מטלת מנחה (ממ"ן) 11

הקורס: 20942 – מבוא ללמידה חישובית

חומר הלימוד למטלה: פרספטרון, רגרסיה לינארית, רגרסיה לוגיסטית

משקל המטלה: 20 נקודות

סמסטר: 2024א **מועד אחרון להגשה**: 10/01/2024

Handwritten Digit Classification:
A Practical Exploration with the MNIST Dataset

Introduction

The purpose of this assignment is to provide hands-on experience in applying machine learning concepts and techniques to solve a practical problem. The task at hand is digit classification using the MNIST dataset. The primary goals include implementing, training, and evaluating various models for accurate digit recognition.

Guidelines for Implementation

- **Programming Language**: Use Python exclusively for implementing your machine learning models.

- **Model Implementation**: Avoid using pre-built machine learning models or functions for the core tasks of your assignment. Implement the perceptron, logistic regression and linear regression algorithms from scratch, without relying on existing models.

- **Running Time**: Ensure that the overall running time of your machine learning algorithms is reasonable, completing within a few minutes for the entire process, including data loading, preprocessing, model training, and evaluation.

- **Vectorized Implementation**: Strive for a vectorized implementation of your algorithms. Minimize the use of unnecessary 'for' loops and leverage vectorized operations provided by NumPy for enhanced computational efficiency and runtime.

- **Documentation**: Document your code comprehensively. Include comments to explain key sections, functions, and any complex operations, ensuring that your implementation is understandable and can be followed by others.

Submission Requirements

Ensure that your submission includes the following components:

1. A well-documented Jupyter notebook.

2. Any additional files necessary for the execution of your code.

3. A written report summarizing your approach, results, and conclusions.

Introduction to Digit Classification and MNIST Dataset

The objective of this assignment is to develop models capable of accurately identifying handwritten digits. Digit classification is foundational for various applications such as postal code sorting, bank check processing, and automated form filling. It also plays a crucial role in advancing machine learning techniques and artificial intelligence.

The MNIST dataset serves as a cornerstone in the domain of digit classification. It comprises a collection of 28x28 pixel grayscale images, each depicting a handwritten digit (0 through 9). Originating from the National Institute of Standards and Technology (NIST), the MNIST dataset has become a benchmark for assessing the efficacy of various machine learning algorithms.
The following figure depicts 10 examples from the dataset:

The corresponding labels are: [1,9,6,5,8,1,4,4,1,8].

Key Characteristics of MNIST:

- **Image Size**: Each image is 28 pixels in height and 28 pixels in width, resulting in a total of 784 pixels per image.

- **Grayscale Images**: The images are grayscale, with each pixel represented by a single intensity value ranging from 0 (black) to 255 (white).

- **Handwritten Digits**: The dataset primarily consists of handwritten digits, introducing variations in writing styles, slants, and sizes.

- **Labeling**: Each image is associated with a label (digit from 0 to 9), creating a ten-class classification problem.

Data Partitioning

Divide the original MNIST dataset into two subsets: a training set containing 60,000 images and a test set containing 10,000 images. The 'train_test_split' function from the scikit-learn module can be employed for this purpose.

Model Evaluation

Reserve the test set, consisting of 10,000 images, exclusively for evaluating the accuracy of your trained model. This set serves as an independent measure of how well the model performs on unseen data.

Performance Visualization

Visualize training and test losses on the same graph as a function of the iteration index.

Label and Image Transformation

- **One-Hot Vector Representation**: For computational convenience, transform the digit labels into one-hot vectors. Each label, such as '5', will be represented as [0,0,0,0,0,1,0,0,0,0], and '0' will correspond to [1,0,0,0,0,0,0,0,0,0].

- **Flattening Images**: Flatten the input images to create a vector representation. Each original image, initially of size 28x28 pixels, should be transformed into a vector of size 785x1. The first component of this vector represents the bias term and is set to 1. This flattened format facilitates the processing of image data by machine learning algorithms.

Appendix: 'How-To' Fetch MNIST Dataset

Instructions for fetching the MNIST dataset will be provided in the appendix.

PART A: **Perceptron Learning Algorithm**

The Perceptron Learning Algorithm (PLA) for binary classification has been introduced in class. To extend its application to multi-class classification, we employ the one-vs-all strategy. This strategy involves training multiple binary PLA classifiers, each dedicated to distinguishing one class from the rest. In this approach, the multi-class problem is reduced to K binary problems, where K is the number of classes.

Specifically, the process involves the following stages:

-Binary Classification Setup:

In the first stage, we create a binary classifier to distinguish between the '0' class and all other classes. We assign the label +1 to examples labeled '0' and -1 to examples labeled '1', '2', ..., '9'. The weight vector obtained from this classification is denoted as w^0.

-Iterative Binary Problems:

Subsequently, for each binary problem i from 1 to 9, we assign the label +1 to examples labeled 'i' and -1 to examples labeled '0', '1', '2', ..., '9' (excluding 'i'). The weight vectors resulting from these binary classifications are denoted as $w^0, w^1, \dots, w^9$.

-Weight Vector Collection:

Following this process iteratively for each class, we end up with K=10 weight vectors $w^0, w^1, \dots, w^9$, each dedicated to distinguishing one class from all the others.

In summary, the one-vs-all strategy facilitates the extension of the binary Perceptron Learning Algorithm to the multi-class perceptron classifier. This approach allows us to break down a complex multi-class problem into a set of binary problems, effectively training a set of classifiers to handle each class individually.

Prediction

For a new instance x, the prediction of the label is determined by selecting the class with the highest confidence score:

$$\hat{y} = argmax_{y\in\{0,\dots,9\}} w^y x \ .$$

In this expression, $\hat{y}$ represents the predicted label, and $w^y x$ signifies the confidence score associated with class y. The argmax operation identifies the class for which the confidence score is maximized, resulting in the predicted label for the given instance x.

The **Perceptron Learning Algorithm** for binary classification in the linearly separable case:

- Initialization: $w = w_0$

- Pick a misclassified example and denote it as $(x(t), y(t))$

  i.e., $y(t) \neq h\big(x(t)\big) = sign\big(w^T(t)x(t)\big)$

- Update the weights

$$w(t) + y(t)x(t) \rightarrow w(t+1)$$

-Continue iterating until there are no misclassified examples in the training set.

In cases where the data is not linearly separable, the Perceptron Learning Algorithm (PLA) will never terminate and can exhibit unstable behavior, potentially transitioning from a well-performing perceptron to a poorly-performing one within a single update. To address this challenge and obtain an approximate solution, a modification known as the pocket algorithm is introduced.

The pocket algorithm serves to enhance the stability and performance of PLA by maintaining, or keeping 'in its pocket,' the best weight vector encountered up to iteration t during the PLA process. Throughout the iterations, the algorithm evaluates the performance of the current weight vector against the one stored in its pocket.

At the conclusion of the iterations, the weight vector that yielded the best performance, as measured by some criteria (e.g., achieving the lowest error), is reported as the final hypothesis. In summary, the pocket algorithm enhances the robustness of PLA by retaining the most effective weight vector encountered during the learning process, contributing to a more reliable and accurate solution. Specifically,

- Set the pocket weight vector $\widehat{w}$ to $w(0)$ of PLA.
- For t=0,…,T-1 do:
  - Run PLA for one update to obtain $w(t+1)$.
  - Evaluate $E_{in}(w(t+1))$.
  - If $w(t+1)$ is better than $\widehat{w}$ in terms of E_{in}, set $\widehat{w}$ to $w(t+1)$.
- Return $\widehat{w}$.

A1. Apply the multi-class perceptron algorithm to address the MNIST classification problem.
A2. Compute the confusion matrix for the multi-class classification problem on the test data and determine the accuracy (ACC).
A3. Generate the table of confusion for each digit, and calculate the sensitivity (True Positive Rate, TPR) for each class.
A4. Provide a comprehensive discussion of your results, considering the model's overall performance and its effectiveness in distinguishing individual digits.

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of data for which the true values are known. It allows the visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class (or vice versa). The name "confusion matrix" stems from the fact that it makes it easy to see if the system is confusing two classes, i.e., commonly mislabeling one as another.

Here is a breakdown of the components of a confusion matrix:

True Positives (TP): The number of instances correctly predicted as the positive class.

True Negatives (TN): The number of instances correctly predicted as the negative class.

False Positives (FP): The number of instances incorrectly predicted as the positive class.

False Negatives (FN): The number of instances incorrectly predicted as the negative class.

The confusion matrix is often used to compute various performance metrics for a classification model, such as accuracy, precision, recall, and F1 score. These metrics provide insights into the strengths and weaknesses of the model in terms of correctly and incorrectly classified instances.

The Accuracy is defined as ACC=(TP+TN)/(TP+TN+FP+FN)
The sensitivity (True positive rate) is defined as TPR = TP/(TP+FN)
The selectivity (True negative rate) is defined as TNR = TN/(TN+FP)

PART B: Softmax Regression

In the subsequent section, you will implement Softmax Regression on the MNIST dataset. Softmax regression, also known as multinomial logistic regression, extends logistic regression to accommodate multiple classes, denoted as $y_n \in \{1, \dots, K\}$ where K represents the number of classes.

When presented with a test input x, our objective is to have our hypothesis estimate the probability $P(y = k|\underline{x})$ for each value of k=1,…,K. In other words, we aim to assess the likelihood of the class label assuming each of the K different possible values. Consequently, our hypothesis will yield a K - dimensional vector, where the elements sum to 1, providing us with our K estimated probabilities:

$$\underline{h}(\underline{x}) = \begin{pmatrix} P(y = 1|\underline{x}) \\ \dots \\ P(y = K|\underline{x}) \end{pmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\underline{w}^{(j)^T} \underline{x})} \begin{pmatrix} \exp(\underline{w}^{(1)^T} \underline{x}) \\ \dots \\ \exp(\underline{w}^{(K)^T} \underline{x}) \end{pmatrix}.$$

B1. Minimize the following softmax cost function on the training data with respect to the vectors $\underline{w_j}$ j=0,1,…,9 using the gradient descent method:

$$E_{in}(\underline{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} 1\{y_n = k\} \log \frac{e^{w_k^T x_n}}{\sum_j e^{w_j^T x_n}}.$$

Model Evaluation

B2. Calculate confusion matrix and accuracy (ACC) for the multi-class classification problem on the test data.

B3. Calculate the table of confusion for each digit and compute sensitivity (TPR) for each class.

B4. Discuss the results.

Part C: Linear Regression

In this section of the assignment, you will explore the application of linear regression using the least squares method to classify the digits in the MNIST dataset.

C1. Formulate the digit classification problem as a linear regression task.

C2. Evaluate the performance of the linear regression models on the test set.

C3. Compare the results obtained using linear regression with those achieved through the perceptron algorithm and softmax regression, as outlined in the earlier parts of the assignment.

C4. Discuss the strengths, weaknesses, and limitations of using linear regression for MNIST digit classification.

To import the MNIST dataset in Python, you can use the fetch_openml function from scikit-learn, which provides a convenient interface for downloading and loading various datasets, including MNIST. Here's an example:

```
from sklearn.datasets import fetch_openml

# Fetch MNIST dataset
mnist = fetch_openml('mnist_784', version=1)

# Access features (pixel values) and labels
X, y = mnist['data'], mnist['target']
```