KU SCHOOL OF BUSINESS
The University of Kansas

Advanced Machine Learning | Homework 4

Due March 30 (11:59pm)

Instructions

Submit your solutions on Canvas. Your submission must include the Python code for all questions. Please submit your code all in one file. Adding comments may help the instructor understand your code and is therefore encouraged. **The solution for each question must be generated as a Python variable or output files with specific names as instructed**. All solutions should be generated by running your code without any customization or modification by the instructor. If your code requires input files, you can assume all input files are in the working directory -- **the full path to the file (which changes with computer) should not appear in your code**. Your files should end in .ipynb (Python Notebook). **Please also make sure to add a print statement for each of the variables required for each problem (this makes it easier to grade your homework)**.

Problems

1. Download "*news_label.csv*" from Canvas. Each class label equals either "med" (medical science) or "space" (space science), which indicates the main subject of each message. These class labels are under the column name "Class". The documents you will use are under the column name "Message". You task is to use KNN with k=7 to predict the class of each message. For validation purposes, a file "*partition_hw4.pkl*" is given. This pkl file contains a dictionary with "train_inds" and test_inds as keys. Load this pkl file into Python and use the dictionary keys "train_inds" and "test_inds" to partition the data intro training and testing sets. *Do not generate your own partition!* Use "space" as the positive class.

   Please implement the following pre-processing steps to create your DTM:
   - Turn every letter to lower case
   - Remove punctuations and numbers
   - Remove the words in the default sklearn stopwords list
   - Strip whitespace
   - Apply stemming
   - Terms that appear in only one document or in all documents must be discarded
   - Use a customized vocabulary that consists of only the unigrams whose total frequency is at least 100.
   - Use the regular term frequency (CountVectorizer) as the weight of terms in DTM.

   Output variables:
   - `myvocab1`: (10 points) A list of unigrams whose total frequency is at least 100, obtained after applying the above preprocessing steps.
   - `dtm1`: (10 points) A DTM satisfying the requirements above.
   - `pred_label_knn1`: (5 points) The predicted class labels for the testing set.
   - `accuracy_knn1`: (5 points) The accuracy of KNN on the testing set.
   - `precision_knn1`: (5 points) The precision of KNN on the testing set if "space" is the positive class.
   - `recall_knn1`: (5 points) The recall of KNN on the testing set if "space" is the

positive class.

2. [Continuation oof Problem #1] (30 points) Improve your kNN model from Problem #1 such that the accuracy on the testing set is above 90%. The solution is open. You can try to improve the KNN by changing the following parameters:
   - Include more or fewer unigrams in the DTM by changing the CTF threshold. (Be careful. Too many unigrams may lead to a long running time.)
   - Changing the value *k* (i.e., the number of *k* nearest neighbors considered)
   - Try DTMs with term weight being TF and (normalized/unnormalized) TFIDF.
   - Try a vocabulary containing the top bigrams or both top unigrams and top bigrams.

   You don't need to try all of them. As long as the accuracy is above 90%, you will get all 30 points. Your output variables should mirror those in Problem 1, but end in a "2" (e.g., dtm2, myvocab2, etc.). Your code will be reviewed by the instructor or TA.

3. [Continuation of Problem #2] Use the DTM that produced the best KNN result in Problem #2 to build neural network models. Your code should search across the following hidden node values: 10, 20, 30, 50. When building each model use 20% of the training data as a validation set, a batch size of 60, and 15 epochs. Your code should keep track of the validation AUC for each model. Once all models have been built, your code should rebuild the model that produced the best validation AUC. This best model should then be evaluated using the test set. Report the accuracy and AUC of the test set. If needed, please feel free to consult the lecture and lab slides for help solving this problem.
   Output variables:
   - `h_val_auc`: (10 points) A DataFrame containing the validation AUC values for each model along with the number of hidden nodes each validation AUC value corresponds to.
   - `best_model`: (10 points) A trained neural network model that uses the number of hidden nodes found to produce the best validation AUC.
   - `test_res:`(10 points) A list containing the testing loss, accuracy, and AUC values in that order (i.e., test_res[0] is test loss, test_res[1] is test accuracy, test_res[2] is test AUC).