Statistical Methods in Artificial Intelligence

Assignment 4

Deadline : 29 October 2024 11:55 P.M.

Instructor: Prof Ravi Kiran Sarvadevabhatla

October 14, 2024

1 General Instructions

- Your assignment must be implemented in Python.
- While you're allowed to use LLM services for assistance, you must explicitly declare in comments the prompts you used and indicate which parts of the code were generated with the help of LLM services.
- Plagiarism will only be taken into consideration for code that is not generated by LLM services. Any code generated with the assistance of LLM services should be considered as a resource, similar to using a textbook or online tutorial.
- The difficulty of your viva or assessment will be determined by the percentage of code in your assignment that is not attributed to LLM services. If during the viva you are unable to explain any part of the code, that code will be considered as plagiarized.
- Clearly label and organize your code, including comments that explain the purpose of each section and key steps in your implementation.
- Properly document your code and include explanations for any non-trivial algorithms or techniques you employ.
- Ensure that your files are well-structured, with headings, subheadings, and explanations as necessary.
- Your assignment will be evaluated not only based on correctness but also on the quality of code, the clarity of explanations, and the extent to which you've understood and applied the concepts covered in the course.
- Make sure to test your code thoroughly before submission to avoid any runtime errors or unexpected behavior.

- Submit your assignment sticking to the format mentioned below. Not doing so will result in penalization.

```
├── assignments
│   ├── 1
│   │   └── a1.py
│   ├── 2
│   │   └── a2.py
│   ├── 3
│   │   └── a3.py
│   ├── 4
│   │   ├── CNN.ipynb
│   │   ├── AutoEncoder.ipynb
│   │   ├── figures
│   │   └── README.md
│   └── 5
│       └── a5.py
├── data
│   ├── external
│   │   ├── linreg.csv
│   │   ├── regularisation.csv
│   │   ├── spotify-2
│   │   │   ├── test.csv
│   │   │   ├── train.csv
│   │   │   └── validate.csv
│   │   └── spotify.csv
│   ├── interim
│   │   ├── 1
│   │   ├── 2
│   │   ├── 3
│   │   ├── 4
│   │   ├── 5
│   │   └── README.md
│   └── README.md
├── models
│   ├── cnn
│   │   ├── cnn.py
│   │   └── multilabel-cnn.py
│   └── Autoencoders
│       ├── cnn-autoencoder.py
│       └── pca-autoencoder.py
├── performance-measures
│   └── README.md
└── README.md
```

  - The `data/external` folder contains the datasets being provided to

you. You should use the `data/interim` folder to store data that has been transformed and that is in use.

  - There are folders for each assignment inside the `data/interim` directory, and make sure to store data specific to each assignment accordingly.
  - Store implementations of the models in the `models` folder.
    * Contrary to what was mentioned in the tutorial, you are supposed to make `fit` and `predict` routines inside the specific model class instead of making separate train, test and eval files. These routines **should** be present in the specific model class and have the naming convention mentioned.
  - For particular tasks related to the assignments, do them inside the folder for that specific assignment and import classes and data from the other files.
  - `performance-measures/` should contain the generalized implementations of all evaluation metrics, in a way such that they can be used for any model as and when needed.
  - Each assignment folder has a `README.md` which you will have to modify to make the final report with all your observations, analyses, and graphs.
  - The figures folder has been moved into the assignment-specific directory. You should save all the generated plots, animations, etc inside that folder and then include them in the report.
  - Mention the references used for each assignment in the report.

- The deadline will not be extended.
- MOSS will be run on all submissions along with checking against online resources.
- We are aware of how easy it is to write code now in the presence of LLM services, but we strongly encourage you to write the code yourself.
- We are aware of the possibility of submitting the assignment late in GitHub classrooms using various hacks. Note that we will have measures in place accordingly and anyone caught attempting to do the same will be given a straight zero in the assignment.
- You are only allowed to use **PyTorch** for this assignment.
- ***Note***: For this assignment, you need to directly commit notebooks to the assignment 4 directory. You are still required to commit all models that you create, under the respective folders in the `model` directory. Please refer to the directory structure for further clarity.

2 Classification Using CNN

Convolutional Neural Networks (CNNs) are widely used for image-based tasks such as classification, object detection, and even image generation. CNNs have the ability to automatically capture spatial hierarchies and features from images.

For this task, you will implement a CNN to predict the number of digits in a given image using both classification and regression approaches. You will be using modified **Multi MNIST** dataset for the same.

2.1 Data Loading and Preprocessing

1. You have been provided with a ZIP file **double_mnist** for this task. This ZIP contains three main folders: **train**, **val**, and **test**. Within each of the three folders, there are subfolders, each named after a digit label (e.g., 1, 2, 123, etc.), and containing images that all represent the same digit(s). The subfolder **0** contains blank images with no digits. Explore the dataset to get more clarity.

2. Implement a function `load_mnist_data()` that extracts images from the dataset folders and organizes them into separate lists for images and labels corresponding to the train, validation, and test splits. Ensure that the images are loaded from their respective folders without any overlap or mixing.

3. Create a class called **MultiMNISTDataset** that will be used to create dataloaders for training and evaluation purposes.

*Note that since in this task your are trying to predict the **number of digits** in an image instead of the actual digits, the actual label for your current task would not be the subfolder name itself, rather the number of digits in subfolder name.*

2.2 Implement the CNN Class

Implement a single **CNN** class that can handle both classification and regression tasks.

1. The class should take a parameter `task` as input, which can take up two values: *classification* or *regression*.

2. The class should define the convolutional layers, activation functions, and pooling layers necessary for the architecture.

3. The `forward()` method takes input data and passes it through the defined layers sequentially. This method should handle the different output requirements for classification and regression tasks based on the value of the `task` parameter.

2.3 Hyperparameter Tuning

1. Identify the key hyperparameters to tune, including learning rate, dropout rate, the number of convolutional layers and optimizer choice.

2. Plot training and validation loss graphs for at least 5 different combinations of hyperparameters for each of the classification and regression models.

3. Identify the best perfoming model and include relevant plots and analyses in the report. Report accuracies of both classification and regression CNN on validation and test set.

2.4 Feature map Visualization

Feature maps reveal which patterns and features the CNN is focusing on at various layers. To obtain feature maps, you can access the outputs after each block of [convolution, activation function, and pooling layer] during the forward pass of the model.

1. Modify the `forward()` function to capture intermediate outputs after applying each convolutional block. Visualize feature maps of any **three** images.

2. Based on the output, try to infer the type of features being learned at different layers.

3 Multi Label Classification

Unlike traditional single-label classification, where each input is associated with only one label, multi-label classification involves predicting multiple labels for a single input. The goal of this task is to create a model capable of predicting all digits present in an image (while preserving the sequence of digits), where each digit in a sequence is treated as a separate label.

3.1 Data Loading and Preprocessing

1. For this task, modify the function `load_mnist_data()` to now use the subfolders' names as the actual labels to load train, val and test dataset for model training.

2. **One-Hot Encoding the labels**: Implement a function that creates one-hot encoded vectors for each label instead of using the label values directly. This is to allow the model to understand that each digit can be present independently in an image.

3. Reuse the **MultiMNISTDataset** class to create train, val and test loaders.

3.2 Implement the MultiLabelCNN Class

1. Define the convolutional layers, activation functions, pooling layers, and dropout layers necessary for the architecture.
2. Implement the `forward()` method that takes input data and passes it through the defined layers sequentially.
3. Would this model require an additional activation function after the last linear layer, unlike the previous CNN classifier? Why or why not? If yes, justify your choice of function in the report.

3.3 Hyperparameter Tuning

1. Experiment with different values of hyperparameters such as learning rate, number of epochs, dropout rate, the number of convolutional layers and optimizer choice.
2. Plot training and validation loss graphs for 5 or more such combinations of hyperparameters.
3. Identify the best perfoming model and include relevant plots and analyses in the report. For this model, report both the **exact match accuracy** and **hamming accuracy** on train, val and test data.

4 AutoEncoder: Comparative Analysis

4.1 Introduction

A Convolutional Neural Network (CNN) autoencoder is a deep learning architecture used to learn efficient representations of input data. It consists of two main parts: an **encoder**, which compresses the input data into a low-dimensional latent space, and a **decoder**, which reconstructs the original data from this compressed representation.

4.1.1 AutoEncoder Classification

In addition to unsupervised tasks, CNN autoencoders can be adapted for classification tasks. After the encoder compresses the data into a latent representation, a classifier can be added on top of the latent space to map the compressed features to corresponding class labels.

In this task, you will design and implement a CNN-based autoencoder which will be used to classify images from the **Fashion MNIST** dataset. The Fashion MNIST dataset consists of 28x28 grayscale images of 10 different classes of clothing.

4.2 CNN AutoEncoder

4.2.1 Dataset Analysis and Preprocessing

1. Load the Fashion MNIST dataset from kaggle. Split the training data into train, val and test in the ratio 70:10:20.
2. Visualize some images from the dataset to see different kinds of clothings and the labels assigned to them.
3. Try to list the clothing type represented by each class (don't google this, try it out yourself).

You will find the dataset here: https://www.kaggle.com/datasets/zalando-research/fashionmnist

4.2.2 Implement the CnnAutoencoder class

1. The class should consist of the `encoder` and `decoder` components and the functions `encode()`, `decode` and `forward()`.
2. The `encode()` function calls the `encoder`, which reduces the spatial dimensions of the input image.
3. The `decode()` function calls the `decoder`, which takes the compressed latent space vector and reconstructs it back into an image.
4. The `forward()` function governs the forward pass by calling the `encode()` and `decode()` functions on the input.

4.2.3 Plots and Visualizations

1. Visualize 10 images (original and reconstructed) from the test dataset to evaluate the quality of the autoencoder's output.
2. Create a plot of the encoded representations in the latent space to analyze how well different classes are clustered or separated. For this, you can make use of ***sklearn PCA*** to reduce the dimensionality for visualization. Plot both 2D and 3D representations. Mention any observations you can make.

4.2.4 Hyperparameter Tuning

1. Try to optimize the model performance by trying different values for parameters like learning rate, kernel size, number of filters in each layer and optimizer choice.
2. Using **MSE loss** as the criterion, plot train and validation loss graphs over the epochs for at least 3 such combinations of hyperparameters. Identify the best performing model.

3. Keeping the latent space dimension constant, experiment with varying the number of layers you are using. Compare loss graphs across 3 different layer configurations (while keeping all other parameters constant) to analyze the effect of depth on model performance.

4.3 MLP Autoencoder

In this section, you will be using the MLP autoencoder that you implemented in Assignment 3 to classify the images from **FashionMNIST** dataset.

1. Train your MLP classifier using the train and validation datasets. Plot loss curves over the epochs. Compare the model performance with the CNN autoencoder model.

2. Visualize the same set of images (original and reconstructed) from the test dataset using MLP autoencoder, as previously analyzed for the CNN autoencoder. Compare the results obtained from the two.

4.4 PCA Autoencoder

In this section, you will implement a Principal Component Analysis (PCA) - based autoencoder to compress and reconstruct image data by reducing the dimensionality.

4.4.1 Implement the PcaAutoencoder Class

1. The class should contain the `encode()`, `forward()` and `fit()` functions.

2. The `fit()` function calculates eigenvalues and eigenvectors from the input data.

3. The `encode()` function reduces the dimensionality of the input data using the learned eigenvectors.

4. The `forward()` function returns the reconstructed data.

4.4.2 Estimate optimal number of components

1. Train the model and evaluate the reconstruction error by calculating the MSE loss between the original and reconstructed images from **validation** dataset. Estimate and report the optimal number of components using the **elbow plot** of Reconstruction error v/s Number of Components. We'll call this $k_{optimal}$.

2. Visualize the original and reconstructed images from the test data obtained on running PCA autoencoder with $k_{optimal}$ number of components. Compare results with those obtained using the CNN and MLP autoencoders.

4.5 KNN Classification

In this task, you will use the **encoded representations** (latent features) obtained from the encoder component of the three autoencoders implemented above. You will evaluate how well these representations classify the data by applying the K-Nearest Neighbors algorithm. Use the best performing models for all three autoencoders, that you obtained above after tuning parameters on the validation set.

1. Obtain the latent features from encoder component for each of the 3 autoencoders, for both the training and test data. Ensure that all the 3 models' reduced dataset are of $k_{optimal}$ (from 3.4.1) dimensions for fair comparison and that the extracted features are reshaped for compatibility with KNN classification.

2. Train the KNN classifier that you implemented in Assignment 1, using the extracted latent features of train set.

Note: *You need to fully vectorize your code for KNN if you did not do it for Assignment 1* ☺

4.5.1 Performance Comparison

1. Evaluate performance by predicting the labels for the test set and calculating the accuracy. Plot the accuracies on train and test for all three models.

2. Plot confusion matrices for the three models (CNN, MLP, PCA) using actual test labels and the predictions obtained from KNN classifier. Are there are any classes for which all three autoencoders give similar performance? Mention observartions and analyses in the report.

Good luck with the assignment!