CSE472 (Machine Learning Sessional)

Assignment 1: Decision Tree and AdaBoost for Classification

Introduction

Decision tree is one of the models extensively used in classification problems. In ensemble learning, we combine decisions from multiple weak learners to solve a classification problem. In this assignment you will implement a decision tree classifier and use it within **AdaBoost** algorithm. For any query about this document, contact Sharif sir.

Dataset preprocessing

You need to demonstrate the performance and efficiency of your implementation for the following three different datasets.

1. https://www.kaggle.com/blastchar/telco-customer-churn
2. https://archive.ics.uci.edu/ml/datasets/adult
3. https://www.kaggle.com/mlg-ulb/creditcardfraud

They are different in terms of size, number and types of attributes, data quality (missing attribute values), data descriptions (whether train and test data are separate, attribute description format etc.) etc. Your core implementation for both decision tree and Adaboost model must work for all three datasets without any modification. You can (possibly need to) add a separate dataset-specific preprocessing script/module/function to feed your learning engine a standardized data file in matrix format. On the day of submission, you are likely to be given another new (hopefully smaller) dataset for which you need to create a preprocessor. Any lack of understanding about your own code will severely hinder your chances to make it. Here are some suggestions for you,

1. Design and develop your own code. You can take help from tons of materials available on the web, but do it yourself. This is the only way to ensure that you know every subtle issue needed to be tweaked during customization.
2. Don't assume anything about your dataset. Keep an open mind. Deal with their subtleties in preprocessing.
3. To get an idea about different data preprocessing tasks and techniques, specifically how to handle missing values and numeric features using information gain [AIMA 3rd ed.18.3.6] visit the following link http://www.cs.ccsu.edu/~markov/ccsu_courses/DataMining-3.html
4. Use Python library functions for common preprocessing tasks such as normalization, binarization, discretization, imputation, encoding categorical features, scaling etc. This will make your life easier and you will thank me for enforcing Python implementation. Visit http://scikit-learn.org/stable/modules/preprocessing.html for more information.
5. Go through the dataset description given in the link carefully. Misunderstanding will lead to incorrect preprocessing.
6. For the third dataset, don't worry if your implementation takes long time. You can use a smaller subset (randomly selected 20000 negative samples + all positive samples) of that dataset for demonstration purpose. Do not exclude any positive sample, as they are scarce.

7. Split your preprocessed datasets into 80% training and 20% testing data when the dataset is not split already. All of the learning should use only training data. Test data should only be used for performance measurement. You can use Scikit-learn built-in function for train-test split. See https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data for splitting guidelines.

Decision tree implementation

1. Use information gain to evaluate attribute importance.
2. Control the depth of decision tree using an external parameter.
3. Incorporate the depth parameter in the following pseudo code for decision tree learning.

function Decision_Tree_Learning($examples$, $attributes$, $parent\_examples$) **returns** a tree
  if $examples$ is empty **then return** PLURALITY_VALUE($parent\_examples$)
  else if all $examples$ have same classification **then return** the classification
  else if $attributes$ is empty **then return** PLURALITY_VALUE($examples$)
  else
    $A \leftarrow$ Importance(a, $examples$)
    $tree \leftarrow$ a new decision tree with root test A
    for each value v_k of A **do**
      $exs \leftarrow \{ e : e \in examples$ **and** $e.A = v_k \}$
      $subtree \leftarrow$ Decision_Tree_Learning(exs, $attributes - A$, $examples$)
      add a branch to $tree$ with label ($A = v_k$) and subtree $subtree$
    return $tree$

Adaboost implementation

1. Use the following pseudo-code for Adaboost implementation

function AdaBoost($examples$, L_{weak}, K) **returns** a weighted majority hypothesis
  inputs: $examples$, set of N labeled $examples$ $(x_1, y_1), \dots, (x_N, y_N)$
    L_{weak}, a learning algorithm
    K, the number of hypotheses in the ensemble
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of K hypotheses
    z, a vector of K hypothesis weights
  for $k = 1$ **to** K **do**
    $data \leftarrow$ Resample($examples$, **w**)
    h[k] $\leftarrow L_{weak}(data)$
    error $\leftarrow 0$
    for $j = 1$ **to** N **do**
      if h[k](x_j) $\neq y_j$ **then** error $\leftarrow$ error + **w**[j]
    if error $> .5$ **then continue**
    for $j = 1$ **to** N **do**
      if h[k](x_j) $= y_j$ **then w**[j] $\leftarrow$ **w**[j] $\cdot$ error / (1 − error)
    w $\leftarrow$ Normalize(**w**)
    z[k] $\leftarrow \log\left[(1 - \text{error})/\text{error}\right]$
  return Weighted_Majority(**h**, **z**)

2. As the weak/base learner use decision stump. A decision stump is a decision tree of depth one (i.e., it branches on only one attribute and then makes decision). You should use your decision tree implementation with the depth parameter set to one.
3. Adaboost should treat the base learner as a black box (in this case a decision stump) and communicate with it via a generic interface that inputs resampled data and outputs a classifier.
4. In each round, resample from training data and fit current hypothesis (decision stump) using resampled data but calculate the error over original training data. After learning the ensemble classifier evaluate performance over test data. Don't get confused over which dataset to use at which step.
5. Use +1 for positive decision and -1 for negative decision, so that the sign of your combined majority hypothesis indicates decision.

Performance evaluation

sensitivity

1. Always use a constant seed for any random number generation so that each run produces same output.
2. Report the following performance measure of decision tree implementation on both training and testing data for each of the three datasets. Use the following table format for each dataset.

Performance measure	Training	Test
Accuracy		
True positive rate (sensitivity, recall, hit rate)		
True negative rate (specificity)		
Positive predictive value (precision)		
False discovery rate		
F1 score		

3. Report the **accuracy** of Adaboost implementation with decision stump (K=5, 10, 15 and 20 rounds) on both training and testing data for each of the three datasets.

Number of boosting rounds	Training	Test
5		
10		
15		
20		

Submission

1. Upload the codes in Moodle within **10:00 P.M. of 17th March, 2020 (Tuesday)**. (Strict deadline)
2. You need to submit a report file in pdf format containing the tables shown in the performance evaluation section with your experimental results. No hardcopy is required.

3. Write code in a single *.py file, then rename it with your student id. For example, if your student id is 1505123, then your code file name should be “1505123.py” and the report name should be “1505123.pdf”.
4. Finally make a main folder, put the code and report in it, and rename the main folder as your student id. Then zip it and upload it.

Evaluation

1. You have to reproduce your experiments during in-lab evaluation. Keep everything ready to minimize delay.
2. You are likely to give online tasks during evaluation which will require you to modify your code.
3. You will be tested on your understanding through viva-voce.
4. If evaluators like performance, efficiency or modularity of a particular code, they can give bonus marks. This will be completely at the discretion of evaluators.
5. You are encouraged to bring your computer in the sessional to avoid any hassle. But in that case, ensure an internet connection as you have to instantly download your code from the Moodle and show it.

Warning

1. Don’t copy! We regularly use copy checkers.
2. First time copier and copyee will receive **negative** marking because of dishonesty. Their default is bigger than those who will not submit.
3. Repeated occurrence will lead severe departmental action and jeopardize your academic career. We expect fairness and honesty from you. Don’t disappoint us!