IDS575: Machine Learning & Statistical Methods *Updated: 10/26/2021*

Programming Assignment

Instructor: Moontae Lee *Total Points: 60*

Policy

1. This assignment must be submitted as a group assignment.

Problem 1: Multiclass Support Vector Machine [90 points]

Text classification is an important problem in information retrieval and natural language processing. The task is to classify each text article into one of the predefined classes/categories. Here you have four sets of articles about: 1) operating systems, 2) vehicles, 3) sports, and 4) politics. Each set consists of various articles collected from a corresponding newsgroup where each article is represented by Bag-of-Words (BoW) features. That is, after extracting all the relevant words from the entire dataset, feature vectors are defined as the frequency of words occurring in each article.

Since you have more than two classes, using a single binary SVM classifier is not sufficient. Instead, you are going to combine several binary SVM classifiers to predict multiple classes. Thus, the goal is to train multiple linear SVM classifiers that predict binary classes based on BoW features, then combining them to properly predict one of the four classes. You can download the following data files in the class webpage.

- *articles.train* Training data consisting of 4,000 articles (1,000 articles per class)
- *articles.test* Test data consisting of 2,400 articles (600 articles per class)
- *words.map* A word table mapping every relevant word into its line number.

A single line in the training and test data has the following format:

- *Format: (Class #) (Features)*
- *Class #:* One of 1, 2, 3, 4 (1=operating systems, 2=vehicles, 3=sports, 4=politics)
- *Features:* A space-separated sequence of *word #:frequency*

If a line is *1 11:2 13:1 23:12 25:1 27:2 28:1*, for example, this is an article about *operating system* which contains word 11 (*addresses*) twice, word 13 (*organizations*) once, and so forth.

Word-integer mapping information is given in the *words.map* where each word is mapped into its line number.[1] The followings are all **coding** questions!

(a) Try to load the training and test data into data frames in Python.

(b) First, train four different (hard-margin) linear classifiers. As SVM classifies only binary labels, you have to **replace** the target class number to **1** and all others to **-1** before calling the library function. For instance, if you try to classify whether or not *politics*, you are to use 1,000 articles about *politics* as positive samples and 3,000 others as negative samples for training. Once you learn the four classifiers, the output label of each test example x is determined by the following formula:

$$h_{\mathbf{w},b}(x) = argmax_{k\in\{1,2,3,4\}}(\mathbf{w}^{(k)^T}x + b^{(k)})$$

where $\mathbf{w}^{(k)}$ is the learned weight vector, and $b^{(k)}$ is the biased term for the class k. If the prediction $h_{\mathbf{w},b}(x)$ is different from the ground-truth class, it yields an error. Report the training and test errors of four classifiers, respectively.

(c) Now you are to train soft-margin linear classifiers with different C values from {0.125, 0.25, 0.5, 1, 2, 4, 8, ... , 256, 512}. In order to pick the best C value, you are required to perform a hold-out validation:

1. Split the entire training data randomly into 75% for training and 25% for validation.
2. For each C value, learn four binary classifiers similar to part (a) but only on the training data.
3. Measure the overall classification error on the validation data.
4. Pick the C with the lowest validation error.

Plot a graph showing **both training and validation errors together** with varying C in log-scale. (i.e., x-axis: $\log_2 C$, y-axis: error rate) What are the best C value for multiclass classification? (Note: You should apply one uniform C value identically to all four soft-margin classifiers)

(d) With the best C value chosen from part (b), learn four soft-margin classifiers again on the entire training set. (Note that your classifiers from part (b) used only 75% of the training set for learning, holding out 25% for validation) Test your newly learned best classifiers on the test set similar to part (a) where the output label is determined by the *argmax* class given in the formula in part (a). Compare the test error rates to hard-margin classifiers on part (a). Which classifier works better? Justify your observation.

(e) For this problem, you will normalize feature vectors so that the feature vectors of each example have unit length. For each example $x = (x_1, x_2, ..., x_n)$, divide every component into $\|x\|_2$ so that $\|x\|_2 = 1$. Repeat the part (b) with normalized features and measure the test error rates again with newly picked C value. Compare the new test error to

[1] *The words.map* is a simple text file having one word per each line. Every word corresponds to its unique line number starting from 1

previous test error from soft-margin classifier without normalization, and explain why normalization makes a difference.

(f) What you have done so far is one way to extend binary SVM to a multiclass classifier, which is called *1-vs-all*. There is another way of called *1-vs-1*, where you are supposed to train all $\binom{4}{2}$ binary classifiers distinguishing between every pair of classes. Then for each test example, you will pick the (not necessarily unique) class that achieves the highest votes.

Formally speaking, suppose that the binary classifier h_{ij} is trained by taking the examples from class i as positives and the examples from class j as negatives. [2] Then for each test example x, add one vote to class i if h_{ij} says x is in class i. Otherwise, add one vote to class j. Finally *1-vs-1* assigns x to the class with the maximum votes. Since vote is added by one each time, the maximum voted class is not necessarily unique.

Compare the **accuracy** of *1-vs-1* to *1-vs-all*. Accuracy should be measured fairly between two methods via using the normalized features as done the in part (e) and the best C parameters as done in the part (b).

[2]Now you use 1,000 positive samples and only 1,000 negative samples for learning each classifier.