TTIC 31020: Introduction to Machine Learning
Autumn 2019

Problem Set #4 (90 Points)

Out: November 8, 2019

Due: Wednesday November 20, 11:59pm

Instructions

How and what to submit? Please submit your solutions electronically via Canvas. Please submit two files:

1. A PDF file with the written component of your solution including derivations, explanations, etc. You can create this PDF in any way you want, e.g.,LaTeX (recommended), Word + export as PDF, scan handwritten solutions (note: must be legible!), etc. Please name this document ⟨`firstname-lastname`⟩`-sol4.pdf`.

2. The empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures) in a Jupyter notebook file. Rename the notebook ⟨`firstname-lastname`⟩`-sol4.ipynb`.

Late submissions: there will be a penalty of 20 points for any solution submitted within 24 hours past the deadline. No submissions will be accepted past then.

What is the required level of detail? When asked to derive something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations. When asked to plot something, please include in the `ipynb` file the figure as well as the code used to plot it. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers) and references in a legend or in a caption. When asked to provide a brief explanation or description, try to make your answers concise, but do not omit anything you believe is important. If there is a mathematical answer, provide it precisely (and accompany by succinct wording, if appropriate).

When submitting code (in Jupyter notebook), please make sure it's reasonably well-documented, runs, and produces all the requested results. If discussion is required/warranted, you can include it either directly in the notebook (you may want to use markdown style for that) or in the PDF writeup.

Collaboration policy: collaboration is allowed and encouraged, as long as you (1) write your own solution entirely on your own, and (2) specify names of student(s) you collaborated with in your writeup.

1 Boosting

In stepwise fit-forward (least squares) regression, in each iteration a simple regressor is fit to the residuals obtained by the ensemble model up to that iteration. As a result, it is easy to see that *after* this regressor is added, the new residuals are uncorrelated with its predictions, due to a general property of least squares regression.

We will now investigate a similar phenomenon that occurs with weak classifiers in AdaBoost. Here, we assume that the weights are normalized after each update, so that

$$\sum_i W_i^{(m)} = 1$$

for each boosting round m.

Problem 1 **[14 points]**
Consider an ensemble classifier $H_M(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$ constructed by M rounds of AdaBoost on n training examples. Now we will add the next classifier h_{M+1} to the ensemble. To do so, we first learn h_{M+1} by minimizing the training error with examples weighted by $W_1^{(M)}, \ldots, W_n^{(M)}$. Then, we compute the weighted training error ϵ_{M+1} for h_{M+1} as follows:

$$\epsilon_{M+1} = \sum_{i:y_i \neq h_{M+1}(\mathbf{x}_i)} W_i^{(M)}$$

Given ϵ_{M+1}, we then set α_{M+1} using the closed-form expression from the lecture slides (also shown below in Problem 2). Then we compute the weights $W_1^{(M+1)}, \ldots, W_n^{(M+1)}$ as in the lecture slides, ensuring that they are normalized over training examples.

(a) Show that the training error (number of misclassified training examples) of h_{M+1} weighted by the *updated* weights $W_1^{(M+1)}, \ldots, W_n^{(M+1)}$, is exactly $1/2$. Note the differences between "training error" (the number of misclassified training examples), "weighted training error" ϵ_{M+1} (training error weighted by the weights $W_1^{(M)}, \ldots, W_n^{(M)}$), and "weighted training error with updated weights" (like weighted training error except using updated weights $W_1^{(M+1)}, \ldots, W_n^{(M+1)}$). This question is asking you to show that the weighted training error with updated weights is exactly $1/2$.

(b) With the above fact in mind, could we select the same classifier again in the immediately following round, i.e., can we have $h_m = h_{m+1}$ for some m? Explain why or why not.

(c) Can we have $h_{m+k} = h_m$ for some $k > 1$? Explain why or why not.

End of problem 1

Problem 2 **[10 points]**
Recall the expression of the vote strength α_m for a weak classifier h_m in AdaBoost,

$$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m} \tag{1}$$

where ϵ_m is the weighted training error as defined in the lecture slides and described above in Problem 1.

Show that (1) minimizes the empirical exponential loss (i.e., exponential loss on the training data) given the selection of h_m.

End of problem 2

2 Kernels

Here we will look at an example of constructing feature spaces for classification problems. We will explore the idea of the "kernel trick" applied to classifiers other than SVMs (i.e., classifiers using loss functions other than hinge loss). Specifically, we will introduce kernel logistic regression (KLR). Recall that in class we described the general logistic regression model as (omitting the bias weight):

$$\hat{p}\left(y = 1 \,|\, \mathbf{x};\, \mathbf{w}\right) \;=\; \frac{1}{1 + \exp\left(\sum_{j=1}^{d} w_j \phi_j(\mathbf{x})\right)}$$

where $\phi_j(\mathbf{x})$ is the j-th *basis function*, or *feature* - generally, a function mapping $\mathcal{X} \to \mathbb{R}$.

Note (11/14/2019): The above formulation of logistic regression differs from the form we saw in class, which had a negative sign inside the exp(). **Sorry for the confusion. You can answer this question either using the form shown above or the form shown in class.**

Problem 3 **[16 points]**

Show how by appropriate choice of basis functions $\boldsymbol{\phi}$, given a training set $\mathbf{x}_1, \ldots, \mathbf{x}_n$, one can obtain a logistic regression model whose predictions on a test point $\mathbf{x}_0$ depend on the training data only through the kernel values $K(\mathbf{x}_i, \mathbf{x}_0)$ for $i = 1, \ldots, n$. That is, the predictions should not depend on the training data in any way other than through those kernel values. (Note, however, that the predictions can still depend on $\mathbf{w}$. That is, unlike how we introduced kernels into SVMs, you do not need to rewrite $\mathbf{w}$ in terms of kernel values.) Then write down the gradient of the log loss with L_2 regularization for this logistic regression model, on a single example, and show that the training for this model via gradient descent also depends on the training data only through kernel computations.

End of problem 3

3 Decision Trees for Spam Detection

In this section we will apply decision trees to the problem of determining whether or not an email is spam. The dataset has been split into a training set of 3000 examples, along with validation and test sets.

Each example has 57 continuous valued features, mostly indicating the frequencies of words such as "money", "free", and "credit". Other features include the length of the longest run of capital letters, and some character frequencies (e.g., "$" and "!"). The labels are binary, and the annotation is slightly noisy (there are two pairs of training examples with exactly the same features, but different labels).

One difference in our implementation from that discussed in class is that we do not explicitly include a complexity penalty hyperparameter. Recall the cost-complexity criterion given below:

$$C_\lambda(T) = \lambda|T| + \sum_{m=1}^{|T|} N_m Q_m(T)$$

where $|T|$ is the number of leaves of T, N_m is the number of training examples classified by leaf m, and $Q_m(T)$ is some measure of impurity (in our case the Gini index). It was stated in lecture that the subtree of T that minimizes C_λ will be contained in the sequence of trees produced by repeatedly collapsing the internal node which increases the training error by the least. Rather than treating λ as a hyperparameter, we simply take the tree from this sequence that minimizes the validation error.

Problem 4 [15 points]

Fill in the missing pieces of the functions needed to grow and prune our decision trees. Specifically, the functions `gini_index`, `compute_impurity`, and `compute_error_change`.

These functions deal with weights rather than counts of examples because we will be using our decision trees later for weighted data. However, in the unweighted case the weights are just counts (e.g. in the `gini_index` function, `total_weight` is simply the number of examples in the leaf).

End of problem 4

Problem 5 [10 points]

Run your decision tree code on the data we have provided, for a variety of hyperparameter settings (the `max_depth` and `min_split_size` arguments).

We have provided a grid search over some reasonable parameter settings, but you are welcome to explore more. Once you are satisfied with your validation accuracy, report it in your write-up, and run the `make_submission` function. This will generate a file called `submission_single.csv`, which you should upload to Kaggle at:

`https://www.kaggle.com/c/2019-ttic-31020-hw4-spam-single-tree`

End of problem 5

4 AdaBoost with Decision Trees

In this section we will use the decision trees we developed in the previous section as weak learners, and combine them with AdaBoost. Due to the fact that our pruning code uses

the validation set for computational efficiency, we will not prune our weak learners (we will instead just use low depth to limit complexity).

Problem 6 **[15 points]**

Fill in the missing code in the `adaboost` function. Your code should compute the weighted error of the predictions in the `predictions` array, compute the α weight for the new weak learner, and update the weights for each training example.

End of problem 6

Problem 7 **[10 points]**

Fit an ensemble to the training data, and report your validation accuracy. Feel free to improve your performance by modifying the hyperparameters. Run the `make_submission` function to create the file `submission_ensemble.csv`, and upload it to Kaggle at:

`https://www.kaggle.com/c/2019-ttic-31020-hw4-spam-adaboost`

End of problem 7