MIE 1513 Decision Support Systems
Assignment 2: Machine Learning (ML)

This assignment allows students to apply machine learning knowledge through document classification using supervised learning techniques and to perform a performance analysis of the learned classifier. In addition, this assignment contains a competitive component to encourage students to explore feature sets, feature representations and hyperparameter tuning.

- Programming language: Python (Google Colab Environment)
- Due Date: Posted in Syllabus

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals, (2) for providing appropriate answers to the questions in a Jupyter notebook (named `ml_assignment.ipynb`) committed to the student's github assignment repository, and (3) attendance at the code review session, running your solution notebook for instructors, and providing clear and succinct answers in response to instructor questions regarding your solution.

Please note the plagiarism policy in the syllabus. If you borrow or modify any *multiline* snippets of code from the web, you are required to cite the URL in a comment above the code that you used. You do not need to cite tutorials or reference content that demonstrate how to use packages – you should certainly be making use of such content.

What/how to submit your work:

1. All your code should be included in the notebooks named `ml_assignment.ipynb` that is provided in the cloned assignment repository.
2. Commit and push your work to your github repository in order to submit it. Your last commit and push before the assignment deadline will be considered to be your submission. You can check your repository online to make sure that all required files have actually been committed and pushed to your repository.
3. A link to create a personal repository for this assignment is posted on Portal.

Credit: This lab's notebook material has been prepared based on an Advanced Scikit-Learn tutorial provided by Data Scientist Workbench.

Notes that you should pay attention to

1. The same auto-grade restrictions apply for this assignment as they did for the previous assignments. Please check the pinned posts about autograder on Piazza for more information.

2. During the code review, we will ask questions about your free text answers and your results in general as well as some details about your code and possible alternative approaches you might have taken. If you made both an on-time and late submission, you will have to tell us which version we should use for code review with a 30% deduction applying to the code review if you wish us to use your late submission. Note: you will receive a zero for the code review if you did not make an on-time or late submission – in this case you are considered to not have submitted your assignment.

This assignment has *7 points* in total, allocated as follows:

- Auto-grading points(5 points):
  - Q1: 1.5 points
  - Q2: 0.5 points
  - Q3: 0.5 points
  - Q4: 0.5 points
  - Q5: 0.5 points
  - Q6: 0.5 points
  - Q7: 1 point
- Code review (2 points)

Points will be deducted from the auto-grading points for missing or incomplete text answers.

1 Before the Introductory lab

In the lab you'll familiarize yourself with several python machine learning libraries;

- **scikit-learn** a powerful machine learning library for python.
- **pandas** a powerful python data analysis toolkit
- **numpy and scipy** powerful computing packages for python
- **matplotlib** a python 2D plotting library

The scikit-learn documentation can be found on its website at `http://scikit-learn.org/stable/`

The pandas documentation can be found on its website at `http://pandas.pydata.org/pandas-docs/stable/`

The numpy and scipy documentations can be found on its website at `http://docs.scipy.org/doc/`

The matlibplot documentation can be found on its website at `http://matplotlib.org/`

You just need to execute the import statements at the top of the lab ipynb notebook to import these standard Python libraries.

2 In the Introductory lab

The lab can be found within your assignment-ml repository, under the lab_ml.ipynb file, clone it and open the Jupyter Notebook in the Google Colab to follow along.

In the Introductory lab section, we will go through the process of loading a dataset called 20 Newsgroups, and run a baseline classification using logistic regression.

The data you need is distributed on the course website but should be identical to the data found at `http://qwone.com/~jason/20Newsgroups/` with the following description:

> The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. To the best of my knowledge, it was originally collected by Ken Lang, probably for his Newsweeder: Learning to filter netnews paper, though he does not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

3 Main Assignment

In the introductory lab, you have been exposed to the 20 newsgroup dataset. The lab included loading and basic preparation of the data and a baseline classification using logistic regression and covered the libraries used in this assignment.

In this assignment, we will analyze different choices when configuring your classifier:

Feature Set Feature selection may not only improve classification performance (especially when data is sparse), but it can also improve computational performance. For this assignment, you can experiment with any features and feature selection technique, although you will probably find single term (unigram) features and a frequency-based feature selection approach to work well.

Feature Encoding As two simple variations of feature encoding, consider a boolean $\{0, 1\}$ vector encoding (as produced in the lab) as well as a term frequency (TF) encoding (which you need to produce yourself).

Amount of Data While generally you should use all training data available, it is instructive to vary the amount of training data provided to an algorithm to assess the impact of the amount of data on learning performance.

Hyperparameters All algorithms typically have at least one hyperparameter that needs tuning. We should use cross-validation for tuning hyperparameters in practice, but in this assignment, we will simply analyze performance as a function of hyperparameters.

We provide a template notebook for the coding part that you must use in your submission. Autograding requires that all function names are not modified and that the entire notebook execute in one pass from bottom to top in Colab environment (please verify before submitting). Please fill in the missing part of the functions in the template notebook to provide the experimental results. The function you need to implement for each part of the assignment is clearly noted in the question description and the return value is validated using asserts in the end of the functions in the template notebook.

Please note the following:

- For all questions except Q3, use a hyperparameter that you find performs well (can be the default value). In general one should use nested cross-validation (CV) for hyperparameter tuning, but we avoid that here due to the time-consuming nature of nested CV.
- Do NOT change the function name. We need that for grading.
- Do NOT use/refer to global variables in the functions. This will cause your code to fail during autograding because we execute functions independent of any content outside the function. However, you can (and should) call other functions when needed.
- If you need to use **external libraries** that are not provided in lab, please ask for approval on piazza.

Please answer the following questions:

Q1. Binary Encoding

(a) The function `binary_baseline_data` takes in a list of files and runs a baseline evaluation based on a binary encoding of the most commonly words as feature set. Based on the above description of choices, please describe the feature set, the amount of data, and the hyper parameters used in this baseline

(b) Try to improve the results of the baseline by improving (only) the feature set. You can use all the techniques covered in the IR lab to improve your features (e.g., stemming, lemmatization, lowercasing, stopwords; you can use NLTK for this purpose). Your code should be written in the provided function `binary_improved_data` (input and return values should be similar to `binary_baseline_data`).

(c) Calculate the train accuracy and test accuracy of your new function (partial code provided). How did the results change?

(d) Different train-test splits can lead to different results. In order to get a more robust estimation of the performance of your classifier, we want to calculate the mean and the 95% confidence interval on the accuracy of the classifier over a set of multiple runs with random splits. Notice that the function `train_test_split` takes an argument `random_state` that can be used to create different (random) splits by passing a random value to this argument. Please implement the function `random_mean_ci` that creates multiple random splits of your dataset (the argument `num_tests` will determine the number of splits to evaluate) and returns a tuple `(train_mean, train_ci_low, train_ci_high, test_mean, test_ci_low, test_ci_high)` that represent the mean and the low and high ends of the 95% confidence interval for both the training accuracy and the test accuracy. We recommend you use test_size=0.3 in train_test_split.
Note the following:

- To generate random numbers for the `random_state`, you can use the following code `random.randint(1,1000)` that generate a random integer in the range 1 to 1000.
- The code to calculate the mean and confidence interval is provided, given a lists of accuracy results (the variables `train_results, test_results`) for the different random splits.

(e) Run the above function for 10 iterations (`num_tests=10`, see provided code). What do the average and 95% confidence intervals tell you? Are they more informative than a single trial? Yes or no, and why? [2 sentences.]

(f) Implement a function `random_cm` that produces a confusion matrix that is based on multiple random splits. Such matrix is created by summing the confusion matrices for the different splits. Build a confusion matrix based on the results of 10 iteration (produced, as before, by calling `train_test_split` function with random `random_state` values. Note that partial code is provided that includes the summation of the different confusion matrices.

(g) Show the confusion matrix for 10 random splits (`num_tests=10`, see provided code). Are some classes more easily confused with others? Which ones and why? [2 sentences.]

Q2. Number of Features

In this question, you will vary the number of words used as features and see how it affects the results. Please be careful to only use a single train/test split for this evaluation.

(a) Calculate the train accuracy and the test accuracy when using the first p percent of the features (for a fixed ordering of features), $p \in [10\%, 30\%, 50\%, 75\%, 85\%, 100\%]$. The function `feature_num` has partial code you need to complete. It returns a dataframe of the results.

(b) Use the provided code to plot the results. Explain any trends you see (average over multiple trials if trends are not clear). [1 sentence.]

Q3. Hyperparameter Tuning

(a) Calculate the train accuracy and the test accuracy for different values for the hyperparameter C: [10^{-3}, 10^{-2}, ..., 10^{0}, ..., 10^{3}]. The function `hyperparameter` has partial code you need to complete. It returns a dataframe of the results.

(b) Use the provided code to plot the results (we use a logarithmic x axis). Explain any trends you see (average over multiple trials if trends are not clear). [1 sentence.]

Note: In practice, you need to tune hyper-parameter on the validation set only, not the test set!

Q4. Feature Encoding

In this question, you will evaluate the effect of using term-frequency (TF) encoding instead of a binary encoding on this dataset.

(a) Implement a TF encoding in the function `tf_improved_data`. You should use your improved function `binary_improved_data` from Q1 (b) as a base, and change the encoding from binary to TF.

(b) Compare the two encodings by comparing their mean accuracies and 95% confidence intervals over 10 trials. Use the function `random_mean_ci` from Q1 (d). Which method performs better on this dataset? Why do you think this occurs? [1 sentence.]

Q5. Comparison vs. Naive Bayes

In this question, you will compare mean accuracy and 95% confidence interval of the logistic regression classifier to a naive bayes (NB) classifier.

(a) Implement a naive bayes classifier evaluated over multiple random splits in the function `nb_random_mean_ci`. You should use your `random_mean_ci` function from Q1 (d) as a base, and change the classifier from logistic regression to NB. Use the encoding (binary or TF) you found to be better.

(b) Compare the two classifiers by comparing their mean accuracies and 95% confidence intervals over 10 trials. Which method performs better on this dataset? Why do you think this occurs? [1 sentence.]

Q6. Binary Logistic Regression

In this question you will build a binary logistic regression that is trained to classify the target `sci.med` vs. any other target. Use the binary encoding of features of this question.

(a) Implement the function `binary_med_data` that return the features and targets dataframe. In this question there are only two possible targets: 1 for `sci.med` and 0 for any other label. You should use the code in `binary_improved_data` as a base, and change the targets to be binary.

(b) Using the function `random_mean_ci` in Q1 (d), calculate the average accuracy and 95% confidence interval over ten iterations (`num_tests=10`, see provided code). What do the average and 95% confidence intervals tell you? How do they compare to the multiclass logistic regression in Q1 [1 sentences.]

Q7. Classification with Ranking

In this question, you will again build a multi-class classification system. However, this time you will be interested in the ranked results of your system. You will be putting together a system to optimize the **average precision score**. The classification results will be ranked by the probability estimates from the classifier. To evaluate your system, use the provided `calculate_average_precision_at_k` function (where k is defaulted to the number of testing documents). You may use any feature set and feature encoding you wish for this question, but you may only use the standard classifiers built into sci-kit learn. Your model should be built and returned in the provided function `build_model_q7`. There is a runtime limit of 15 minutes for this question. Running the function `calculate_average_precision_at_k` with an input of `build_model_q7()` must run on Google Colab with a runtime less than 15 minutes. Models that exceed this limit will receive 0 marks for this question. For example usage, please see the Jupyter notebook.

(a) Implement the `data_q7` function using your chosen feature set and feature encoding.

(b) Implement the `build_model_q7` function with a machine learning model of your choice.

(c) In markdown cells, provide:

- A clear and concise description of your chosen feature set and feature encoding
- The name of the classifier you chose
- Why you chose the feature set, feature encoding and classifier you did
- The final AP performance that your choices attained. We will verify this score by running the model returned by your `build_model_q7` function, and the data as returned by your `data_Q7` function.

Your score for Q7 will be calculated based on your effort and understanding shown in this question. We encourage you to explore different parameters that are relevant, and the effects they have on the machine learning performance.

Note: We provide a set of training and testing file names as an example of the final dataset your models will be tested against. However, the actual dataset we use in grading will be different. It is your responsibility to try to improve your classifier so that it will do well across all train-test splits. (Number of documents of each class will be balanced for all train-test splits. The ratio of train to test documents will be 2-to-1.)

Hint: Recall the average precision formula:

$$AP@k = \frac{1}{number\_of\_relevant\_documents\_at\_k} \sum_{i}^{k} Precsion@i \cdot rel(i)$$

In this assignment, we evaluate based on full AP, where k= total number of testing documents. Additionally, since this is a multi-class classification , $number\_of\_relevant\_documents\_at\_k$ always equal to k.