INF8245E(Fall 2024) : Machine Learning — Given on: Sep 25^{th}

Assignment #2 — Due on : Oct 25th, 10 pm

- This is an individual assignment. You are not allowed to discuss the problems with other students.
- Make sure to write your code only in the .py files provided. Avoid creating new files. Do not rename the files or functions as it will interfere with the autograder's ability to evaluate your work correctly. Also, do not change the input or output structure of the functions.
- When Submitting to GradeScope, be sure to submit
  1. A '.zip' file containing all your python codes (in .py format) to the 'Assignment 2 - Code' section on Gradescope.
  2. A 'pdf' file that contains your answers to the questions to the 'Assignment 2 - Report' entry.
- Part of this assignment will be auto-graded by Gradescope. You can use it as immediate feedback to improve your answers. You can resubmit as many times as you want. We provide some tests that you can use to check that your code will be executed properly on Gradescope. These are **not** meant to check the correctness of your answers. We encourage you to write your own tests for this.
- Please provide the title and the labels of the axes for all the figures your are plotting.
- If you have questions regarding the assignment, you can ask for clarifications in Piazza. You should use the corresponding tag for this assignment.
- Before starting the assignment, make sure that you have downloaded all the data and tests related for the assignment and put them in the appropriate locations.
- **Warning:** Throughout the assignment, you will be asked to implement certain algorithms and find optimal values. In the solution you submit, do not simply call a library function which performs the entire algorithm for you, this is forbidden, as it would obviously defeat the purpose. For example, if you were asked to implement the linear regression, do not simply call an outside package (such as scikit-learn) for help.
- You cannot use ChatGPT or any other code assistance tool for the programming part, however if you use ChatGPT to edit grammar in your report, you have to explicitly state it in the report.

In this assignment, you would implement various standard classification algorithms and test them on an image classification dataset, EMNIST. EMNIST is an extension of the standard hand-written digit classification dataset, MNIST, and you would be using the 'Letters' split of the dataset, which has 26 classes corresponding to each letter in the English alphabet. We select the first 15 letters for simplicity for this assignment. You would be implementing K-Nearest Neighbors, Gaussian Naive Bayes, and Logistic Regression in this assignment. The code to use the dataset is provided in each of the corresponding .py files. You have been provided with necessary docstrings for each function for reference.

Important:

1. Please download the necessary dataset and code at https://drive.google.com/file/d/1NdWV8sRQ4gr-oVrXJTobcUHOHJT1k_I8/view?usp=sharing.

2. When uploading your solutions, only compress the **gnb.py**, **knn.py**, and **logistic_regression.py** files to create the zip. **Do not** compress and upload any other python files or the dataset.

3. Please also go through the README file for instructions in setting up the environment for this assignment. You are not allowed to import any additional libraries as all the necessary libraries have already been imported in the scripts.

4. You have also been provided with a helper colab notebook to test the codes on Google Colab. However, this file is only to ensure your code runs without failing, and you are advised to modify the notebook as required to ensure further correctness of your work.

Link to notebook:
https://colab.research.google.com/drive/1DReUELyXF_X4nTyuADV0B1Go1CkYMphx?usp=sharing
In all the parts of the question, it has been mentioned if it is a "[Code]" question (to be done in the python files) or a "[Report]" question (to be written in the pdf report) for clarity.

K-Nearest Neighbors (30 Points)

In this question, you will implement the KNN algorithm to classify the characters of the EMNIST Letters split into 15 classes. In question 1, you will implement the functions required to perform classification with K-Nearest Neighbors. In question 2, you would utilize the functions to run the KNN algorithm and evaluate its performance on the test split of the dataset. Refer to the file **knn.py** for this question.

1. (16 Points) K-nearest neighbor utility functions implementation.
   (a) (2 Points [Code]) Feature scaling: Implement the `preprocess_data` function to perform feature scaling on the original data such that all values are in the range of [0, 1] and then vectorize the data, i.e. convert the 2D images into a 1D vector form. Please note that you would also be using this function when you implement Gaussian Naive Bayes and Logistic Regression. Though there are numerous ways

to do feature scaling, we use max normalization. Max normalization is applied on all the instances in the dataset as,

$$x_{norm} = \frac{x}{x_{\max}} \tag{1}$$

, where $x_{\max}$ is the maximum value in the dataset. Though this information is mentioned in the docstring of the function, note that we only use the 'max' of the training dataset. Hence, when normalizing the train set, observe that we have passed max=None, in which case it is to be calculated from the dataset. However, for the test set, we pass the 'max' value obtained previously from the train set, and hence, it is not calculated again. Refer to the `__main__` call for reference. Ensure you implement this check in the function.

(b) (2 Points [Code]) Euclidean distance: Implement the `euclidean_distance` function to calculate the Euclidean distance.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2}$$

(c) (8 Points [Code]) Get nearest neighbor labels: Implement the `get_k_nearest_neighbors` function to get the labels of the k-nearest neighbors of the training set.

(d) (4 Points [Code]) Get the prediction: Implement the `get_class_prediction` function which finds the most frequently occurring label from the nearest neighbors, i.e. the class prediction.

2. (6 Points) Using the utility functions you have implemented, we have provided the kNN algorithm in the `knn_classifier` function. This function returns the accuracy of the classifier on the validation or test set.

   (a) (3 Points [Report]) Use the validation set to find the best value of k which gives the best validation performance, by running the above classifier function with different values of $k \in \{2, 3, 4, 5\}$. We have already filled some dummy values of k in the `__main__` method. You can get the performance accuracy by simply executing the `knn.py` file. Write the different validation set accuracies obtained for each value of k and the best test accuracy obtained with the best value of k in the report.

   (b) (3 Points [Report]) When experimenting with different values of k, discuss the trends observed on the validation set performance. Plot the performance on the validation set when $k \in \{2, 3, 4, 5\}$ in the report, with the accuracy on the y-axis and value of k on the x-axis. This kind of experimentation to study the effect of crucial hyperparameters on the performance is generally called an 'ablation'. You would work with 'hyperparameter tuning' more in the next assignment :)

3. (8 Points) K-Nearest Neighbors requires calculating the distance between different instances in the dataset to get the majority label. The most standard distance used is

Euclidean distance which you implemented above. However, there can be other measurements for distance as well, such as Manhattan distance, Minkowski distance which is a generalization of Euclidean and Manhattan distance, and cosine distance. In this question, you would first implement cosine distance, and then use it in place of Euclidean distance to run KNN.

(a) (4 Points [Code]) Implement the `cosine_distance` function in knn.py which takes in two **vectorized** images, i.e. 1-dimensional flattened vectors obtained after preprocessing and calculates the cosine distance. The cosine distance d_{cosine} between vectors $\mathbf{x}$ and $\mathbf{y}$ is given as,

$$d_{\text{cosine}}(\mathbf{x}, \mathbf{y}) = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{3}$$

(b) (4 Points [Report]) Perform the same validation with $k \in \{2, 3\}$ only and plot the validation set performance with changing values of k in the report. Use the best value of k and measure the accuracy on the test set and present it in the report. You can uncomment the last part of the code in the `__main__` method to run KNN with cosine distance instead of Euclidean distance for this question. Compare the trends and performance obtained when using Euclidean distance with the performance obtained when using cosine distance. Why do you think this trend might be there?

Gaussian Naive Bayes (15 Points)

In this section, we will implement the Gaussian Naive Bayes (GNB) classifier for the EMNIST dataset. For this, you would be primarily working with the **gnb.py** file. We would still use the dataset preprocessing function from the previous **knn.py** file.

The GNB classifier belongs to the family of probabilistic classifiers based on the application of Bayes' Theorem. The term "naive" in naive Bayes classifiers comes from the fact that they have a strong independence assumptions between the features. In particular, it assumes that the value of a particular feature is independent of the value of another feature, *given the class variable.*

Consider the training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)}))\}$ of N labeled examples, and the input features are $x^{(i)} \in \mathbb{R}^n$. Since we are interested in multi-class classification, the label y can take K different values i.e. $y^{(i)} \in \{1, 2, \ldots, K\}$. The GNB model assumes that the **class-conditional densities** are distributed according to a multi-variate Gaussian distribution. In other words, the probability of observing the data $x^{(i)}$ given the class variable (also known as, *likelihood*) is given by a Gaussian distribution as shown below:

$$P(x \mid y = k, \mu_k, \Sigma_k) = \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

where μ_k denotes the class-specific mean vector and Σ_k denotes the class-specific covariance matrix (meaning that each class has its own mean vector and the covariance matrix). Note that since we have a separate covariance matrix for each class k, the covariance matrices are not shared among all the classes.

Given the likelihood of the model, we can now calculate the posterior probability, that is, the probability of a label belonging to a particular class given the data $x^{(i)}$, using Bayes' theorem as follows:

$$P(y = k \mid x, \mu_k, \Sigma_k) = \frac{P(x \mid y = k, \mu_k, \Sigma_k)P(y = k)}{\sum_{c=1}^{K} P(x \mid y = c, \mu_c, \Sigma_c)P(y = c)}$$

where $P(y = k)$ denotes the prior probability of a label belonging to a particular class. The denominator is essentially a normalization constant and is not technically required to be implemented. Observe that it this likelihood $P(x \mid y = k, \mu_k, \Sigma_k)$ that is distributed according to a multi-variate Gaussian distribution given below:

$$P(x \mid y = k, \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{K/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)\right)$$

where $|\Sigma_k|$ denotes the determinant of the covariance matrix, and K denotes the number of classes. An important note here is that the probabilities are small and in the case of high dimensionality they tend to be very close to zero and result in numerical underflow issues. Therefore, we will be considering the *log* of the likelihood function instead. Hence, the resulting log-likelihood can be written as:

$$\log P(x \mid y = k, \mu_k, \Sigma_k) = -\frac{K}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma_k|) - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)$$

Likewise, the log-posterior can be written as:

$$\log P(y = k \mid x, \mu_k, \Sigma_k) \propto \log P(x \mid y = k, \mu_k, \Sigma_k) + \log P(y = k) - \log(\text{const.})$$

Note that we have used the $\propto$ symbol above which indicates that the log of the posterior probability density for a class can be computed as the sum of the log-likelihood and the log prior probability densities, *upto* the log of the normalization constant.

Now, given the theory, there are two implementation questions in this section. In the first, you are asked to compute the class-specific mean and covariance vectors and the prior probabilities for each class. In the second question, you are required to use these 3 quantities to calculate the posterior probability of each class given the data. While the math and notations given above maybe a bit overwhelming, you are not required to implement the multi-variate Gaussian function. You can use SciPy's function for the same, more details are given in the question.

1. (9 Points) Similar to above, in this part, you will implement the utility functions for running the GNB algorithm.
   (a) (5 Points [Code]) GNB classifier: Implement the `gnb_fit_classifier` function that fits the GNB classifier on the training data.
   (b) (4 Points [Code]) GNB predict: Implement the `gnb_predict` function to run predictions on a given set of images.

2. (6 Points [Code, Report]) Using the functions above, complete the `gnb_classify` function to output the classification accuracies when using GNB. Use the GNB classifier to obtain the prior probabilities, means, and variances, and use the predict function to get the predictions. The labels passed in this function are used to calculate the accuracy. By running the gnb.py file, report the validation and test set accuracies in the report.

Logistic Regression (50 Points)

In this segment, you would extend logistic regression to multi-class classification and evaluate it on the EMNIST dataset. You would be working with the **logistic_regression.py** file. We would still use the dataset preprocessing and normalization functions from the previous **knn.py** file. This question would also introduce you to the more standard phases of a standard machine learning training regime: training, validation, and testing.

Consider a logistic regression model for classifying the EMNIST categories, where we have a training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)}))\}$ of N labeled examples, and the input features are $x^{(i)} \in \mathbb{R}^n$.

Since we are interested in multi-class classification, the label y can take K different values i.e. $y^{(i)} \in \{1, 2, \ldots, K\}$. Note that for ease of notation, we start the index of classes from 1, rather than from 0.

Now, given a test input $x^{(i)}$, we want our hypothesis to estimate the probability that $P(y = k|x^{(i)})$ for each value of $k = 1, \ldots, K$, i.e. we want to estimate the probability of the class label taking on each of the K different possible values. Thus, our hypothesis will output a $K-$ dimensional vector (whose elements sum to 1) giving us our K estimated probabilities. Concretely, the hypothesis function (denoted by z) for a single input $x^{(i)}$ takes the following form:

$$z_{(w,b)}(x^{(i)}) = \begin{bmatrix} P(y = 1 \mid x^{(i)}; w, b) \\ P(y = 2 \mid x^{(i)}; w, b) \\ \vdots \\ P(y = K \mid x^{(i)}; w, b) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(h_j^{(i)})} \begin{bmatrix} \exp(h_1^{(i)}) \\ \exp(h_2^{(i)}) \\ \vdots \\ \exp(h_K^{(i)})) \end{bmatrix},$$

where $h^{(i)} = x^{(i)}.w + b$ containing the parameters of the model. Particularly, $w \in \mathbb{R}^{n \times K}$ denotes the weight matrix, $b \in \mathbb{R}^K$ is the bias vector associated with each of the classes. Lastly, notice in the hypothesis function that we have a term of the form $\frac{\exp(\cdot)}{\sum_j \exp(\cdot)}$, this is called the softmax function and is used frequently in machine learning for multi-class classification problems since it outputs values as probabilities between 0 and 1.

We will use the negative log-likelihood loss for training our logistic regression model. As the name suggests, it simply calculates the negative of the log likelihood of the model and is given by:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log \hat{y}^{(i)} = -\frac{1}{N} \sum_{i=1}^{N} \log \left(\frac{e^{h_y^{(i)}}}{\sum_{j=1}^{K} e^{h_j^{(i)}}} \right)$$

1. (30 Points) We will derive the gradients for performing logistic regression for a multi-class classification problem and implement the utility functions required from training a classifier.

   (a) (10 Points [Report]) Calculate the derivative of the loss function L w.r.t the weight matrix w and the bias vector b. In other words, get the gradient update expressions for w i.e. $\frac{\partial L}{\partial w}$ and b i.e. $\frac{\partial L}{\partial b}$. Make sure the shapes of all matrices involved are consistent as given in the description above.

   Hints:

   1. It might be good to start by calculating the derivative of the softmax function.
   2. Think about how you can get the derivative of a particular quantity when it is inside a summation. Is there a way to divide the term into sub-terms and then computing the individual gradients?

   (b) (5 Points [Code]) Softmax function: Implement the `softmax` function in the logistic_regression.py file. Recall that the softmax function is given by,

   $$\text{softmax}(x) = \frac{\exp(x)}{\sum_j \exp(x_j)}$$

   (c) (5 Points [Code]) y_hat computation: In the given skeleton of the `LogisticRegression` class in logistic_regression.py file, implement the `__call__` method which computes the output class probabilities given the batch of inputs. **Do not forget to use the softmax function here before returning the output probabilities.**

   (d) (4 Points [Code]) Loss function: Implement the `nll_loss` function which computes the loss given the target labels and output probability predictions. Recall that the negative log likelihood is given as,

   $$L = -\frac{1}{N}\sum_{i=1}^{N} \log \hat{y}^{(i)} \tag{4}$$

   (e) (6 Points [Code]) Gradient computation: Using the gradients derived in **q1.a** and the helper functions above, implement the `compute_gradients` function to compute the gradient of the loss w.r.t the parameters.

2. (20 Points) Using the utility functions you have implemented in **q1**, train a logistic regression classifier for the EMNIST dataset using a standard machine learning regime.

   (a) (5 Points [Code]) Validation: Validation is one of the most important phases in training machine learning models. This is done so as to evaluate the learning capability of the model by running it on the samples from the validation set. The validation performance tells how well the model is 'generalizing' on the dataset, and not 'overfitting' to the training set. The procedure is as follows: Given the model and the batch size, iterate through the validation set to compute the loss and accuracy of the model. Note that in the validation phase, we do not compute the gradients. Implement the `validation` function to evaluate the performance of the model on the validation set.

(b) (5 Points [Code]) Training loop: Implement the `train_one_epoch` function that uses the `nll_loss` and `compute_gradients` with the `LogisticRegressionModel` class to run one full epoch of training. The function also performs `validation` after every fixed number of steps during the training.

(c) (6 Points [Report]) Do the training: Run the `logistic_regression` file which runs for 4 epochs and trains the model. Running this training has already been implemented in the `__main__` call. At the end, you would obtain plots for training and validation performance. Play with different learning rates to obtain the best setting (hint: the best learning rate can be one of $(0.05, 0.1, 0.5)$). Add the plots for the different learning rates along with mentioning the validation accuracy to the report and discuss them.

(d) (4 Points [Report]) Test and plot: Run the `test_model` function to obtain plots for the test loss and accuracy using your best learning rate from above. Add the plot and the test set performance in your report and comment on your findings.

(5 Points [Report]) Comparison of all the methods

Compare the performance of all the three classification methods on the test set that you have implemented in one table and comment about your findings in the report. Which method performs the best?