Homework #3

RELEASE DATE: 04/29/2022

RED BUG FIX: 5/2/2022 18:00

BLUE BUG FIX: 5/8/2020 18:00

DUE DATE: 05/20/2022 23:59 on NTU COOL

QUESTIONS ABOUT HOMEWORK MATERIALS
ARE WELCOMED ON THE NTU COOL FORUM.

Unless granted by the instructor in advance, you must upload your solution to NTU COOL as instructed by the TA.

Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

You should write your solutions in English or Chinese with the common math notations introduced in class or in the problems. We do not accept solutions written in any other languages.

Hand-written Part

1. (10%) Consider a binary classification data set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ with $\mathbf{x}_n \in \mathbb{R}^d$ and $y_n \in \{-1, +1\}$. For any weight vector $\mathbf{w}$ within a linear model, define an error function

$$\text{err}(\mathbf{w}^T\mathbf{x}, y) = \big(\max(1 - y\mathbf{w}^T\mathbf{x}, 0)\big)^2.$$

That is,

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\big(\max(1 - y_n\mathbf{w}^T\mathbf{x}_n, 0)\big)^2$$

Running gradient descent to optimize $E_{\text{in}}(\mathbf{w})$ requires calculating its gradient direction $\nabla E_{\text{in}}(\mathbf{w})$ (and then move opposite to that direction). What is $\nabla E_{\text{in}}(\mathbf{w})$?

The error function err *is called the squared hinge error and is a core component in the so-called ℓ_2-loss SVM.*

2. (10%) Consider a data matrix $\mathrm{X} \in \mathbb{R}^{N\times d}$ and $\mathbf{y} \in \mathbb{R}^{N\times 1}$ as defined for linear regression. When $\mathrm{X}^T\mathrm{X}$ is invertible, we showed in class that the linear regression solution

$$\mathbf{w}_{\text{lin}} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathbf{y}.$$

If the singular value decomposition of $\mathrm{X} = \mathrm{U}\Sigma\mathrm{V}^T$ where $\mathrm{U} \in \mathbb{R}^{N\times N}$ and $\mathrm{V} \in \mathbb{R}^{d\times d}$ are real orthogonal matrices and $\Sigma \in \mathbb{R}^{N\times d}$ is a diagonal rectangular matrix. Prove that

$$\mathbf{w}_{\text{lin}} = \mathrm{V}\Gamma\mathrm{U}^T\mathbf{y},$$

where $\Gamma \in \mathbb{R}^{d\times N}$ is a diagonal rectangular matrix with $\Gamma[i,j] = \frac{1}{\Sigma[j,i]}$ if $\Sigma[j,i] > 0$ and $\Gamma[i,j] = 0$ otherwise.

Actually, $\mathrm{X}^\dagger$, the so-called (Moore-Penrose) pseudo inverse as shown in our slides, can be defined as $\mathrm{V}\Gamma\mathrm{U}^T$, regardless of whether $\mathrm{X}^T\mathrm{X}$ is invertible.

3. (10%) Consider a process that generates d-dimensional vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ independently from a multivariate Gaussian distribution $\mathcal{N}(\mathbf{u}, \mathrm{I})$, where $\mathbf{u} \in \mathbb{R}^d$ is an unknown parameter vector and $\mathrm{I} \in \mathbb{R}^{d \times d}$ is an identity matrix. The maximum likelihood estimate of $\mathbf{u}$ is

$$\mathbf{u}^* = \operatorname*{arg\,max}_{\mathbf{u} \in \mathbb{R}^d} \prod_{n=1}^{N} p_{\mathbf{u}}(\mathbf{x}_n),$$

where $p_{\mathbf{u}}$ is the probability density function of $\mathcal{N}(\mathbf{u}, \mathrm{I})$. Prove that

$$\mathbf{u}^* = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n.$$

The same derivation can be used to obtain the cross-entropy error function of logistic regression.

4. (10%) A classic binary classification data set that cannot be separated by any line is called the XOR data set, with

$\mathbf{x} = [x_1, x_2]$	y
$\mathbf{x}_1 = [+1, +1]$	$y_1 = -1$
$\mathbf{x}_2 = [-1, +1]$	$y_2 = +1$
$\mathbf{x}_3 = [-1, -1]$	$y_3 = -1$
$\mathbf{x}_3 = [+1, -1]$	$y_4 = +1$

You can see why it is called XOR by interpreting $+1$ as a boolean value of TRUE and -1 as FALSE. Consider a second-order feature transform $\Phi_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$ that converts the data set to

$\mathbf{z} = \Phi_2(\mathbf{x})$	y
$\mathbf{z}_1 = \Phi_2(\mathbf{x}_1)$	$y_1 = -1$
$\mathbf{z}_2 = \Phi_2(\mathbf{x}_2)$	$y_2 = +1$
$\mathbf{z}_3 = \Phi_2(\mathbf{x}_3)$	$y_3 = -1$
$\mathbf{z}_4 = \Phi_2(\mathbf{x}_4)$	$y_4 = +1$

Show a perceptron $\tilde{\mathbf{w}}$ in the $\mathcal{Z}$-space that separates the data. That is,

$$y_n = \operatorname{sign}(\tilde{\mathbf{w}}^T \mathbf{z}_n) \text{ for } n = 1, 2, 3, 4.$$

Then, plot the classification boundary

$$\tilde{\mathbf{w}}^T \Phi_2(\mathbf{x}) = 0$$

in the $\mathcal{X}$-space. Your boundary should look like a quadratic curve that classifies $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$ perfectly.

Programming Part

In the programming part, we will guide you through the machine learning steps for developing an email spam detection system. The goal is to help you experience a typical process of supervised learning that includes data preparation, model training, result evaluation, and parameter tuning. We will let you experience three supervised learning models that we have discussed in class: logistic regression, decision tree, and random forest.

Please download `email.csv` from NTU COOL. The file contains 200 rows, each representing one email. For each email, there are 502 columns. The first column originally indicates the recipient's name of the email, and the name has been replaced with an Email No. for privacy protection. Each of the next 500 columns represents the number of times some commonly-used word has been used in the email. The last column contains the labels for prediction: 1 for spam and -1 for non-spam.

Your code will be graded under the following environment:

- python 3.8
- numpy
- pandas
- seaborn
- scikit-learn, while we **prohibit** using
  - sklearn.tree.DecisionTreeClassifier
  - sklearn.ensemble.RandomForestClassifier
  - sklearn.metrics.accuracy_score
  - sklearn.metrics.f1_score

5. (5%) First, complete the function `data_preprocessing` for data preparation. The code splits all the rows to a training set with the first 80% of data, and a test set with the rest of data. In addition, remove the first column ("Email No.") as it is arguably not the input that we want to use. In practice, we shuffle the data before splitting, but here please **do not** do so as this particular data set has been pre-shuffled. (maximum execution time: 10 seconds)

6. (20%) Complete the `fit` and `predict` functions for the `DecisionTree` class that represents the C&RT decision tree. We will `fit` your model with the whole 200-example data set above and then evaluate your `predict` results with a hidden test set on our hand. For `fit`, the pseudo code is

```
function fit(examples)
if cannot branch anymore then
  * build and return a leaf node with the constant final decision
else
  * find a branch such that the total confusion is smallest, store the branch in the root of the tree
  * separate examples to two subsets, one for the left-child and one for the right-child
  * set the left-subtree to be fit(example subset for the left-child)
  * set the right-subtree to be fit(example subset for the right-child)
  * return the tree
end if
```

The calculation of the **total confusion** (usually called the Gini-index) is as follows. The confusion of a subset of a examples labeled $+1$ and b examples labeled -1 is defined as

$$confusion(a,b) = 1 - \left(\frac{a}{a+b}\right)^2 - \left(\frac{b}{a+b}\right)^2.$$

and the **total confusion** after branching from $(c+e)$ examples labeled $+1$ and $(d+f)$ examples labeled -1 to (c of $+1$'s plus d of -1's) and (e of $+1$'s plus f of -1's) is

$$total(c,d,e,f) = \frac{c+d}{c+d+e+f} confusion(c,d) + \frac{e+f}{c+d+e+f} confusion(e,f).$$

The branching step tries to find a branch such that the **total confusion** is the smallest.

To find a branch, consider each component of $\mathbf{x}$ (from 1 to 500). Assume that there are M examples, and consider a simple case where the values of the component on those examples are $v_1, v_2, \ldots, v_M$ with $v_1 < v_2 < \ldots < v_M$. If the values are not sorted or not unique, you can always sort them and remove the duplicates to get sorted and unique values. Trivially, there are only $M-1$ possible "regions" of thresholds t, where all the thresholds within the same region are equivalent—that is, they separate the M examples to two subsets in identical ways. Consider choosing the thresholds among $\{\frac{v_1+v_2}{2}, \frac{v_2+v_3}{2}, \ldots, \frac{v_{M-1}+v_M}{2}\}$. A naïve algorithm for making the best choice is to evaluate the **total confusion** (which is $O(M)$) with each threshold, needing a total time of $O(M^2)$. The naïve algorithm does not use the property that the v_m values are sorted, by the way. You are encouraged (but not required) to think about how to speed up the naïve algorithm. Repeat the threshold-choosing process for each component (from 1 to 500). If there is a tie in branching, pick the best branch from the lowest order component. If there is a tie within that component too, pick the best branch with the smallest threshold.

When we cannot branch anymore, which happens when the confusion on the subset of examples is exactly 0, we declare a leaf with the final decision with the remaining labels (such as the +1 in a subset that only contains +1's). (maximum execution time: 10 minutes)

7. (15%) Complete the `fit` and `predict` functions for the `RandomForest` class that represents the random forest. Your `fit` should call the implemented `bagging` function (which actually implements the so-called *bootstrapping* sampling) to get a random version of the data (which is half of the original size), and then call `DecisionTree.fit` routine on the random data to get a decision tree. Your forest should contain `num_tree` trees (by default 5), with `predict` done by voting of those trees. (maximum execution time: 1 hour)

8. (10%) Implement the `accuracy_score` and the `f1_score` functions. Those are (reversed) error functions that help you evaluate the goodness of your learned model. On N given examples, let

$$\begin{aligned}
\text{TP} &= \sum_{n=1}^{N} [y_n^{label} = +1][y_n^{pred} = +1] \\
\text{FP} &= \sum_{n=1}^{N} [y_n^{label} = -1][y_n^{pred} = +1] \\
\text{TN} &= \sum_{n=1}^{N} [y_n^{label} = -1][y_n^{pred} = -1] \\
\text{FN} &= \sum_{n=1}^{N} [y_n^{label} = +1][y_n^{pred} = -1]
\end{aligned}$$

We have

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \\
\text{F1} &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}
\end{aligned}$$

You can use TP, TN, FP, FN, which are components of the so-called *confusion matrix*, to further understand the performance of your learned model. It may be worth understanding why we suggest using F1 score to evaluate your model, given that the spam classification task is *unbalanced.* (maximum execution time: 10 seconds)

9. (5%) Complete the `cross_validation` function. Split the data (without shuffling) to five folds. For each round, use four of the folds for training and the other fold for validation, and average the performance over the five folds. What are the resulting cross-validation accuracies and cross-validation F1 scores for LogisticRegression, DecisionTree, and RandomForest? Please list those in your report file.

10. (5%) Complete the `tune_random_forest` function that uses 5-fold cross-validation on the F1 score to select the number of decision trees of your random forest within $\{5, 11, 17\}$. What are the resulting cross-validation F1 scores for each choice? Which number appears to be the best choice? Please list those in your report file.

Submission

Write a PDF report for P1-P4, P9, and P10. Complete `hw3.py` for P6-P10. Then, submit a zip file to NTU COOL. The zip file should be named b0x902xxx.zip that contains a directory called b0x902xxx, which includes two files, the completed `hw3.py` and `report.pdf`.