Assignment 2

Supervised Learning

Deadline: October 22, 11:59pm
Perfect score: 100 points

Assignment Instructions:

Teams: Assignments should be completed by pairs of students. No additional credit will be given for students working individually. You are strongly encouraged to form a team of two students. Please inform the TAs as soon as possible about the members of your team so they can update the scoring spreadsheet and no later than October 17. In order to do so, please use the following sign-up form link: `https://goo.gl/j3cWqD`. You can find the TAs' contact information under the course's website: `http://www.pracsyslab.org/cs440`). Failure to inform the TAs on time will result in a lower grade in the assignment.

Submission Rules: Submit your reports electronically as a PDF document through Sakai (`sakai.rutgers.edu`). For programming questions, you need to also submit a compressed file via Sakai, which contains your results and/or code as requested by the assignment. For your reports, do not submit Word documents, raw text, or hardcopies etc. Make sure to generate and submit a PDF instead. *Each team of students should submit only a single copy of their solutions and indicate all team members on their submission*. Failure to follow these rules will result in a lower grade in the assignment.

Program Demonstrations: You will need to demonstrate your program to the TAs and graders on a date after the deadline. The schedule of the demonstrations will be coordinated by the TAs. During the demonstration you have to use the files submitted on Sakai and execute it either on your laptop computer (easier) or an available machine, where the TAs and graders will be located (you probably need to coordinate this ahead of time). You will also be asked to describe the architecture of your implementation and algorithmic aspects of the project. You need to make sure that you are able to complete the demonstration and answer the TAs' questions within the allotted 12 minutes of time for each team. If your program is not directly running on the computer you are using and you have to spend time to configure your computer, this counts against your allotted time.

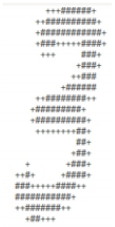
Late Submissions: No late submissions are allowed. You will be awarded 0 points for late assignments!

Extra Credit for LaTeX: You will receive 10% extra credit points if you submit your answers as a typeset PDF (i.e., using LaTeX). Resources on how to use LaTeX are available on the course's website (`http://www.pracsyslab.org/cs440`). There will be a 5% bonus for electronically prepared answers (e.g., on MS Word, etc.) that are not typeset. If you want to submit a handwritten report, scan it and submit a PDF via Sakai. We will not accept hardcopies. If you choose to submit scanned handwritten answers and we are not able to read them, you will not be awarded any points for the part of the solution that is unreadable.

Precision: Try to be precise in your description and thorough in your evaluation. Have in mind that you are trying to convince a very skeptical reader (and computer scientists are the worst kind...) that your answers are correct.

Collusion, Plagiarism, etc.: Each team must prepare its solutions independently from other teams, i.e., without using common code, notes or worksheets with other students or trying to solve problems in collaboration with other teams. You must indicate any external sources you have used in the preparation of your solution. Do not plagiarize online sources and in general make sure you do not violate any of the academic standards of the course, the department or the university (the standards are available through the course's website: `http://www.pracsyslab.org/cs440`). Failure to follow these rules may result in failure in the course.

Question 1: Optical character recognition (OCR) is the task of extracting text from image sources. This is a very commercially useful technology, similar to the technique used by the US post office to route mail by zip codes. In this question, you will design three classifiers, which could be potentially used to implement this technology: a perceptron classifier, a multilayer Neural Network and an SVM classifier. You will test your classifiers on a data set consisting of scanned handwritten images of individual digits (0-9). Even with simple features, your classifiers may be able to do quite well when given enough training data.



Which Digit?

Figure 1: An example of data point in the data set.

Instructions:

1. Download the software framework from the url: (The commands are explained in `commands.txt`)
   `https://drive.google.com/drive/folders/0B6tuOE-7NjZAOHJSVzJZblZqclU?usp=sharing`

2. Implement three classification algorithms for digits identification:

   - **Perceptron**: A skeleton implementation of a perceptron classifier is provided in the file `perceptron.py`. You will fill in the `train` function and the `findHighWeightFeatures` function. The algorithm keeps a set of weight vector w^y for each class y (y is an identifier, not an exponent). Given the input vector x, the perceptron computes the class y whose weight vector is most similar to the input vector x. Formally, given an input vector x (in our case, a map from pixel locations to indicators of whether they are on), we score each class with:

     $$\text{score}(x, y) = \sum_i x_i w_i^y$$

     In the basic multi-class perceptron, we scan over the data, one instance at a time. When we come to an instance (x, y), we find the label with highest score:

     $$y' = \arg\max_{y''} score(f, y'')$$

     We compare y' to the true label y. If $y' = y$, we've gotten the instance correct, and we do nothing. Otherwise, we guessed y' but we should have guessed y. That means that w^y should have scored f higher, and $w^{y'}$ should have scored f lower, in order to prevent this error in the future. We update these two weight vectors accordingly:

     $$w^y += x$$

     $$w^{y'} -= x$$

     Using the addition, subtraction, and multiplication functionality of the Counter class in util.py, the perceptron updates should be relatively easy to code. Certain implementation issues have been taken care of for you in perceptron.py, such as handling iterations over the training data and ordering the update trials. Furthermore, the code sets up the weights data structure for you. Each legal label needs its own Counter full of weights.

   - **Multi-layer Neural Network**: In cases where the data is not linearly separable, it is advantageous to use a hidden layer of non-linear activations (e.g sigmoids) in between the input and the output layers of the perceptron. You will implement a Multi-layer perceptron (also referred to as "vanilla" neural networks) which uses the `back-propagation` algorithm to update the weights of the nodes in the network. Thus, you need to implement the `train` and `classify` methods in the file `mlp.py`. For this purpose you would need to implement:

– a forward pass over the desired network,
– a method to back-propagate error gradient based on the predicted output and the known training data,
– and a method to update the weights based on a gradient descent technique.

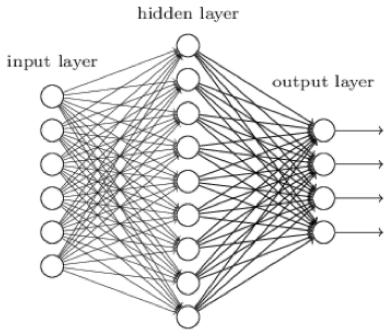


Figure 2: Multi-layer perceptron with a single hidden layer

- **Support Vector Machine**: SVM is yet another technique to perform linear binary classification. It operates by computing the hyperplane that maximizes the minimum distance between points belonging to opposite classes. This is represented mathematically as an optimization problem of the form:
  Minimize $\|\vec{w}\|$ subject to $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$, for $i = 1, \ldots, n$

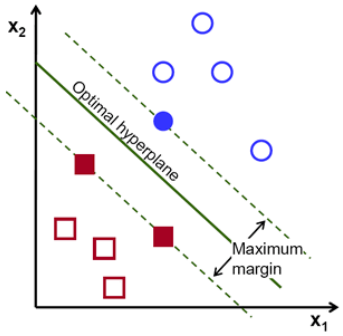


Figure 3: SVM classifier

where, w is the normal vector to the hyperplane and (x_i, y_i) is the training data. The solution can be computed using techniques like quadratic programming or gradient descent. In this task you could use a pre-built library like `scikit-learn` in python which has the LinearSVC class. You should use the (one vs the rest) classifier for each class in the dataset to extend the binary SVC classifier to multi-class predictor.

3. Train the three algorithms on the part of the data set that is reserved for training. First, use only 10% of the data points that are reserved for training, then 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and finally 100%. At each level of training, perform the prediction on the complete training set and report the prediction error for each algorithm. Thus, the x-axis for your plot would be the percentage of training data used for the training, and the y-axis would be the prediction error on the complete training set. You will need to generate three plots (or three clearly distinct lines on the same plot), each corresponding on a separate algorithm.

4. Now, perform the prediction on the part of the data set that is reserved for testing. For each of the three algorithms, plot the prediction error on the complete set of test data as a function of the percentage of data used for training (as above). i.e., the x-axis for your plot would be the percentage of training data used for the training, and the y-axis would be the prediction error on the complete testing set.

5. Provide a qualitative evaluation of the 3 algorithms by discussing their relative performance, their behavior for increasing training size, failure conditions and any fine-tuning you performed in order to get better results. What do you think you would do to improve the results even further given additional time and computational resources?

6. (Extra Credit) Use the validation set to tune the hyper-parameters of the algorithms. Report the performance improvement when you use the validation set.

(50 points + 5 extra credit points for appropriately utilizing the validation set)

Question 2: Consider the criteria for accepting graduate students at the hypothetical Univ. of Excellence. Each candidate is evaluated according to four attributes:

1. the grade point average (GPA)
2. the quality of the undergraduate degree
3. the publication record
4. the strength of the recommendation letters

To simplify our example, lets limit the possible values of each attribute: Possible GPA scores are ≥ 3.9, $3.9 > GPA > 3.2$, and ≤ 3.2; universities are categorized as "Rank 1", "Rank 2", and "Rank 3"; Prior research is a binary attribute - either the applicant has performed prior research or not. Recommendation letters are similarly binary, they are either good or normal. Finally, the candidates are classified into two classes: accepted, or P (for 'positive') and rejected, or N (for 'negative'). Figure 4 provides an example of one possible decision tree determining acceptance.

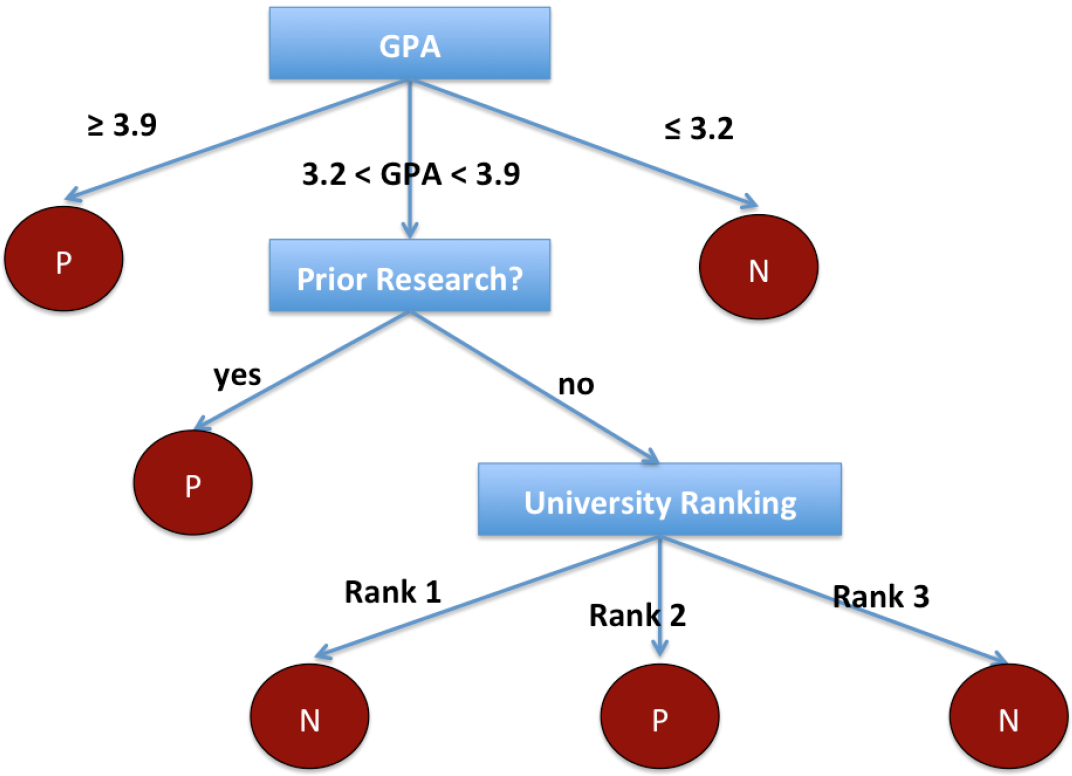


Figure 4: The decision tree that University of Excellence is using to determine acceptance.

An applicant doesn't know this decision tree, but does have the data regarding twelve of last year's applicants as in Table 1.

a) Does the provided tree correctly categorize the provided examples?

b) The applicant uses the decision tree algorithm shown in class (with the information gain computations for selecting split variables) to induce the decision tree employed by U. of E. officials. What tree will the algorithm come up with?

No.	GPA	University	Published	Recommendation	Class
1	4.0	rank 1	yes	good	P
2	3.9	rank 1	no	good	P
3	3.9	rank 2	no	normal	P
4	3.8	rank 1	yes	good	P
5	3.6	rank 2	no	good	P
6	3.6	rank 3	yes	good	P
7	3.4	rank 3	no	good	N
8	3.4	rank 1	no	good	N
9	3.2	rank 2	yes	normal	N
10	3.1	rank 1	no	normal	N
11	3.1	rank 3	yes	normal	N
12	3.0	rank 3	no	good	N

Table 1: The available knowledge for the applicant.

Show the computations involved, in addition to the decision tree itself.
[Hint: The information content of the examples before choosing any split variable is:

$$I(\frac{6}{12}, \frac{6}{12}) = -\frac{6}{12}log_2\frac{6}{12} - \frac{6}{12}log_2\frac{6}{12} = 1 \text{ bit of information}$$

You have to find the attribute that has the highest information gain:

$$Gain(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \sum_{i=1}^{n} \frac{p_i + n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

where attribute A divides the examples into i subsets, and p_i and n_i represent the number of positive and negative examples in subset i.]

c) Is the tree that you got in part b) equivalent to the tree provided here (i.e., do the two trees classify every application in the same way)? If the answer is yes, explain whether this is a coincidence or not. If the answer is no, give an example of a data case that will be classified differently by the two trees.

(10 points)

Question 3: Consider building an SVM for the following two-class training data:
Positive class:

$$[-1\ 3]^T, [0\ 2]^T, [0\ 1]^T, [0\ 0]^T$$

Negative class:

$$[1\ 5]^T, [1\ 6]^T, [3\ 3]^T$$

(a) Plot the training points and, by inspection, draw a linear classifier that separates the data with maximum margin.

(b) This linear SVM is parameterized by $h(x) = w^t x + b$. Provide the parameters w and b.

(c) Suppose you observe an additional set of points, all from the positive class:

More positive points:

$$[-2\ 0]^T, [-2\ 1]^T, [-2\ 3]^T, [-1\ 0]^T, [-1\ 1]^T, [0\ 0]^T$$

What is the linear SVM (in terms of w and b) now?

(15 points)

Question 4: Consider the chart below:

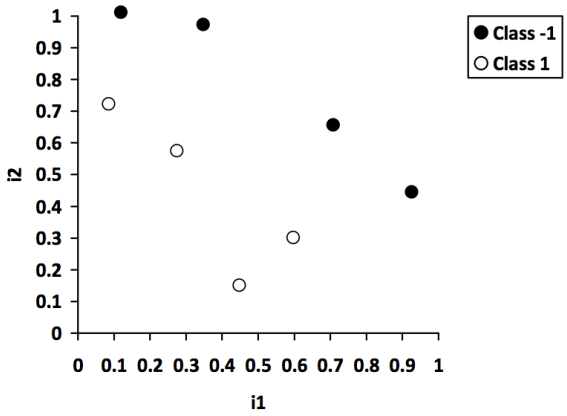


Figure 5: A set of two-dimensional input samples from two classes.

(a) Notice that for the above graph there is a perfect linear separator for the two input classes. Therefore, a single perceptron should be able to learn this classification task perfectly. Your task is to replicate the learning process, starting with a random perceptron with weights $w_0 = 0.2$, $w_1 = 1$, and $w_2 = -1$, where the weight w_0 corresponds to the constant offset $i_0 = 1$. For the inputs, just estimate their coordinates from the chart.

Start by adding the perceptron's initial line of separation to the chart. Compute then how many samples are misclassified? Then, select an arbitrary misclassified sample and describe the computation of the weight update (you can choose $\eta = 1$ or any other value; if you like you can experiment a bit to find a value that leads to efficient learning).

Illustrate the perceptron's new line of division in the same chart or a different one, and give the number of misclassified samples. Repeat this process four more times so that you have a total of six lines (or fewer if your perceptron achieves perfect classification earlier). Have in mind that the classification may not be perfect after these four steps.

You can generate the computations and/or graphs either by hand or by writing a simple computer program. If you write a program, please attach a printout, and let the program run for a larger number of steps.

(b) If your perceptron did not reach perfect classification, determine a set of weights that would achieve perfect classification, and draw the separating line for those weights.

(c) Now assume that less information were available about the samples. For instance, consider we only know the value for i_1 for each sample, which means that our perceptron has only two weights to classify the input as best as possible, i.e., it has weights w_0 and w_1, where w_0 is once again the weight for the constant offset $i_0 = 1$. Draw a diagram that visualizes this one-dimensional classification task, and determine weights for a perceptron that does the task as best as possible (minimum error, i.e., minimum proportion of misclassified samples). Where does it separate the input space, and what is its error?

(15 points)

Question 5: Consider the more difficult classification problem shown in the chart below:

(a) As you certainly noticed, a single perceptron cannot do this classification task perfectly. Determine the minimum error that a single perceptron can reach, and show the dividing line in the input space for such a perceptron.

(b) Clearly, we need a multi-layer perceptron to do this task perfectly. Develop such a system of connected perceptrons and write it down, together with the required weights for each unit. Illustrate the dividing lines for these perceptrons in a copy of the chart above.

(10 points)

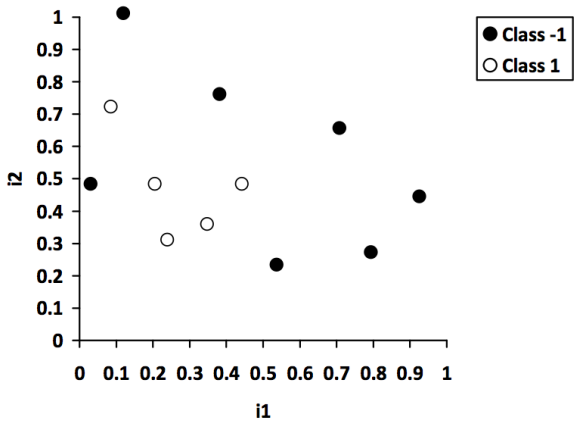


Figure 6: A set of two-dimensional input samples from two classes.

Extra Credit Problem:

Supervised learning methods are often used for the purpose of computing function approximators (i.e., a hypothesis function $h(x)$ approximating an underlying ground truth function f) using a set of training examples of the form $(x, f(x))$.

Consider the puzzles you work with in Assignment 1. One of the tasks which you performed was to compute a puzzle evaluation function (task 2), which expressed whether a puzzle can be solved or not and how difficult a puzzle is. For large puzzles, this evaluation based on breadth-first search (BFS) could take quite some time to compute.

For this extra credit assignment you are asked to design a potentially multi-layer Neural Network, which will be trained with puzzles of a fixed size (e.g., 50x50) as x and their corresponding evaluation values $f(x)$ using Task 2 from the previous assignment. The objective for the neural network is to approximate the value of a large puzzle as close as possible and to do so faster than the BFS. You may want to include both random puzzles as examples as well as optimized ones given one of the methods for stochastic optimization from the previous assignment.

For the training, generate first a training set of puzzles and compute their corresponding evaluation function with BFS. You should also generate a separate set of puzzles that will constitute the test set and will be used to evaluate your learning performance. For additional credits, you can follow a cross-validation approach to evaluate your model and report your performance.

You need to report the performance of the trained model on the test set you create and the amount of training data required to achieve the results. You should also report how much time takes on average for the BFS approach to compute the exact evaluation versus the neural network.

We will also create our own set of puzzles and will evaluate your trained model on that set.

(20 to 30 points)