

Question 1.
Optical Digit Recognition

This question (the only one on the homework) is designed to lead you through, step by step, the process of developing and testing a new machine learning algorithm (well, new to you that is). Make sure you understand each step before moving on to the next one.¹ Do not get the steps confused; we are doing something different at each step.

part a. [10 points]

Let's first develop a very simple classifier. Because we will be considering such a simple hypothesis space, we are not worried about over-fitting, so we will use (weighted) empirical risk minimization.

For this part assume we have a binary classification problem ($\mathcal{Y} = \{-1, +1\}$). Furthermore assume that each input is a vector of real-valued numbers. Finally let us assume (for reasons that will hopefully become clear in part e) that each training example is given a different weight (w_i for example number i) specifying its relative importance. That is, instead of minimizing the sum of the 0-1 loss on each training data point, we want to minimize the weighted sum of the 0-1 loss:

$$\sum_i w_i (1 - \delta(y_i - f(x_i)))$$

where δ returns 1 if the argument is equal to 0 and 0 otherwise.

All that is left is to pick our hypothesis space. We only want a very simple (perhaps not that powerful) classifier. So, let's let our hypothesis space be the set of functions that check a single feature (or dimension) of the input vector and compare it to a single threshold and return +1 if the feature is on one side of the threshold and -1 if it is on the other side. Such a hypothesis space can be parameterized by three things: the feature index upon which to make a decision, the threshold against which to compare, and which side of the threshold to predict as +1.

Give the pseudo-code for an algorithm to optimize the weighted empirical risk of a dataset over the hypothesis space defined above. Your algorithm should run in time $O(dm \log m)$ where m is the number of data points and d is the number of features (dimensions) of the input vectors.

[Hint: you do not have to search over all possible split points. Many split points are all functionally equivalent on the given dataset (they will all result in the same weighted loss). Picking any one of them is fine.]

part b. [15 points]

So, how well could such a simple classifier do? Let's find out. First, write up your algorithm from above in C++. You will want to read to the end of this part so that you will know how to structure your algorithm to connect with the other pieces.

Supplied with this assignment is a digit classification task. The files `optdigits.tra` and `optdigits.tes` each contain 8x8 gray-scale images of hand-written digits.² The code in `dataset.cpp` will help you to read in these datasets and even give crude ASCII images of the digits.

There are, not surprisingly, 10 classes. However, you only constructed an algorithm for binary classification. This is easily solved, though. We can simply consider all possible 2-way classification tasks. That is, we can consider the

¹To be clear, you could do all of the theory and then go back and do all of the programming. You would gain a better understanding of how to structure your code, but you would lose some of the natural progression of the problem set. However, I do not advocate any other reordering of the problem set.

²This is not simulated data, but rather actual data from 43 different people. Interestingly, the test set is from a different set of individuals than the training set, so this assignment also tests your classifier's ability to generalize across people.

task of separating the “0”s from the “1”s, and the task of separating the “0”s from the “2”s, and so on. So, we will be training up 45 different classifiers, each on a subset of the full data set.

For each classifier, we would like to test how well it does. Unfortunately, we do not have access to the “true distribution” of pairs of images and digit-labels. So, instead we will use a test set (a set of data that was not seen during learning). Train your algorithm on the data in `optdigits.tra` and report your performance on the data in `optdigits.tes`.

You should train all 45 different 2-way classifiers (for this part, all examples have the same weight). Then, for each one, calculate the percentage of the test set examples (only the germane ones, of course) each classifier correctly classifies. Report these results in a 10x10 table where entry i, j is the test result of deciding between digit i and digit j . This is a symmetric table and the diagonal elements can be left blank.

part c. [10 points]

But what about the full (10-way) classification task? We can solve that without much further work by having the 45 classifiers from above each “vote” on which class they think the example should be in. Each classifier will only be able to pick between two different classes. Once all of the votes are cast, your classifier should report the class with the most votes.³

Write code to implement this more advanced classifier and report the percentage correct on the test set for this 10-way classification task.

Additionally, report the confusion matrix for this task. (The i, j element of a confusion matrix is the number of times the algorithm predicted j when the correct class was i . The diagonal elements are the “good” answers and the off-diagonals are the “errors.”)

[Hint: your testing accuracy should be about 78%.]

part d. [20 points]

Well, that accuracy rate isn’t really good enough, so let’s see about constructing a better classifier. Yet, we don’t want to throw away all of the good work we did in part a. (We are ignoring part b and going back to a binary classification task in this part).

The simple classifiers from before gave some predictive power. We will try to combine multiple such classifiers to generate a more powerful classifier. Let imagine we have a sequence of n such simple classifiers $g_1(\cdot), g_2(\cdot), \dots, g_n(\cdot)$. Each one has its own parameters (we are ignoring for the moment how those might have been chosen) and returns either -1 or $+1$ depending on its prediction.

We can combine these classifiers into a single classifier by taking a linear combination of their outputs and thresholding it:

$$\begin{aligned} s(x) &= \sum_{k=1}^n \alpha_k g_k(x) \\ f(x) &= \theta(s(x)) \end{aligned} \tag{1}$$

where θ returns -1 if its argument is negative and $+1$ otherwise. We can then define the margin⁴ of data point i in the training set to be

$$m_i = y_i s(x_i) .$$

This measures how far the data point is from being correctly classified. The multiplication by y_i ensures that it is positive if the point is being correctly classifier by f and negative if it is not. Stare at this for a while and make sure it makes sense to you.

³If you do have ties, break them by selecting the class with the smallest label.

⁴This definition of margin is different from the one used in class for support vector machines. The SVM margin is a function of the dataset. This margin is a function of a single training point.

One way of trying to optimize such margins (to make sure they are large) is to minimize the following risk instead of the empirical risk.

$$\sum_i e^{-m_i} . \quad (2)$$

There are theoretical reasons for minimizing such an expression. But, for this problem set you will have to take it on faith that such a minimization serves to reduce the true error as well as the training error.

So, let's imagine we have a classifier as described in equation 1 and we would like to improve it by adding one additional term to the sum. That is, we have a function $s(\cdot)$ (and thereby $f(\cdot)$) and we'd like to construct a new one

$$f'(x) = \theta(s(x) + \alpha' g'(x))$$

where α' is the new weight (to become α_{n+1}) and $g'(x)$ is the new simple classifier (to become $g_{n+1}(x)$).

For the moment, let's assume that $g'(x)$ is given (*i.e.* its parameters are fixed). We want to find the α' that minimizes the exponential risk of equation 2 for this new classifier $f'(\cdot)$. Differentiate equation 2 with respect to α' and set it equal to zero. Derive a closed-form expression for α' . Show your steps. Feel free to define extra variables to make your expressions easier to understand.

[Hint: Think about dividing the dataset into those points for which $g'(x_i)$ correctly predicts y_i and those points for which it does not.]

part e. [20 points]

Finding the optimal g' is a little trickier (but in some ways simpler). By selecting g' , you are picking which data points are correctly classified and which are not. Obviously, you don't have complete control; you can only select classifiers in the hypothesis space. However, given the expression for α' derived above, you should be able to write down a selection criteria that exactly corresponds to the one used in part a. Show your derivation.

[Hint: This one may take a bit to get your head around. Your goal is to transform the problem of picking the g' classifier into a weighted empirical risk minimization problem.]

part f. [10 points]

Parts d and e should suggest a method for building a classifier in the form of equation 1. Write down the pseudo-code for such an algorithm. Pay careful attention to the termination conditions for this algorithm. Try to come up with a natural termination condition.

[Hint: Negative α values are not a good idea.]

part g. [15 points]

Implement the classification algorithm of section f (and to some degree everything before it). Re-run the tests of parts b and c for this new algorithm.

[Hint: In order to reduce the running time, it is okay in this part to replace the nice termination condition of part f with a fixed termination condition. 30-50 rounds are recommended for good results.]