Programming Assignment 1: k Nearest Neighbors

Instructions:

- The aim of this assignment is to give you a hands-on with a real-life machine learning application.
- Use separate training, and testing data as discussed in class.
- You can only use Python programming language and Jupyter Notebooks.
- There are three parts of this assignment. In part 1, you can only use **numpy**, **scipy**, **pandas, matplotlib** and are not allowed to use **NLTK, scikit-learn or any other machine learning toolkit**. However, you have to use **scikit-learn** in parts 2 & 3.
- **Carefully read the submission instructions, plagiarism and late days policy at the end of assignment.**
- Deadline to submit this assignment is: **Friday, 16th October 2020.**

Problem:

The purpose of this assignment is to get you familiar with k nearest neighbor classification. By the end of this assignment you will have your very own "Tweet Sentiment Analyzer". You are given Twitter US Airline Sentiment Dataset that contains around 14,601 tweets about airlines labelled as positive, negative and neutral. Your task is to implement k nearest classifier and use it for predicting the sentiment of the tweets about airlines.

Dataset:

The dataset is divided into training and test sets in a stratified fashion. Train set contains 11,680 tweets and test set contains 2,921 tweets. There are three classes in the dataset: positive, negative and neutral.

Preprocessing:

In the preprocessing step you're required to remove the stop words, punctuation marks and other unwanted characters from the tweets and convert them to lower case. You may find the string and regex module useful for this purpose. A stop word list is provided with the assignment statement.

Feature Extraction:

In the feature extraction step you'll represent each tweet as a bag-of-words (BoW), that is, an unordered set of words with their position ignored, keeping only their frequency in the tweet. For example, consider the below tweets:

T1 = Welcome to machine learning, machine!
T2 = kNN is a powerful machine learning algorithm.

The bag-of-words representation (ignoring case and punctuation) for the above tweets are:

Vocabulary	welcome	to	machine	learning	knn	is	a	powerful	algorithm
T1	1	1	2	1	0	0	0	0	0
T2	0	0	1	1	1	1	1	1	1

Note: We only use the training set to construct the vocabulary for the BoW representation.

Part 1:

Implement k Nearest Neighbors from scratch keeping in view all the discussions from the class lectures. Specifically, follow the steps given below:

Input: Training samples $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, Test sample $d = (x)$ and number of nearest neighbors k. Assume $x_1, x_2, \dots, x_n$ *and* x are m-dimensional vectors.

Steps:

- Compute the distance between d and every sample in D.
- Choose the k samples from D that are nearest to d; denote the set by $S_d \in D$.
- Assign d the label y of the majority class in S_d.

Use Euclidean as your distance metric. You can either use sorting or Quickselect to choose k nearest neighbors. Make sure you code it generically enough so that it can run with any value of k. Handle the ties by backing off to k-1 neighbors. Report classification accuracy, macro-average (precision, recall, and F1) and confusion matrix on the test set with $k = \{1, 3, 5, 7, 10\}$. Also, report plots with the k on x-axis and performance measures (accuracy, precision, recall, F1) on y-axis.

Use the procedural programming style and comment your code thoroughly.

Part 2:

Use scikit-learn's kNN implementation to train and test the k nearest neighbor classifier on the provided dataset. Use scikit-learn's accuracy score function to calculate the accuracy, classification report to calculate macro-average (precision, recall and F1) and confusion matrix function to calculate confusion matrix on test set with $k = \{1, 3, 5, 7, 10\}$. Also, report plots with the k on x-axis and performance measures (accuracy, precision, recall, F1 etc.) on y-axis.

Part 3:

In this part you'll enhance your input features i.e. instead of using sparse BoW representation you'll use a 300-dimensional dense vector representation also known as Word2Vec. Word2Vec is a popular representation of text and is capable of capturing linguistic contexts of words. It was developed by Mikolov et al. in 2013.

- Download pre-trained 300-dimensional word2vec *representations* from here. (It's 1.5 GB! Don't wait till last date)

- Install and import gensim to use the pre-trained representations.
- A code snippet is provided here that shows how to load a pre-trained embedding model and represent a sample sentence with a dense vector.
- Re-run part 1 & 2 with Word2Vec representation and report the results.
- [Tip] You can use Google Colab for implementation, it will download the 1.5 GB data in less than two minutes.
- [Optional] The interested once can read The illustrated word2vec by Jay Alammar to grasp the concept.

Submission Instructions:

Submit your code both as notebook file (.ipynb) and python script (.py) on LMS. The name of both files should be your roll number. If you don’t know how to save .ipynb as .py see this. **Failing to submit any one of them will result in the reduction of marks**.

Plagiarism Policy:

The code MUST be done independently. Any plagiarism or cheating of work from others or the internet will be immediately referred to the DC. If you are confused about what constitutes plagiarism, it is YOUR responsibility to consult with the instructor or the TA in a timely manner. No “after the fact” negotiations will be possible. The only way to guarantee that you do not lose marks is “DO NOT LOOK AT ANYONE ELSE'S CODE NOR DISCUSS IT WITH THEM”.

Late Days Policy:

The deadline of the assignment is final. However, in order to accommodate all the 11th hour issues there is a late submission policy i.e. you can submit your assignment within 3 days after the deadline with 25% deduction each day.