CS 5350/6350: Machine Learining Spring 2019

Homework 2

Handed out: 11 Feb, 2019
Due: 11:59pm, 25 Feb, 2019

1 Paper Problems [40 points + 6 bonus]

1. [5 points] We have derived the PAC guarantee for consistent learners (namely, the learners can produce a hypothesis that can 100% accurately classify the training data). The PAC guarantee is described as follows. Let H be the hypothesis space used by our algorithm. Let C be the concept class we want to apply our learning algorithm to search for a target function in C. We have shown that, with probability at least $1-\delta$, a hypothesis $h \in H$ that is consistent with a training set of m examples will have the generalization error $\mathrm{err}_D(h) < \epsilon$ if

$$m > \frac{1}{\epsilon}\big(\log(|H|) + \log\frac{1}{\delta}\big)$$

   (a) [2 points] Suppose we have two learning algorithms L_1 and L_2, which use hypothesis spaces H_1 and H_2 respectively. We know that H_1 is larger than H_2, i.e., $|H_1| > |H_2|$. For each target function in C, we assume both algorithms can find a hypothesis consistent with the training data.

      i. [1 point] According to Occam's Razor principle, which learning algorithm's result hypothesis do you prefer? Why?

Solution

An interpretation Occam's Razor principle states that given two explanations of the data, the simpler explanation is preferable. According to this, I would prefer hypothesis with the smaller hypothesis space size, H_2. This is in line with the goal of Machine learning, to find the simplest hypothesis that is consistent with the sampled data.

      ii. [1 point] How is this principle reflected in our PAC guarantee? Please use the above inequality to explain why we will prefer the corresponding result hypothesis.

Solution

If we assume fixed error and confidence parameters (δ and ϵ) for both learning algorithms, algorithm L_1 will require more samples (m_1) in order to produce the same error and confidence levels. This can be seen in the following inequalities:

$$m_1 > \frac{1}{\epsilon}\big(\log(|H_1|) + \log\frac{1}{\delta}\big)$$

$$m_2 > \frac{1}{\epsilon}\big(\log(|H_2|) + \log\frac{1}{\delta}\big)$$

Using the inequality $|H_1| > |H_2|$, and since δ and ϵ are constant:

$$\frac{1}{\epsilon}\big(\log(|H_1|) + \log\frac{1}{\delta}\big) > \frac{1}{\epsilon}\big(\log(|H_2|) + \log\frac{1}{\delta}\big)$$
$$m_1 > m_2$$

This means that if both algorithms use the same number of training samples during the learning phase, algorithm L_2 will have higher accuracy $(1-\epsilon)$ and confidence $(1-\delta)$. I therefore prefer the hypothesis with the smaller hypothesis space, H_2.

(b) [3 points] Let us investigate algorithm L_1. Suppose we have n input features, and the size of the hypothesis space used by L_1 is 3^n. Given $n = 10$ features, if we want to guarantee a 95% chance of learning a hypothesis of at least 90% generalization accuracy, how many training examples at least do we need for L_1?

2. [7 points] Let us check the PAC learnability for several concept classes. We assume that our learning algorithm can always result in a hypothesis consistent with the training data. We assume the hypothesis space H is the same as the concept class C. Please determine whether the following concept classes are PAC learnable or not and prove your conclusions [Hint: please carefully check the definition of PAC learnability].

   (a) [1 points] Simple disjunctions out of n binary variables

   (b) [1 points] m-of-n rules (Note that m is a fixed constant).

   (c) [1 points] General conjunctions out of n binary variables ("General" means that negations are allowed to operator on each variable before the conjunction).

   (d) [2 points] General Boolean functions of n binary variables.

   (e) [2 points] Can ID3 algorithm *efficiently* PAC learn the above concept classes?

3. [5 points] In our lecture about AdaBoost algorithm, we introduced the definition of weighted error in each round t,

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\big(\sum_{i=1}^{m} D_t(i) y_i h_t(x_i)\big)$$

where $D_t(i)$ is the weight of i-th training example, and $h_t(x_i)$ is the prediction of the weak classifier learned round t. Note that both y_i and $h_t(x_i)$ belong to $\{1, -1\}$. Prove that equivalently,

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

Starting with the following definition of Z_t, I then take the derivative w.r.t. α_t and set equal to zero:

$$Z_t = \sum_{i:h_t(x_i)=y_i} D_t(i)e^{-\alpha_t} + \sum_{i:h_t(x_i)\neq y_i} D_t(i)e^{\alpha_t}$$

$$\frac{dZ_t}{d\alpha_t} = \sum_{i:h_t(x_i)=y_i} -D_t(i)e^{-\alpha_t} + \sum_{i:h_t(x_i)\neq y_i} D_t(i)e^{\alpha_t} = 0$$

$$\sum_{i:h_t(x_i)=y_i} D_t(i) = \sum_{i:h_t(x_i)\neq y_i} D_t(i)e^{2\alpha_t}$$

Then, by definition and the proof proof presented in class, the error can be written as:

$$\epsilon_t = \sum_{y_i \neq h_t(x_i)} D_t(i).$$

4. [8 points]

   (a) [1 point] $f(x_1, x_2, x_3) = x_1 \vee x_2 \vee x_3$
   Starting with the generic hyperplane formula for three dimensions, we can solve for the bias and weights to fit the above boolean disjunction.

   $$b + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

   where $\mathbf{w} = [1, 1, 1]$ and $b = -1$. Putting this together gives the following linear classifier:

   $$x_1 + x_2 + x_3 \geq 1$$

   This means that $f(x_1, x_2, x_3) = 1$ iff the above inequality is true. The separating hyperplane is therefore:

   $$x_1 + x_2 + x_3 - 1 = 0$$

(b) [1 point] $f(x_1, x_2, x_3) = x_1 \wedge \neg x_2 \wedge \neg x_3$

Just as the problem above:

$$b + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

where $\mathbf{w} = [1, -1, -1]$ and $b = -1$. Putting this together gives the following linear classifier:

$$x_1 - x_2 - x_3 \geq 1$$

The separating hyperplane is therefore:

$$x_1 - x_2 - x_3 - 1 = 0$$

(c) [1 point] $f(x_1, x_2, x_3) = \neg x_1 \vee \neg x_2 \vee \neg x_3$

Using De Morgan's law, the negation of a conjuntion is the disjuntion of the negations. We can therefore rewrite the above equation as:

$$\begin{aligned} f(x_1, x_2, x_3) &= \neg x_1 \vee \neg x_2 \vee \neg x_3 \\ &= \neg(x_1 \wedge x_2 \wedge x_3) \end{aligned}$$

Solving for the weights and bias gives the following:

$$b + w_1x_1 + w_2x_2 + w_3x_3 = 0$$

where $\mathbf{w} = [1, 1, 1]$ and $b = -3$. Putting this together gives the following linear classifier:

$$x_1 + x_2 + x_3 < 3$$

This means that $f(x_1, x_2, x_3) = 1$ iff the above inequality is true. The separating hyperplane is therefore:

$$x_1 + x_2 + x_3 - 3 = 0$$

Comparing $f(x_1, x_2, x_3) = \neg(x_1 \wedge x_2 \wedge x_3)$ and $f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3)$, the only difference is in this case we use a $<$ instead of $\geq$.

(d) [2 points] $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$

(e) [2 points] $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$

(f) [1 point] What do you conclude about the expressiveness of decision trees and linear classifiers? Why?

5. [6 points] The following boolean functions cannot be represented by linear classifiers. Can you work out some feature mapping such that, after mapping all the inputs of these functions into a higher dimensional space, you can easily identify a hyperplane that separates the inputs with different corresponding boolean function values? Please write down the separating hyperplane as well.

(a) $f(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

(b) $f(x_1, x_2) = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

(c) $f(x_1, x_2, x_3)$ is listed in the following table

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	1
1	0	0	1
0	1	1	0
1	1	0	0
1	0	1	0
1	1	1	1

6. [**Bonus**] [6 points] Given two vectors $\mathbf{x} = [x_1, x_2]$ and $\mathbf{y} = [y_1, y_2]$, find a feature mapping $\phi(\cdot)$ for each of the following functions, such that the function is equal to the inner product between the mapped feature vectors, $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$. For example, $(\mathbf{x}^\top \mathbf{y})^0 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = [1]$ and $\phi(\mathbf{y}) = [1]$; $(\mathbf{x}^\top \mathbf{y})^1 = \phi(\mathbf{x})^\top \phi(\mathbf{y})$ where $\phi(\mathbf{x}) = \mathbf{x}$ and $\phi(\mathbf{y}) = \mathbf{y}$.

(a) [2 points] $(\mathbf{x}^\top \mathbf{y})^2$

(b) [2 points] $(\mathbf{x}^\top \mathbf{y})^3$

(c) [2 points] $(\mathbf{x}^\top \mathbf{y})^k$ where k is any positive integer.

7. [9 points] Suppose we have the training data shown in Table 1, from which we want to learn a linear regression model, parameterized by a weight vector $\mathbf{w}$ and a bias parameter b.

(a) [1 point] Write down the LMS (least mean square) cost function $J(\mathbf{w}, b)$.

(b) [3 points] Calculate the gradient $\frac{\nabla J}{\nabla \mathbf{w}}$ and $\frac{\nabla J}{\nabla b}$

i. when $\mathbf{w} = [0, 0, 0]^\top$ and $b = 0$;

x_1	x_2	x_3	y
1	-1	2	1
1	1	3	4
-1	1	0	-1
1	2	-4	-2
3	-1	-1	0

Table 1: Linear regression training data.

ii. when $\mathbf{w} = [-1, 1, -1]^\top$ and $b = -1$;

iii. when $\mathbf{w} = [1/2, -1/2, 1/2]^\top$ and $b = 1$.

(c) [2 points] What are the optimal $\mathbf{w}$ and $\mathbf{b}$ that minimize the cost function?

(d) [3 points] Now, we want to use stochastic gradient descent to minimize $J(\mathbf{w}, b)$. We initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$. We set the learning rate $r = 0.1$ and sequentially go through the 5 training examples. Please list the stochastic gradient in each step and the updated $\mathbf{w}$ and b.

2 Practice [60 points + 10 bonus]

1. [2 Points] Update your machine learning library. Please check in your implementation of decision trees in HW1 to your GitHub repository. Remember last time you created a folder "Decision Tree". You can commit your code into that folder. Please also supplement README.md with concise descriptions about how to use your code to learn decision trees (how to call the command, set the parameters, etc). Please create two folders "Ensemble Learning" and "Linear Regression" in the same level as the folder "Decision Tree".

2. [36 points] We will implement the boosting and bagging algorithms based on decision trees. Let us test them on the bank marketing dataset in HW1 (bank.zip in Canvas). We use the same approach to convert the numerical features into binary ones. That is, we choose the media (NOT the average) of the attribute values (in the training set) as the threshold, and examine if the feature is bigger (or less) than the threshold. For simplicity, we treat "unknown" as a particular attribute value, and hence we do not have any missing attributes for both training and test.

   (a) [8 points] Modify your decision tree learning algorithm to learn decision stumps — trees with only two levels. Specifically, compute the information gain to select the best feature to split the data. Then for each subset, create a leaf node. Note that your decision stumps must support weighted training examples. Based on your decision stump learning algorithm, implement AdaBoost algorithm. Vary the number of iterations T from 1 to 1000, and examine the training and test errors. You should report the results in two figures. The first figure shows how the training and test errors vary along with T. The second figure shows the training and test errors of all the decision stumps learned in each iteration. What can you observe and conclude? You have had the results for a fully expanded decision tree in HW1. Comparing them with Adaboost, what can you observe and conclude?

(b) [8 points] Based on your code of the decision tree learning algorithm (with information gain), implement a Bagged trees learning algorithm. Note that each tree should be fully expanded — no early stopping or post pruning. Vary the number of trees from 1 to 1000, report how the training and test errors vary along with the tree number in a figure. Overall, are bagged trees better than a single tree? Are bagged trees better than Adaboost?

(c) [6 points] Through the bias and variance decomposition, we have justified why the bagging approach is more effective than a single classifier/predictor. Let us verify it in real data. Experiment with the following procedure.

- REPEAT for 100 times
- [STEP 1] Sample $1,000$ examples *uniformly without replacement* from the training datset
- [STEP 2] Run your bagged trees learning algorithm based on the $1,000$ training examples and learn 1000 trees.
- END REPEAT
- Now you have 100 bagged predictors in hand. For comparison, pick the first tree in each run to get 100 fully expanded trees (i.e. single trees).
- For each of the test example, compute the predictions of the 100 single trees. Take the average, subtract the ground-truth label, and take square to compute the bias term (see the lecture slides). Use all the predictions to compute the sample variance as the approximation to the variance term (if you forget what the sample variance is, check it out here). You now obtain the bias and variance terms of a single tree learner for one test example. You will need to compute them for all the test examples and then take average as your final estimation of the bias and variance terms for the single decision tree learner. You can add the two terms to obtain the estimation of the general squared error (that is, expected error w.r.t test examples). Now use your 100 bagged predictors to do the same thing and estimate the general bias and variance terms, as well as the general squared error. Comparing the results of the single tree learner and the bagged trees, what can you conclude? What causes the difference?

(d) [8 points] Implement the random forest algorithm as we discussed in our lecture. Vary the number of random trees from 1 to 1000. Note that you need to modify your tree learning algorithm to randomly select a subset of features before each split. Then use the information gain to select the best feature to split. Vary the size of the feature subset from $\{2, 4, 6\}$. Note that if your feature subset happen to be all the features used before, simply resample the subset. Report in a figure how the training and test errors vary along with the number of random trees for each feature subset size setting. How does the performance compare with bagged trees?

(e) [6 points] Following (c), estimate the bias and variance terms, and the squared error for a single random tree and the whole forest. Comparing with the bagged trees, what do you observe? What can you conclude?

3. [**Bonus**][10 points] In practice, to confirm the performance of your algorithm, you need to find multiple datasets for test (rather than one). You need to extract and process data by yourself. Now please use the credit default dataset in UCI repository https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Randomly choose 24000 examples for training and the remaining 6000 for test. Feel free to deal with continuous features. Run bagged trees, random forest, and Adaboost with decision stumps algorithms for 1000 iterations. Report in a figure how the training and test errors vary along with the number of iterations, as compared with a fully expanded single decision tree. Are the results consistent with the results you obtained from the bank dataset?

4. [22 points] We will implement the LMS method for a linear regression task. The dataset is from UCI repository (`https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test`). The task is to predict the real-valued SLUMP of the concrete, with 7 features. The features and output are listed in the file "concrete/data-desc.txt". The training data are stored in the file "concrete/train.csv", consisting of 53 examples. The test data are stored in "concrete/test.csv", and comprise of 50 examples. In both the training and testing datasets, feature values and outputs are separated by commas.

   (a) [8 points] Implement the batch gradient descent algorithm, and tune the learning rate r to ensure the algorithm converges. To examine convergence, you can watch the norm of the weight vector difference, $\|w_t - w_{t-1}\|$, at each step t. if $\|w_t - w_{t-1}\|$ is less than a tolerance level, say, $1e-6$, you can conclude that it converges. You can initialize your weight vector to be $\mathbf{0}$. Please find an appropriate r such that the algorithm converges. To tune r, you can start with a relatively big value, say, $r = 1$, and then gradually decrease r, say $r = 0.5, 0.25, 0.125, \ldots$, until you see the convergence. Report the learned weight vector, and the learning rate r. Meanwhile, please record the cost function value of the training data at each step, and then draw a figure shows how the cost function changes along with steps. Use your final weight vector to calculate the cost function value of the test data.

   (b) [8 points] Implement the stochastic gradient descent (SGD) algorithm. You can initialize your weight vector to be $\mathbf{0}$. Each step, you randomly sample a training example, and then calculate the stochastic gradient to update the weight vector. Tune the learning rate r to ensure your SGD converges. To check convergence, you can calculate the cost function of the training data after each stochastic gradient update, and draw a figure showing how the cost function values vary along with the number of updates. At the beginning, your curve will oscillate a lot. However, with an appropriate r, as more and more updates are finished, you will see the cost function tends to converge. Please report the learned weight vector, and the learning rate you chose, and the cost function value of the test data with your learned weight vector.

(c) [6 points] We have discussed how to calculate the optimal weight vector with an analytical form. Please calculate the optimal weight vector in this way. Comparing with the weight vectors learned by batch gradient descent and stochastic gradient descent, what can you conclude? Why?