code:

```
from sklearn import tree
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
clf.predict([[2, 2]])
```

If this code runs without error and gives you the following output:

```
array([1])
```

then everything should be configured correctly for this homework.

You can add `from sklearn import linear_model` and substitute the 4th line with `linear_model.SGDClassifier(...)` for training an SGD model.

Also, (optionally) try to explore the method `tree.export_graphviz(...)` to print the decision tree graphically.