Statistical Methods in Artificial Intelligence

Assignment 1

Deadline : 24 August 2024 11:55 P.M

Instructor: Prof Ravi Kiran Sarvadevabhatla

August 11, 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 K-Nearest Neighbours

2.1 Music Genre Prediction

The Spotify dataset to be used can be found in the GitHub repository or here. It is in the form of a `csv` file where each row corresponds to a datapoint and the number of rows is equivalent to the number of examples in the dataset.
You will be using the other features in this dataset to predict the genres of the respective songs.

2.2 Exploratory Data Analysis

2.2.1 Task 1

Generate suitable plots that show distributions of various features in the dataset. You are allowed to use standard libraries like Matplotlib.
Write comments about what you observe (for example, outliers, skewed data, etc)
Also, try to visualize different combinations of features to try and understand the correlation of the features with the target variable. Doing so will be useful for some of the upcoming tasks.
Try coming up with a hierarchy of which columns are more necessary than others for the classification task based on the data exploration. Mention all these results in the report.

2.3 KNN Implementation

You must implement the KNN algorithm from scratch using Python classes in this section. This approach encapsulates the KNN model and its methods within a single class, enhancing code readability, reusability, and maintainability. You can easily manage hyperparameters, data preprocessing, and result interpretation by designing the class.

2.3.1 Task 2

Create a KNN class where you implement the following: You should not use sklearn for this.

1. Create a class where you can modify and access `k` and the distance metric (and any required parameter) of the class
2. Return the inference (prediction) when given the above parameters (`k` and distance metric).
3. Return the validation f-1 score, accuracy, precision, and recall after splitting the provided dataset into train and val subsets. You are **not** allowed to use sklearn metrics for this part, you must make a separate class that calculates the metrics (accuracy, macro and micro for f-1, precision and recall scores).

2.4 Hyperparameter Tuning

2.4.1 Task 3

Tasks:

1. Find the best {k, distance metric} pair that gives the best validation accuracy for an 80:10:10 split (train:test:validation).
2. Print an ordered rank list of the top 10 such pairs.
3. Plot k vs accuracy given a choice (yours) of any given {`k`, distance metric} pair with a constant data split.
4. More data need not necessarily mean best results. Try dropping various columns and check if you get better accuracy. Document the combination with which you get the best results.
5. [**Bonus**] Try for all combinations of columns and see which combination gives the best result. See if that matches with the analysis made during data exploration.

2.5 Optimization

2.5.1 Tasks

1. Improve the execution time of the program using **vectorization**. Make sure it is n time complexity.
2. Plot inference time for initial KNN model, best KNN model, most optimized KNN model, and the default sklearn KNN model.
3. Plot the inference time vs train dataset size for initial KNN model, best KNN model, most optimized KNN model, and the default sklearn KNN model. Write down observations that you make based on this graph.

2.6 A Second Dataset

There is another dataset given in `data/external/spotify-2`, which is already split into `train.csv`, `validate.csv` and `test.csv`. Using the best {`k`, distance metric} pair you got previously, apply KNN on this data and state your observations on the data and the performance of this data.

3 Linear Regression

Linear Regression is a supervised machine learning algorithm that fits a linear equation to the observed data.
The equation for multiple linear regression is of the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon,$$

where Y is the dependent variable, X_j is the j^{th} predictor and β_j quantifies the association between the dependant and the independent variable. The ϵ term is the error term, which is independent of X and has mean zero.
Implement this part using **numpy only**. Additionally, do not use any library functions that directly perform model fitting like `np.polyfit`. Just as instructed for KNN, you need to make a class for Linear Regression.
Regularization is also a part of this assignment, while making the class for simple regression, add a parameter which has $\lambda = 0$ which you can modify to another value for regularisation. Note that you cannot create separate classes for regression and regularisation.

3.1 Simple Regression

The file **linreg.csv** contains 400 points sampled from a polynomial.
The first column, x is the independent variable, and the second column has the dependent variable. Shuffle the data and make an 80:10:10 split into train, validation and test. Report metrics for all three sets. For both the tasks, experiment with different learning rates and use the best one. Visualize all the splits in different colors in one graph.

3.1.1 Degree 1

Fit a line to the curve ($y = \beta_1 x + \beta_0$).
Report the MSE, standard deviation and variance metrics on train and test splits. Plot the training points with the fitted line.

3.1.2 Degree > 1

Fit a polynomial to the curve ($y = \sum_0^k \beta_k x^k$).
Make a class for regression that can take the value of k as a parameter. Test it with various values of k and report the MSE, standard deviation and variance metrics on train and test splits for all the values. Additionally, report the k that minimizes the error on the test set.
Save the parameters of the best model to a file so that the coefficients can be loaded for testing.

3.1.3 Animation

For each iteration, plot the original data along with the line you are fitting to it, as well as the MSE, standard deviation and variance. Instead of plotting each

of these graphs separately, save it as a GIF (in the figures directory) visualizing how you gradually reached convergence. (Do this for atleast 5 values of k).
The GIF should have 4 cells, 3 with metrics and one with the line gradually fitting to the curve.
Additionally, random initialization will affect the results. Experiment with various seeds to see how performance is affected and how long it takes to converge.
Converting images to GIFs: Link
Example: The right half of this GIF represents the curve fitting part of the animation (first cell),

3.2 Regularization

The file **regularization.csv** contains 300 points sampled from a 5-degree polynomial. The first column contains the independent variable and the second the dependent. Shuffle the data and make an 80:10:10 split to work on. Visualize the training dataset.

3.2.1 Tasks

Try fitting higher degree polynomials to the data (go up to 20). This will result in overfitting. Plot the data points and the resulting curves and report the MSE, standard deviation and variance for each.
Use regularization to reduce the overfitting and plot the resulting curves again. Try both L1 and L2 regularization and compare the results. Add to the regression class that takes the type of regularization as a parameter.
Write down any observations that you have in the report.

Good luck with the assignment!