Assignment 1
CSE 446: Machine Learning

University of Washington

0 Policies [0 points]

Please explicitly answer the three questions below and include your answers marked in a "problem 0" in your solution set. If you have not included these answers, your assignment will not be graded.

Readings: Read the required material.

Submission format: Submit your report as a *single* pdf file and submit all your code in a gzipped tarball named `code.tgz`. The report (in a single pdf file) must include all the plots and explanations for programming questions (if required). We highly recommend typesetting your scientific writing using LaTeX. Some free tools that might help: ShareLaTeX (`www.sharelatex.com`), TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Detexify[2] (online). If you want to type, but don't know (and don't want to learn) LaTeX, consider using a markdown editor with real-time preview and equation editing (e.g., `stackedit.io`, `marxi.co`). Writing solutions by hand will be accepted provided they are neat; written solutions need to be scanned and included into a single pdf.

Written work: Please provide succinct answers *along with succinct reasoning for all your answers*. Points may be deducted if long answers demonstrate a lack of clarity. Similarly, when discussing the experimental results, concisely create tables and figures to organize the experimental results.

Python source code: for the programming assignment. Please note that we will not accept Jupyter notebooks. Submit your code together with a neatly written README file to instruct how to run your code with different settings (if applicable). We assume that you always follow good practice of coding (commenting, structuring); these factors are not central to your grade.

Coding policies: You must write your own code. You are welcome to use any Python libraries for data munging, visualization, and numerical linear algebra. Examples includes Numpy, Pandas, and Matplotlib. You may **not**, however, use any machine learning libraries such as Scikit-Learn or TensorFlow. If in doubt, post to the message boards or email the instructors.

Collaboration: It is acceptable for you to discuss problems with other students; it is not acceptable for students to look at another students written answers. It is acceptable for you to discuss coding questions with others; it is not acceptable for students to look at another students code. Each student must understand, write, and hand in their own answers. In addition, each student must write and submit their own code in the programming part of the assignment.

Acknowledgments: We expect the students not to refer to or seek out solutions in published material from previous years, on the web, or from other textbooks. Students are certainly encouraged to read extra material for a deeper understanding.

0.1 List of Collaborators

List the names of all people you have collaborated with and for which question(s).

0.2 List of Acknowledgements

If you do inadvertently find an assignment's answer, acknowledge for which question and provide an appropriate citation (there is no penalty, provided you include the acknowledgement). If not, then write "none".

0.3 Certify that you have read the instructions

Please make sure to read and follow these instructions. Write "I have read and understood these policies" to certify this.

1 Criteria for Choosing a Feature to Split [20 points]

When choosing a single feature to use to classify—for example, if we are greedily choosing the next feature to use for a subset of the data inside of a decision tree—we proposed in lecture to choose the feature that will give the greatest reduction in error.

Let D be the dataset under consideration, n be the number of examples in D with label 0 (negative examples), and p be the number of examples in D with label 1 (positive examples).

1.1 Not Splitting [5 points]

Suppose we have reached the maximum depth allowed by our learning algorithm; we may not use any more features. At this point, we have a partition of the data, D' (which is a subset of D) and it contains n' negative examples and p' positive examples. Assuming that our algorithm is deterministic, what's the smallest number of mistakes (in D') we can make, and how do we obtain it? (Give a formula in terms of the symbols we have introduced above.)

1.2 Splitting [5 points]

Consider a binary feature ϕ with the following contingency table for D':

y	ϕ	
	0	1
0	n_0	n_1
1	p_0	p_1

Note that $p' = p_0 + p_1$ and $n' = n_0 + n_1$. If we split D' based on feature ϕ, how many mistakes will we make (in D')? (Give a formula in terms of the symbols we have introduced.)

Another way to think about the which-feature-if-any-to-split-on decision is to *maximize* the number of mistakes that we would *correct* by splitting (as opposed to not splitting). This is given by subtracting the answer to 1.2 from the answer to 1.1. If we divide this value by $|D'|$ (the number of examples in D'), then we have the absolute reduction in error *rate* achieved by making the split (as opposed to not splitting). From here on, call this value "error rate reduction."

Before proceeding, think carefully about this question: is it possible for a split to *increase* the error rate on the training set? (No points for this, but you should have a confident answer in your mind.)

1.3 Mutual Information [10 points]

A rather different way to think about the decision is to use information theory. Simply put, we want to choose a feature ϕ that gives us as much information as possible about the label, within D. Information theory gives us a way to think about measuring information.

Mutual information[1] is an association score between two random variables A and B. For simplicity's sake, we assume both of them are binary and take values in $\{0, 1\}$. The mutual information between A and B is given by:

$$I(A; B) = \sum_{a \in \{0,1\}} \sum_{b \in \{0,1\}} P(A = a, B = b) \log \frac{P(A = a, B = b)}{P(A = a) \cdot P(B = b)} \tag{1}$$

If we use base-2 logarithms, then this quantity is measured in bits. It is the average number of bits that observing random variable A tells you about random variable B, and vice versa, across many trials. If A and B are perfectly predictive of each other (either always the same or always different), then $I(A; B) = 1$ bit. If they carry no information about each other, then $I(A; B)$ approaches 0.

The formula for the mutual information between the binary value of feature ϕ and the binary label y, under the distribution observed in D, is as follows; we use $|D|$ to denote the size of D; note that $|D| = p + n = p_0 + n_0 + p_1 + n_1$. This is obtained by thinking about the feature and the label as two random variables and plugging in relative frequency estimates of the probabilities.

$$I(\phi; Y) = \frac{n_0}{|D|} \log \frac{|D| n_0}{(n_0 + p_0) n} + \frac{p_0}{|D|} \log \frac{|D| p_0}{(n_0 + p_0) p} + \frac{n_1}{|D|} \log \frac{|D| n_1}{(n_1 + p_1) n} + \frac{p_1}{|D|} \log \frac{|D| p_1}{(n_1 + p_1) p} \tag{2}$$

We've done a bit of simplification on the formula above, but it's still pretty daunting. There are two ways to go about gaining some intuition here. The first is to learn about information theory; we hope you'll consider doing that.[2]

[1] Strictly speaking, this quantity is known as *average* mutual information. When used for building decision trees, it is often called "information gain."

[2] Cover and Thomas [1] is an excellent introductory textbook, an older edition of which your instructor owns a copy that is perpetually borrowed by a Ph.D. student—a good sign that it's useful. If you find an amazing tutorial online, please recommend it to the class!

The second way is to explore how these functions (error rate reduction and mutual information) change for features with different relationships to the labels. Suppose that $|D| = 1000$. Let's start by fixing $n = p = 500$; our dataset is perfectly balanced. Imagine that the dataset-creators have been quite generous to us and provided us with 499 features. For i from 1 to 499, let the ith feature ϕ_i have values $n_0 = p_1 = i$ and $n_1 = p_0 = 500 - i$. (Remember: this is a contrived example. Real world data will never be so perfectly constructed!)

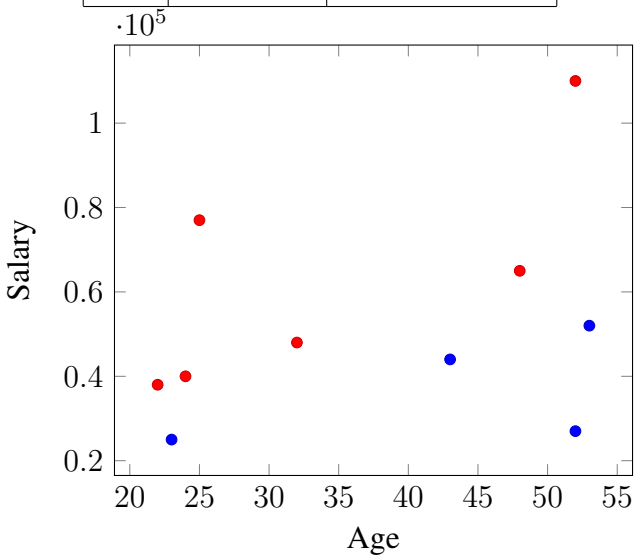
Generate a plot where the x-axis is i (the index of one of the 499 features) and the y-axis is error rate reduction for that feature. On the same plot, in a different color, show the mutual information between the feature and the label. What do you notice, and what conclusions can you draw?

2 Decision Stumps and Trees [40 points]

It might be helpful for you to draw the decision boundary, in the figure, of each classifier in each part.

Consider the problem of predicting whether a person has a college degree based on age and salary. The table and graph below contain training data for 10 individuals.

Age	Salary ($)	College Degree
24	40,000	Yes
53	52,000	No
23	25,000	No
25	77,000	Yes
32	48,000	Yes
52	110,000	Yes
22	38,000	Yes
43	44,000	No
52	27,000	No
48	65,000	Yes



2.1 Build Two Decision Stumps [10 points]

In class, we focused on discrete features. One way to deal with continuous (or ordinal) data is to define binary features based on thresholding of continuous features like Age and Salary. For example, you might let:

$$\phi(x) = \begin{cases} 0 & \text{if Age} \leq 50 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

The choice of the threshold 50 is, on the surface, arbitrary. In practice, though, for a given feature, you should only consider the best threshold for that feature. While the search for a threshold might seem daunting, the only thresholds worth considering will be midpoints between consecutive feature values, after sorting.[3]

For each of the two features in this dataset (Age and Salary), generate a plot showing the training set accuracy (on the y-axis) as a function of the threshold value (on the x-axis). You will probably want to automate the calculations. Do the same for mutual information as a criterion. Report the training-set error rates of all four decision stumps. Did the criterion for selecting a threshold make a difference?

2.2 Build Decision Trees [14 points]

Build a decision tree for classifying whether a person has a college degree by greedily choosing threshold splits. Do this two ways: by counting errors (as shown in lecture) and using mutual information (as derived above). Show both decision trees and report their training set error rates.

2.3 Multivariate Decision Tree [12 points]

Multivariate decision trees are a generalization of the "univariate" decision trees we've discussed so far. In a multivariate decision tree, more than one attribute can be used in the decision rule for each split. That is, from a geometric point of view, splits need not be orthogonal to a feature's axis.

For the same data, learn a multivariate decision tree where each split makes decisions based on the sign of the function $\alpha x_{\text{Age}} + \beta x_{\text{Income}} - 1$. Draw your tree, including the α, β, and the information gain for each split.

2.4 Discussion [4 points]

Multivariate decision trees have practical advantages and disadvantages. List an advantage and a disadvantage multivariate decision trees have compared to univariate decision trees.

[3] Imagine moving a threshold just a little bit, but not enough to change the decision for any data point. For example, the threshold in Equation 3 could be reduced to 49 or increased to 51, and there would be no effect on the classification decisions in our dataset.

3 A "Baby" No Free Lunch Theorem [15 points]

Suppose that Alice has a function $f : \{0,1\}^m \to \{0,1\}$ mapping binary sequences of length m to a label. Suppose Bob wants to learn this function. Alice will only provide labelled random examples. Specifically, Alice will provide Bob with a training set under the following data generating distribution: the input x is a uniformly at random binary string of length m and the corresponding label is $f(x)$ (the label is deterministic, based on Alice's function). Bob seeks to learn a function $\hat{f}$ which has low mis-classification expected loss.

Remember to provide short explanations.

3.1 The # of inputs [3 points]

How many possible inputs are there for m length binary inputs? And, specifically for $m = 5$, how many are there?

3.2 The # of functions [5 points]

The function f is one function out of how many possible functions acting on m length binary sequences? And, specifically for $m = 5$, how many possible functions are there?

3.3 Obtaining low expected loss [5 points]

Assume that Bob has no knowledge about how Alice chose her function f. Suppose Bob wants to *guarantee* that he could learn with an expected loss that is non-trivially better than chance, e.g. suppose Bob wants an expected mis-classification loss less than 25% (or any loss that is a constant less than 50%). Roughly (in "Big-Oh" notation), how many distinct inputs x does Bob need to observe in the training set in order to guarantee such an expected loss? Again, as a special case, how many distinct inputs do you need to see for $m = 5$? (You do not need a precise answer for the latter, just in the right ballpark.)

Some comments: Note that the required the training set size that Bob needs to observe is slightly larger than this number of distinct inputs, due to the "coupon collectors" problem where repeats occur. You do not need to write an answer to the following question though it is worth thinking about: is this number of distinct training examples notably different from the number of possible functions? why or why not?

3.4 Discussion [2 points]

Now suppose Bob obtains a training set whose size is much smaller than the correct answer to part 3.3 and that, using this training set, Bob believes he has learned a function $\hat{f}$ whose error rate is non-trivially better than guessing. How can Bob convince another person, say Janet, that he has learned something non-trivially better than chance? Further, suppose that Bob did convince Janet of this (assume Janet is not easily fooled), what does this say about Bob's knowledge with regards to how Alice chose her function?

4 Implementation: K-Means [40 points]

In class we discussed the K-means clustering algorithm. You will implement the K-means algorithm on digit data. A template has been prepared for you: it is named `kmeans_template.py` and is available on Canvas in the A1.tgz download. Your job is to fill in the functions as described below.

4.1 Data

The data are held in two files: one for the inputs and one for the output. The file `digit.txt` contains the inputs: namely, 1000 observations of 157 pixels (a randomly chosen subset of the original 784) from images containing handwritten digits. The file `labels.txt` contains the true digit label (**one**, **three**, **five**, or **seven**, each coded as a digit). The Python template handles reading in the data; just make sure that all the files live in the same directory as your code.

The labels correspond to the digit file, so the first line of `labels.txt` is the label for the digit in the first line of digit.txt.

4.2 Algorithm [15 points]

Your algorithm will be implemented as follows:

1. Initialize k starting centers by randomly choosing k points from your data set. You should implement this in the `initialize` function. [5 points]
2. Assign each data point to the cluster associated with the nearest of the k center points. Implement this by filling in the `assign` function. [5 points]
3. Re-calculate the centers as the mean vector of each cluster from (2). Implement this by filling in the `update` function. [5 points]
4. Repeat steps (2) and (3) until convergence. This wrapping is already done for you in the `loop` function, which lives inside the `cluster` function.

4.3 Within-Group Sum of Squares [5 points]

One way to formalize the goal of clustering is to minimize the variation *within* groups, while maximizing the variation *between* groups. Under this view, a good model has a low "sum of squares" within each group.

We define sum of squares as follows. Let $\boldsymbol{\mu}_k$ (a vector) be the mean of the observations (each one a vector) in kth cluster. Then the within group sum of squares for kth cluster is defined as:

$$SS(k) = \sum_{i \in k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2^2$$

where "$i \in k$" ranges over the set of indices i of observations assigned to the kth cluster. Note that the term $\|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2$ is the Euclidean distance between $\mathbf{x}_i$ and $\boldsymbol{\mu}_k$, and therefore should be

calculated as $\|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2 = \sqrt{\sum_{j=1}^{d} (\mathbf{x}_i[j] - \boldsymbol{\mu}_k[j])^2}$, where d is the number of dimensions. Note that that term is squared in $SS(k)$, cancelling the square root in the formula for Euclidean distance.

If there are K clusters total then the "sum of within-group sum of squares" is the sum of all K of these individual $SS(k)$ terms.

In the code, the within-group sum of squares is referred to (for brevity) as the *distortion*. Your job is to fill in the function `compute_distortion`.

4.4 Mistake Rate [5 points]

Typically K-means clustering is used as an exploration tool on data where true labels are not available, and might not even make sense. (For example, imagine clustering images of all of the foods you have eaten this year.)

Given that, here, we know the *actual* assignment labels for each data point, we can analyze how well the clustering recovered the true labels. For kth cluster, let the observations assigned to that cluster *vote* on a label (using their true labels as their votes). Then let its assignment be the label with the most votes. If there is a tie, choose the digit that is smaller numerically. This is equivalent to deciding that each cluster's members will all take the same label, and choosing the label for that cluster that lead to the fewest mistakes.

For example, if for cluster two we had 270 observations labeled **one**, 50 labeled **three**, 9 labeled **five**, and 0 labeled **seven** then that cluster will be assigned value **one** and had 50 + 9 + 0 = 59 mistakes.

If we add up the total number of mistakes for each cluster and divide by the total number of observations (1000) we will get our total mistake rate, between 0 and 1. Lower is better.

Your job is to fill in the function `label_clusters`, which implements the label assignment. Once you have implemented this, the function `compute_mistake_rate` (already written) will compute the mistake rate for you.

4.5 Putting it All Together [15 points]

Once you have filled in all the functions, you can run the Python script from the command line with `python kmeans_template.py`. The script will loop over the values $K \in \{1, 2, \ldots, 10\}$. For each value of k, it will randomly initialize 5 times using your initialization scheme, and keep the results from the run that gave the lowest within-group sum of squares. For the best run for each K, it will compute the mistake rate using your cluster labeling function.

The code will output a figure named `kmeans.png`. The figure will have two plots. The top one shows within-group sum of squares as a function of K. The bottom one shows the mistake rate, also as a function of K.

Write down your thoughts on these results. Are they what you expected? Did you think that within-group sum of squares and mistake rate would go up or decrease as K increased? Did the plots confirm your expectations?

References

[1] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.