Statistical Methods in Artificial Intelligence

Assignment 5

Deadline : 13 November 2024 11:55 P.M

Instructor: Prof Ravi Kiran Sarvadevabhatla

November 3, 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
│   │   ├── figures
│   │   │   └── README.md
│   │   └── README.md
│   ├── 2
│   │   └── a2.py
│   ├── 3
│   │   └── a3.py
│   ├── 4
│   │   └── a4.py
│   └── 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
│   ├── knn
│   │   └── knn.py
│   ├── linear-regression
│   │   └── linear-regression.py
│   └── README.md
├── 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.

2 KDE

In this section, you will implement the Kernel Denisty Estimation (KDE) algorithm from scratch for n-dimensional data and compare its performance against Gaussian Mixture Models (GMM) on a synthetic dataset.

2.1 Task 1 : KDE Class (60)

Implement a KDE for n-dimensional data, you are not allowed external libraries such as `scikit-learn` for this. Your class must contain,

- Hyperparameter for kernels, box, Gaussian, triangular.
- `fit` data function.
- `predict` function, which returns the density at any given point x.
- Visualize function that plots the density against the data for 2D data.

2.2 Task 2 : Generate Synthetic Data (10)

Construct the dataset in Figure 1, it does not need to be exact. We generated points uniformly and randomly within circles with some noise to simulate variance. The larger diffused circle contains 3000 samples, while the smaller dense circle contains 500 points.

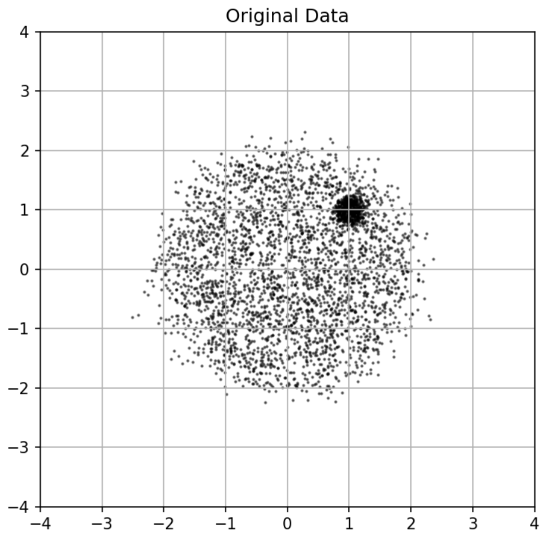

Figure 1: Synthetic dataset with extreme overlap between two regions of density.

2.3 Task 3 : KDE vs GMM (30)

Fit both a KDE model, and a GMM model on the above dataset.
When we use two components for the GMM model, what do you observe? What happens to the GMM model as the number of components increase? Does the KDE model consistently fit the data?

3 HMMs

3.1 Speech Digit Recognition

In this task, you will build a Hidden Markov Model (HMM) to recognize spoken digits from audio signals using the Free Spoken Digit Dataset.

3.2 Dataset (25)

The Free Spoken Digit Dataset consists of audio recordings of digits 0-9, with 300 recordings per digit, totaling 3000 recordings. Download the dataset from https://www.kaggle.com/datasets/joserzapata/free-spoken-digit-dataset-fsdd/data.
For feature extraction:

- Use the *librosa* library to extract Mel-Frequency Cepstral Coefficients (MFCC) from the .wav files. https://librosa.org/doc/latest/index.html
- These MFCC features will serve as inputs to train the HMM models.
- Visualize the MFCC features as a heatmap or spectrogram, and mention any patterns visible in the audio signals. Can these patterns explain why HMM is a suitable method for this task?

3.3 Model Architecture (45)

For this task, use the *hmmlearn* https://hmmlearn.readthedocs.io/en/latest/ library to implement the HMM architecture:

- Train separate HMM models for each of the 10 digits using their respective MFCC features.
- For prediction, evaluate all HMM models on the test sample and select the digit corresponding to the model with the highest probability.

3.4 Metrics (30)

Use recognition accuracy as the evaluation metric, measuring the percentage of correctly classified digits in the test set. To test generalization:

- Evaluate performance on the provided test set
- Record and test your own audio clips for each digit
- Compare the model's performance on both known and personal recordings

4 RNNs

4.1 Counting Bits

In this task, you will be training an RNN that learns to count the number of 1 bits in an input bit-stream.

4.1.1 Task 1 : Dataset (10)

Construct a dataset of binary sequences. Generate 100k sequences of random bits with lengths varying from 1 to 16 for training. Each sequence is paired with a label indicating the count of '1's in the sequence. This gives 100k (sequence, count) pairs. Report examples.
Generate an appropriate training, validation, and test split with ratios $(0.8, 0.1, 0.1)$.

4.1.2 Task 2 : Architecture (30)

For this task, use a character-level RNN architecture. The RNN will process the input sequence of bits one character at a time, learning to accumulate the count of '1's. You are allowed to use PyTorch pre-defined RNN layers. No other library such as TensorFlow or Keras is accepted.
Experiment with different layers and different hidden states to figure out what works. You can experiment with techniques like dropout and normalization.

4.1.3 Task 3 : Training (40)

Implement a basic training loop that evaluates the model on a validation set after every epoch. Ensure that the validation loop has appropriate metrics.

Metrics Use *Mean Absolute Error (MAE)* as the evaluation metric, which will measure the average absolute difference between the predicted count and the actual count across samples. Compare this with a random baseline that outputs a count within the range of possible values for each sequence length.

4.1.4 Task 4 : Generalization (20)

In this task, we want to study how well our trained model generalizes to out of distribution data. Generate 1000 samples of sequence length 17, 18, etc from the dataset generator, then run the evaluation loop for this data set with the trained model.
Plot the metrics with input sequence length from 1 to 32 (x-axis is the sequence length, y-axis is the MAE for that sequence length) and explain the results.

4.2 Optical Character Recognition

In this task, you will be training an RNN to recognize words within an image.

4.2.1 Task 1 : Dataset (20)

You will construct the dataset from scratch. Collect 100k words (`nltk` can help with this, `from nltk.corpus import words`) and render these words onto images of size 256x64, see Figure 2 for an example. You now have 100k image-word pairs which is your dataset.

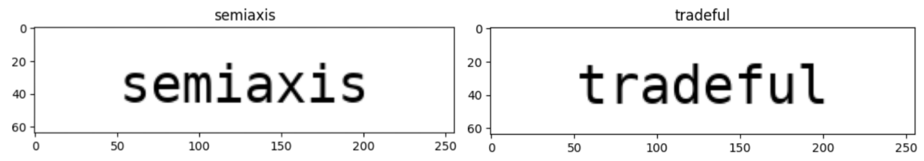

Figure 2: Dataset examples for the words "semiaxis", and "tradeful". The words are being rendered onto a 256x64 plain white image.

4.2.2 Task 2 : Architecture (40)

For this task, you will use a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs):

- CNN: The CNN will extract spatial features from the image, which helps identify visual patterns corresponding to characters.
- RNN: The extracted features are then fed into a character-level RNN, which outputs a sequence of characters representing the word in the image.

The architecture consists of the encoder-decoder paradigm. The encoder is the CNN, and the RNN decoders the characters one by one sequentially. Experiment with different hyper-parameters to figure out what works. Experiment with dropout and batch normalization and other regularization techniques.

4.2.3 Task 3 : Training (40)

Implement a basic training loop that evaluates the model on a validation set after every epoch. Ensure that the validation loop has appropriate metrics. In addition to the metrics, provide some examples of ground truth versus model prediction in the report of your final trained model.

Metrics You'll use *Average Number of Correct Characters* as the evaluation metric, measuring how many predicted characters match the target in value and position, averaged across samples. To assess progress, compare this with a random baseline—randomly generated character sequences of the same length as the target words.

Good luck with the assignment!