CSE 5243: Homework #3

Deadline: **11:59PM on 03/18/2020**.
No late submissions will be accepted.

Instructions

We currently use a 100-point scale for this homework, but it will take 10% of your final grade. The bonus points from the programming competition will be directly added into your final grade.
Have Questions about Homework?
Please create a post on Carmen Discussion to get timely help from other students, the TA, and me. Everyone can benefit from first checking what have already been asked. Please try to avoid directly sending me/TA emails unless necessary (e.g., for sensitive matter).

1 Task: Document Classification (100 points)

This assignment is the second part of a longer-term project. The objective of this assignment is to build and compare document classification models based on your preprocessed data from assignment 2. In case you encountered some difficulty in assignment 2 and the preprocessed data is not usable for this assignment, a starter notebook with minimal preprocessing is provided and you could build your classification models on top of that. **Please KEEP IT CONFIDENFIAL to avoid abuse in future courses**.

- **Programming Requirement**: ***You are required to use Python and Jupyter Notebook. If you really have to use other programming languages, please talk to the TA and make sure she can run your code successfully***.
- **Data**: 20 Newsgroups dataset, which contains text documents regarding different topics. It can be downloaded here `https://ysu1989.github.io/courses/sp20/cse5243/20news-train.zip`. Unzip the file, and there will be 20 subdirectories, each regarding a certain topic (indicated by the subdirectory name) and containing a set of text documents about that topic (each file is a document).

- **Data Format**: Each subdirectory contains posts to a newsgroup regarding a certain topic (i.e., the name of the subdirectory). Each post is a text document. Note that although they don't have the `.txt` suffix, they can be viewed using text editors or read in your code as ordinary text files. Each document has two main sections: meta-data and main body. Meta-data are fields like "From" (who posted it), "Organization", and "Subject" (it could be very indicative of the topic of the post). The meda-data usually has "Lines" as the last line, but not always. There is a new line between the main body and the meta-data.

- **Task**: In this assignment, your task is to build document classification models based on your preprocessed data from assignment 2. See more details about the task requirement in Sec 1.1.

- **Libraries**: You should implement the classification models by yourself. You can use off-the-shelf libraries like NLTK[1] and scikit-learn[2], but only for preprocessing, not classification.

- **Competition**: We will be hosting a competition similar to Kaggle[3] in this assignment. What is released to you is only the public training set, and there is a hidden test set. Students whose model achieves the **top 10 scores** on the hidden test set will get **bonus points**. Specifically, top 3 will get 5 bonus points and top 4-10 will get 2 bonus points towards the final grade. There will be a best report award (2 bonus points) that goes to the one who writes the best report. Note that **as long as you successfully achieve the requirement of the assignment, you will get the full points. The competition is only for bonus points**.

  Please note that your feature vector for each document should only be based on the sentence content and should not contain the class label.

1.1 Task Description

1.1.1 Overall goal

The goal is to build and evaluate classifiers to predict the topic of a given document. The true label of each document is the name of the subdirectory it is in, which you have mapped into an integer in assignment 2.

1.1.2 Dataset split

To achieve good generalization performance on the hidden test set, you should do holdout evaluation or cross-validation on the public training data to select the best model. Take holdout for example, you should split the given training

[1] `https://www.nltk.org/api/nltk.tokenize.html`
[2] `https://scikit-learn.org/stable/`
[3] `https://www.kaggle.com/competitions`

data into a training set (e.g., 80% of the data) and a validation set (e.g., 20% of the data). You can do either random sampling or stratified sampling. You then build/train different models on the training set, select the best model based on their performance on the test set, and submit it for evaluation on the hidden test set. Note that if your vocabulary should only be constructed using the training set, not the validation set. Once you have determined the best model, you could optionally retrain the model using all the training data (training+validation). Please detail how you split your data in your report.

1.1.3 Feature Selection

You have a huge vocabulary (200K+ if not truncated), and each token is a feature/attribute. It will likely pose a challenge to your classification models (remember "curse of dimensionality"?). Therefore, it's a good idea to do feature selection and try different methods to select "important" features. Let us assume you start with a feature vector that comprises or keeps track of 2048 words. What you will need to do is to identify and leverage a measure of importance to pare down the feature vector size down to, say, 256 or 512 words, and compare and contrast the performance of **the original feature set vs pruned feature set**.

You have the freedom to define your own measure of importance. You may consider one of the following two intuitive choices:

(1) One intuitive measure is the overall frequency of a word in the entire dataset. You can keep the top-K most frequent words (e.g., top 10,000 most frequent words in the entire dataset) and prune less frequent ones. Or, you can keep words that appear at least K times (e.g., K=5) in the dataset. If you choose to use this kind of measure, you may first remove stop words[4] like "the" and "a" from your dataset, as they are not informative but very frequent.

(2) TF-IDF. IF-IDF is the product of two statistics, term frequency and inverse document frequency. The tf-idf value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the entire corpus, which helps to adjust for the fact that some words appear more frequently in general. For more information, please refer to `https://en.wikipedia.org/wiki/Tf%E2%80%93idf`.

For word t that occurs in sentence d, you can calculate its tf-idf in the simplest way in the previous link:

- $\mathrm{tf}(t,d) = f_{t,d}$, where $f_{t,d}$ is simply the frequency of word t in document d.
- $\mathrm{idf}(t,D) = \log(\frac{N}{|d \in D : t \in d|})$, is a measure of whether word t is common or rare across all sentences D. Note in your experiment, you can use your training document set as D. It can be obtained by dividing the total number of documents (i.e., N) by the number of documents containing the term (i.e., $|d \in D : t \in d|$), and then taking the logarithm of that quotient.
- Finally, the tf-idf score of word t in document d, is $\mathrm{tf}(t,d)$ * $\mathrm{idf}(t,D)$.

[4]`https://www.geeksforgeeks.org/removing-stop-words-nltk-python/`

Note that you have the freedom to decide what importance measures to use, how many features should be kept, etc. Please detail what you used in your report.

1.1.4 What classification algorithms to use?

You are expected to implement **two** different classifiers (K-nearest neighbors, decision trees and Naïve Bayes are your possible choices). Then you choose which classifier to use for evaluation on the hidden test set.

Our ***strong recommendation*** is that after you implement the classifier(s) by yourself, you also try existing software/packages for the same algorithm(s) to make sure you have correct implementation(s), unless you are very confident. This is not required, but highly encouraged. Keep in mind that if you use existing software, you may have to transform your feature vector dataset to match their input specifications. Your report should describe any further data transformations you have had (if any) in order to work with these software packages.

Note: No matter you choose to do the bonus task or not, make sure you implement at lease one on your own by following the algorithms/pseudo code. For example, simply calling a built-in function of some Python package is NOT your own implementation. Similarly, if you want to test the LSTM network, using existing LSTM cells is not regarded as your implementation, but you can still earn the bonus points, if you also evaluate another two basic classifiers and one of them is implemented by yourself.

1.1.5 What results to expect?

To sum up, assuming you pick two classifiers (CL-1; CL-2) and you have the original feature vector from the last assignment (FV-1) (see notes below on things you may need to correct) and a pared down feature vector where you have selected some features (FV-2) as discussed in Section 1.1.3. You will need to run 4 sets of experiments (**each classifier on each type of feature vector** – think of it as a 2X2 experiment configuration matrix).

For each set of experiments (configuration) you will need to report:

(1) the overall scalability of the resulting approach (time to build classifier model – this is also known as the offline efficiency cost)

(2) time to classify a new tuple (also known as the online efficiency cost)

(3) the accuracy of the classifier respectively on the training and validation sets.

(4) **comments on the advantages (or disadvantages, depending on what you find) using the features and the method you adopted**. Examples could include – feature selection helps improve accuracy and efficiency costs, or feature selection is ineffective. Back up your observations with numbers/graphs/charts.

Please note that if the TA provides feedback on your preprocessing step and obtained features in HW2, please make sure the feedback is addressed in this assignment

What You Should Turn in:

You are expected to turn in the following files:

- Source code in a `.ipynb` file for Jupyter notebook. You do not have to include the raw dataset. Your Jupyter notebook should be made such that one can successfully run all the cells to replicate your results as long as the data directory is in the same directory as the notebook. In the Jupyter notebook, you should explain, in Markdown cells[5], what the following code cell is doing and how to interpret the output, plus any comment you'd like to add that may help others like the TA understand. **If you use off-the-shelf libraries like NLTK, please document which version of what library is used and how you acquired and installed it so that the TA can properly run the code.** The lastest version of NLTK is recommended.
- Important - Evaluation on the hidden test set: The hidden test set will be in a directory named "20news-test" at the same directory as your notebook, with the same file structure (documents belonging to the same topic are put under the same subdirectory) as the training set. The last code cell of your notebook should be running your final model on this hidden test set (including the same kind of feature extraction and preprocessing you do on the training set) and predicting a class label for each document in the test set. The TA will then calculate the final test accuracy based on the predictions. You should test this cell by making a test set of your own using the validation set to avoid troubles running this cell (e.g., a common bug arising from such a setup is `list index out of range`).
- A report describing what you have done/attempted but failed, analysis, takeaways you got from the assignment. There will be a best report award!

Detailing what you did is very important even if it did not work, in which case it's even more important since the TA can then give partial points accordingly. Describe any difficulties you may have encountered and any assumptions you are making.

How to submit your files to Carmen:

After you finish all the problems, you just need to upload a single .zip file to Carmen, as we mentioned in Instructions.

The easiest way to upload files specific to the stdlinux environment is by opening carmen from the stdlinux and uploading the corresponding files from there. To open carmen from stdlinux,

[5] `https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html`

- Login to stdlinux using CSE remote access. Check here for *remote access*, if needed: `https://cse.osu.edu/computing-services/resources/remote-access`.
- Open terminal
- Type "firefox". Click enter
- Navigate to carmen.osu.edu

If it is necessary to transfer files from Windows/Mac environment to stdlinux, one can make use of SCP and SFTP protocols to achieve so. This site[6] can provide additional details. Contact CSE Help Desk if further help is needed setting up File transfer clients.

Necessary Details:
a. Host Name: stdlinux.cse.ohio-state.edu
b. Port Number: 22
c. User Name: osu user-id
d. Password: OSU password

[6]`https://u.osu.edu/floss/documentation/using-scp-sftp-for-ohio-state-cse-students-to-sync-projects-and-remote`