CSC 202: Intro to Machine Intelligence
Machine Learning Project

Option #1: Machine Learning in Python

The last third of the course covered machine learning methods including k-nearest neighbors and neural networks, with a primary focus on image recognition tasks. For this option, you will implement one (or more) classification methods using a dataset of your choice. You will turn in a Jupyter Notebook that carries out the appropriate analysis. Your notebook should begin with your name and a one paragraph summary describing the main goals of your project (e.g., to classify images of animals as cats or dogs). Each code cell should be preceded by a markdown cell that describes what the code does (as in the class notes). Your Notebook should be correctly formatted with an appropriate title, section headers, and subsection headers. [20 points]

1. Find and load a dataset to analyze (required). Possible datasets include the following:

    a. The *wine* dataset, available from *sklearn* (*load_wine*) (this will be the easiest)
    See: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html

    b. The fashion-MNIST dataset, available from
    https://github.com/zalandoresearch/fashion-mnist

    See the Jupyter Notebook for how to load in these images.

    c. The CIFAR-10 dataset, available from
    https://www.cs.toronto.edu/~kriz/cifar.html

    See the Jupyter Notebook for how to load in these images.

    d. Loading from tensorflow (you will need to create a new environment and install *tensorflow* as described in the Python Installation notes. You must then select this environment for use with the Jupyter Notebook.

    The Fashion MNIST dataset can be loaded using the following code:

```
from tensorflow.keras.datasets import fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

    The CIFAR 10 dataset can be loaded from tensor flow using the following code:

```
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

When displaying the images, use a small size to get good resolution. Note that

these are color images, and therefore have 3 channels.

2. Dogs vs cats, https://www.kaggle.com/chetankv/dogs-cats-images. Note that these are color images, and therefore have 3 channels. This is also a *binary* classification problem, since there are only 2 classes. For neural networks, your final layer and loss function should be set appropriately (I can go over this with you). You will need to create an account to get this dataset, as well as learn how to read in image data in Python.

3. You may also analyze a dataset of your choice (a good source of image datasets is https://www.kaggle.com/datasets). But note that many datasets are very large (hundreds of GBs) and will be too large to use for this project. If you want help finding your own dataset, let me know!

Note: In all cases, you may choose to analyze 3 classes of images if there are more than 3 in the dataset). For data sets with a large number of images, you may also limit your training data to include 30,000 images total. Some (most) datasets contain images that are much more difficult to classify correctly than the MNIST dataset used in class. Your grade will not be based on classification accuracy, but you are encouraged to experiment with more complex structures (such as neural networks with multiple layers) than what we have used in order to get a good performance.

4. For both the training and testing sets, use Python to output the number of features and the number of samples, which documentation or output that clearly indicates which number is which. This should generally be accomplished by looking at the *shape* of the data. [5 points]

5. Using the *imshow* method, plot the first 30 images, using 3 rows and 10 images per row. For non-image datasets, construct two separate scatterplots showing the relationship between 2 different pairs of variables, with points color-coded by target values. [15 points]

6. Split the data into training and testing samples if not already done for you. Output the number of samples in each class for both the training and testing sets. Use Python to determine and output the number of unique target values that you have? [10 points]

7. Implement a *knn* classifier or neural network that predicts the target value from the feature data. [10 points]

8. For neural networks, generate a loss curve (showing either loss or accuracy in the training dataset and optionally in the validation dataset). For knn, evaluate the balanced accuracy using values of k = 1,2,5,and 7. Generate a scatterplot of the results, and comment on the optimal value of *k*. For neural networks, you may optionally try to

optimize your neural network with respect to the structure, number of layers, or number of epochs. Note that this process may be time-consuming. Neural networks take a long time to train, but once trained can quickly make predictions. Knn is trivial to train, but can take a long time to make predictions. If something seems to be taking too long, let me know. [10 points]

9. Make predictions in the test dataset, and generate the following, using class labels and not numeric ones [15 points]
    a. A Heatmap of the confusion matrix
    b. A Classification report that shows the recall and precision for each class

10. Create a list containing the test images (or features) and target values for those that were predicted correctly, and another list for all the test images that were not predicted correctly. For images, use *imshow* to plot at least 2 images that were predicted correctly (and their correct predictions), and at least 2 images that were not predicted correctly (and their incorrect predictions) The predicted values and actual values should be part of the title of each image. [15 points]

Option #2. Digital Assistant

Develop an IBM Watson Assistant and optionally demonstrate it using Python and the IBM cloud API. For example, an assistant can be developed that

- Allows the user to register for classes at Eastern
- Provides the user with information, such as
    - Movie information, including description, ratings, and showtimes
    - Song information, including artists, songs, albums, and release dates
    - Trivia, such as states and their capitols

If choosing this option, you will turn in the JSON file exported from the Assistant, and any additional Python code (in a Jupyter Notebook), if applicable. Your assistant will recognize *intents* and *entities* through training on examples and will respond appropriately based on these. (Your assistant will not rely on outside information, such as looking up movie times, as this is beyond the scope of this course).

1. Create a Watson Assistant that correctly carries out a task. Your assistant does not need to cover all possibilities but should work correctly for the entities included. For example, an assistant that provides information about Eastern Faculty should include at least 5 faculty members, and should work correctly for the intents included. If the user wants information about another faculty member, or requests information (has an intent) that

is not available, the assistant should respond appropriately. Your assistant should include

a. At least 5 intents with at least 5 user examples each [20 points]

b. At least 5 entities with appropriate values and synonyms. At least one entity must have at least 4 values. [20 points]

c. At least 3 context variables that are properly set and utilized. Recall that a context variable allows the assistant to remember what the user has said (for example, the faculty member the user is interested in) [20 points]

d. A dialogue that uses at least one system entity such as @sys-date or @sys-time (https://cloud.ibm.com/docs/services/assistant?topic=assistant-system-entities) [10 points]

e. A dialogue that correctly converses with the user, which includes a welcome message that describes what the assistant does [30 points]

Optional:

- Use the IBM cloud API to provide a demo for the assistant (https://cloud.ibm.com/apidocs/assistant/assistant-v2?code=python). Results are returned in JSON format, which can be converted to a dictionary.

- Use Python's *SpeechRecognition* module so a user can "talk" to the assistant using the computer's microphone.