TTIC 31020: Introduction to Machine Learning
Autumn 2020

Problem Set #4

Out: November 12, 2018

Due: Monday November 23, 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 25 points for any solution submitted within 24 hours past the deadlne. 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 is precisely (and accompany by only succinct words, if appropriate).

When submitting code (in Jupyter notebook), please make sure it's reasonably 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 the 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, (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

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

for each boosting round t.

Problem 1 **[10 points]**
Consider an ensemble classifier $H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$ constructed by T rounds of AdaBoost on N training examples. Now we add next classifier h_{T+1} to the ensemble, by minimizing the training error weighted by $W_1^{(T)}, \ldots, W_N^{(T)}$, compute α_{T+1}, and update the weights. Show that the training error of the just added h_{T+1} (note: not the error of H_{T+1}) weighted by the *updated* weights $W_1^{(T+1)}, \ldots, W_N^{(T+1)}$, is exactly 1/2.

Now, with that fact in mind, is it possible that AdaBoost would select the same classifier again in the immediately following round, i.e., can we have $h_t = h_{t+1}$ for some t? Can we have $h_{t+k} = h_t$ for some $k > 1$? Explain why or why not.

End of problem 1

Problem 2 **[10 points]**
Recal the expression of the vote strength α_t for a weak classifier h_t in AdaBoost,

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}, \tag{1}$$

where ϵ_t is the weighted training error of that weak classifier under the current weights at the beginning of iteration t in which it is chosen.

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

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 SVM (i.e., classifiers using loss functions other than hinge loss). Specifically, we will introduce the

kernel logistic regression (KLR). Recall that in class we have described the general logistic regression model as

$$\hat{p}\,(y = 1 \,|\, \mathbf{x};\, \mathbf{w}, w_0) \;=\; \frac{1}{1 + \exp\left(\sum_{j=1}^{d} w_j \phi_j(\mathbf{x})\right)},$$

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

Problem 3 **[20 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 $i = 1, \ldots, N$. Then write down the gradient of the loss, with L_2 regularization, 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 Multivariate Gaussians

Since many (probably most) generative models for real-valued data involve multivariate Gaussian distribution, it is worth getting to know them better. In particular, here we will understand why one always sees those elliptical contours drawn when Gaussians are visualized in 2D.

Recall that the probability density function (pdf) of a Gaussian distribution in $\mathbb{R}^d$ is given by

$$p(\mathbf{x}) \;=\; \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right). \tag{2}$$

Problem 4 **[10 points]**
Show that a contour corresponding to a fixed value of pdf is an ellipse in the 2D space, $\mathbf{x} = [x_1, x_2]$

End of problem 4

Advice: You may find it easier to work in the log domain, and to write out explicitly the expression in (2) *in terms of* x_1 *and* x_2*.*

4 Decision trees and forests

In this hands-on portion of the problem set, we will experiment with decision trees on the familiar MNIST handwritten digit dataset. We have provided a notebook with (partial) implementation of a decision tree. The data files are exactly the same as you used in Problem set 2.

Problem 5 **[20 points]**
Complete the implementation of the decision tree growing. The machinery for evaluating splits and selecting the best split (using the Gini index) for node using a given single feature is provided in `best_split_gini`. You need to write code that uses this machinery to evaluate all possible splits for a node and select the best one (among all features) in `split_data` and the code to grow the tree in `grow_tree`.

Report training and validation set accuracies of a single tree, with hyperparameters (max tree depth and minimum size of a leaf) tuned as you see fit. Submit your predictions with this tree on the test set to Kaggle.

End of problem 5

Problem 6 **[10 points]**
BONUS: Add the code for pruning a tree, and report the accuracy of the CART tree (with pruning) using hyperparameters (including λ) tuned as you see fit. You can submit this result to Kaggle instead of the one with the unpruned tree, or in addition to that one.

End of problem 6

Now we will apply bagging to construct a "random forest" of trees. In class we went over the process: each tree is randomized by only allowing a node to consider a random subset of $M < d$ features (where d is the number of features of the input), and the training set for each tree consists of N examples sampled *with replacement* from the N training points.

Problem 7 **[20 points]**
Now complete the code in `create_ensemble` and use it to construct an ensemble of (unpruned) trees. Use at least ten trees; you are encouraged to try more. You will have to guess/tune a good value of M; we recommend values less than 20% of the number of features in the data. Note that higher M means more computation with constructing the trees.

Note that the code for computing the best split in `split_data` is supposed to already include the (simple) mechanism for randomizing the features considered to split each node. You need to handle the other source of stochasticity (giving each tree a "bag" of data points sampled with replacement out of the training set) in this function. *Hint: the code you add to* `create_ensemble` *should be only a couple of lines.*

Report the training and validation set accuracies of your ensemble. You should aim to get at least 90% validation accuracy.

Finally, submit the predictions from your ensemble on the test set to the second Kaggle competition.

End of problem 7

Advice: The implementation of `best_split_gini` *we have provided is fairly efficient, and most of the computation happens there. You should expect growing 10 trees to take 3-4 minutes or less.*

Problem 8 **[10 points]**

Show the effect of the size of your ensemble as follows: let T be the number of trees in your ensemble. For $t = 1, \ldots, T$ compute the validation set accuracy of the partial ensemble consisting of trees $1, \ldots, t$, and plot this as a function of t. Discuss what this tells you and how it may inform tuning T.

End of problem 8

Advice: Do not re-run the creation of the forest for $T = 1, 2, \ldots$! rather, take your existing ensemble and compute the accuracy of the partial ensembles consisting of only the first tree; the first two trees; etc.