Neuroscience 299: Computing with High-Dimensional Vectors
Assignment 9: Solving classification problems
Due November 3, 1pm

Reminder: Please do *either* the writing assignment *or* the programming assignment. Expected length for the writing assignment is approximately 250-500 words, but there is no strict minimum or maximum.

Writing assignment:

When do HD computing/VSA approaches to classification perform well? In your response, you should address the following 2 cases:

- Discuss a specific example where an HD computing/VSA approach provided strong performance on a classification task. Describe the problem and the structure of the data. What about the problem, if anything, can help explain why the algorithm performed well? What choices of the distributed representations (if any) were useful?
- Describe a kind of classification task (actual or hypothetical) where an HD computing/VSA approach did not (or would not) perform well. *Why* does (would) HD representations be a poor choice for this kind of task, and what might be a better alternative?
    - To be clear: Your response shouldn't boil down to something that can be summarized as "neural nets can do *X* and we don't have a VSA for that yet." That would be a fine start, but then you should add suggestions for why VSAs would be bad for that particular task.

Make sure to include proper references/citations to any papers mentioned.

Programming assignment:

This time, the programming assignment includes 2 parts. A complete assignment must complete both parts.

Part 1: Language identification.

This part of the assignment is largely built on the programming assignment in Module 4. Therefore, please refer to the corresponding document for the detailed description of the problem formulation and training data.

As a reminder, the training data are provided in a folder called "training_texts". Please download it using the following link:
https://www.dropbox.com/sh/hux29e50hzzwpki/AACcfkhLBOvBXP4yai_QOjKHa?dl=0
Please recall that depending on the efficiency of your implementation, processing the dataset may take considerable time. If processing takes too long, consider changing the variable "dataSizeLimit" in the "Parameters" cell. This variable regulates the length of the training sequence. By default, it is set to 0, which means that the whole training sequence is considered.

For this assignment you will also work with test data, which are based on the Europarl Parallel Corpus. The test data also represent 21 European languages. The total number of samples in the test data is 21,000, where each language is represented with 1,000 samples. Each sample in the test data corresponds to a single sentence. The average size of a sample in the test data is 150.3 ± 89.5 symbols.

Test data are provided in a folder called "test_texts". Please download it using the following link: https://www.dropbox.com/s/7jv4rfc16whubcy/test_texts.zip?dl=0

For this part of the assignment, you are provided with the Jupyter notebook (see course website) that includes the code to support loading both training and test data and implements the functionality requested for the assignment in in Module 4, i.e., the formation of profile HD vectors for each language.

The provided Jupyter notebook is only missing a few steps (search for the ***#ToDo*** comments in the notebook), which are related to making the predictions for test data (note that the profiles that the HD vectors obtained from the training data are essentially a classifier) and visualizing the confusion matrix.

Please perform the following experiments and answer the following questions:

1. Are the results in your confusion matrix qualitatively similar to the ones reported in "Language Geometry Using Random Indexing"?
2. Which language was the hardest to classify? Why do you think this is the case?
3. By default, the dimensionality of HD vectors is set to 1,024. Is this dimensionality enough to obtain a decent accuracy? What happens if the dimensionality of HD vectors is reduced dramatically?

Part 2: Classification of numerical data.
The data in Part 1 was symbolic and so it was possible to use random HD vectors. However, this is not always the case. Therefore, in the second part of the assignment, we will work with a dataset that contains numerical data.

The dataset is called "Pen-Based Recognition of Handwritten Digits"; it is from the UCI machine learning repository. You may check the details of the dataset using this link. But as a starting point for this assignment, we provide you with a ".mat" file, which contains normalized features for the training and test splits and the corresponding labels.

Please download the file using the following link:
https://www.dropbox.com/s/jilmya8a9f8cywh/pendigits.mat?dl=0

For this part of the assignment, you are also provided with the Jupyter notebook (see course website) that includes the code to support loading data and the general logic of the process.

The provided Jupyter notebook is missing a few steps, which you will implement (search for ***#ToDo*** comment in the notebook). In particular, you would have to implement the following steps:

1. Form an item memory with a locality-preserving encoding (LPE) method of your choice (see Module 8's lecture slides for examples) and assign an LPE for a given feature value. Note that because of Step 2, you should pick a LPE compatible with a binding operation.
2. Form a compositional HD vector representing the set of key-value pairs, where the key is unique feature ID (random HD vector), and the value is the LPE of that feature's value. (For hints on how to form a set of key-value pairs, please review this week's lecture, or refer to the lecture slides in Module 4.)
3. Form centroids for each class, where a centroid is formed by component-wise addition of HD vectors belonging to the same class.

The second approach to classification we will explore is called Generalized Learning Vector Quantization (GLVQ). You would need to initialize GLVQ to the centroids obtained during the previous step. Since we have only briefly discussed GLVQ in lecture, we also recommend visiting the original paper on GLVQ – "Generalized Learning Vector Quantization" by Sato and Yamada. Essentially, GLVQ is an algorithm for iteratively revising the centroids to minimize the cost function estimated using the training data.

You will need to use the "sklearn_lvq" package. Please take a look at its documentation, as it will help you answer the questions below on the role of GLVQ's hyperparameters. Learning the classifier with GLVQ is largely set up in the notebook. Therefore, the main step that you would need to implement is the initialization of GLVQ prototypes to the centroids obtained in step 3 above.

Please perform the following experiments/answer the following questions:

4. Wrap the code with a "for loop" so that you can perform simulations for random initializations of "keyHVs". Compute the average accuracy over several random initializations for both centroids-based classifier and GLVQ classifier.
5. How do the accuracy of the centroids-based classifier and the accuracy of the GLVQ classifier compare to each other?
6. How does the choice of dimensionality of HD vectors affect the accuracy of both classifiers?
7. Does GLVQ seem to be a natural choice of classifier when using HD vectors? If so, why?
8. How does the choice of number of epochs in GLVQ affect the accuracy?
9. What is the role of "beta" in GLVQ? How does the choice of "beta" affect the accuracy?