

**1. Suppose you want to classify pictures as outdoor/indoor and daytime/nighttime. Should you implement two Logistic Regression classifiers or one Softmax Regression classifier? Why?**

For the task of classifying images into outdoor/indoor and daytime/nighttime groups, applying two separate Logistic Regression classifiers is a suitable choice. Considering the nature of the problem, we are dealing with two distinct binary classification tasks, each with its own two classes (outdoor vs indoor, daytime vs nighttime), and each image can be classified into one class from each classifier. The approach of using two separate Logistic Regression classifiers allows for handling and optimizing each task independently, while providing flexibility in applying preprocessing techniques and feature selection. It also opens up the possibility of analyzing the relationships between labels more deeply through the classification results. This is different from using a Softmax Regression classifier, which is more suitable for multi-class classification problems with independent and non-overlapping classes.

**2. Implement Batch Gradient Descent with early stopping for Softmax Regression (without using Scikit-Learn)**

To perform Batch Gradient Descent with early stopping for Softmax Regression without using Scikit-Learn, the following steps need to be carried out:

Step 1: Initialize weights: Start by randomly initializing weights or setting them to 0 for each feature and for each class.

Step 2: Convert labels to one-hot form: Use one-hot encoding to represent the labels, where each label is represented by a vector of length equal to the number of classes, with only one element being 1 and the rest being 0.

Step 3: Compute Softmax: Apply the softmax function on the weighted sum of inputs to calculate the probabilities for each class.

Step 4: Update weights: Use gradient descent to update the weights based on the gradient of the loss function (usually cross-entropy loss) with respect to each weight. Repeat this process for each batch of training data.

Step 5: Early stopping: During the training process, monitor the loss on the validation set. If the loss does not decrease after a certain number of epochs determined by "patience," stop the training to avoid overfitting.

Step 6: Evaluate the model: Use the test set to evaluate the final performance of the model.

***(Đoạn code cho phần này trong file Jupyter notebook đính kèm.)***

**5. Why can't we use the mean square error cost function used in linear regression for logistic regression**?

In logistic regression, we do not use the mean squared error (MSE) loss function for several main reasons:

*First,* logistic regression uses the sigmoid function to predict probabilities, making the resulting loss surface non-flat and introducing multiple local minima, which makes the optimization process challenging with gradient descent in finding the global optimum efficiently.

*Second,* MSE is not suitable for the assumption. MSE assumes that errors follow a normal distribution, while logistic regression deals with binary outcomes and models the probabilities of classes using the sigmoid function.

*Third,* the logistic loss function is more appropriate for the probabilistic nature. Instead of MSE, logistic regression typically uses the log loss (also known as binary cross-entropy loss) for binary classification. This loss function directly measures the difference between the predicted probability and the actual class value, penalizing mispredictions more heavily. The log loss function is better suited for classification problems and better reflects the probability nature of logistic regression.

*Fourth,* it leads to better training performance. The log loss function creates a flat loss surface for logistic regression, allowing optimization algorithms to more reliably find the global optimum, resulting in a more stable and efficient training process compared to using MSE.

**7. What is the fundamental idea behind Support Vector Machines?**

The fundamental idea behind Support Vector Machines is to fit the widest possible "street" between the classes. In other words, the goal is to have the largest possible margin between the decision boundary that separates the two classes and the training instances. When performing soft margin classification, the SVM searches for a compromise between perfectly separating the two classes and having the widest possible street (i.e., a few instances may end up on the street).

Another key idea is to use kernels when training on nonlinear datasets.

**8. What is a support vector?**

After training an SVM, a support vector is any instance located on the "street", including its border. The decision boundary is entirely determined by the support vectors. Any instance that is not a support vector (i.e., off the street) has no influence whatsoever. Computing the predictions only involves the support vectors, not the whole training set.

**12. Say you’ve trained an SVM classifier with an RBF kernel, but it seems to underfit the training set. Should you increase or decrease γ (gamma)? What about C?**

To reduce the tightness of regularization, it is necessary to increase the values of γ (gamma) and C. Increasing gamma makes the bell-shaped curve narrower, resulting in more transformations of the decision boundary around data points. Conversely, decreasing the value of gamma makes the bell-shaped curve wider, leading to a smoother decision boundary. Reducing the value of C leads to a wider "road" but may cause more boundary violations.

**16. What is the approximate depth of a Decision Tree trained (without restrictions) on a training set with one million instances?**

The depth of a well-balanced binary tree containing m leaves is equal to log2(m), rounded up. A binary Decision Tree (one that makes only binary decisions, as is the case of all trees in Scikit-Learn) will end up more or less well balanced at the end of training, with one leaf per training instance if it is trained without restrictions. Thus, if the training set contains one million instances, the Decision Tree will have a depth of log2(106) ≈ 20 (actually a bit more since the tree will generally not be perfectly well balanced).

**18. If a Decision Tree is overfitting the training set, is it a good idea to try decreasing max\_depth?**

If a Decision Tree is overfitting on the training set, reducing the max\_depth can be a good idea as it will limit the complexity of the model and apply regularization. Reducing max\_depth makes the decision tree less complex and reduces the likelihood of overfitting by preventing the tree from growing too deep and detailed unnecessarily. However, reducing max\_depth can also lead to a slight decrease in the model's performance on the training set, but it can improve the overall performance of the model on the validation set or new data.

**20. If it takes one hour to train a Decision Tree on a training set containing 1 million instances, roughly how much time will it take to train another Decision Tree on a training set containing 10 million instances?**

The computational complexity of training a Decision Tree is O(n × m log(m)). So if you multiply the training set size by 10, the training time will be multiplied by

K = (n × 10m × log(10m)) / (n × m × log(m)) = 10 × log(10m) / log(m). If m = 10^6, then K ≈ 11.7, so you can expect the training time to be roughly 11.7 hours.

**14. Train an SVM classifier on the MNIST dataset. Since SVM classifiers are binary classifiers, you will need to use one-versus-the-rest to classify all 10 digits. You may want to tune the hyperparameters using small validation sets to speed up the process. What accuracy can you reach?**

***(Đoạn code cho phần này trong file Jupyter notebook đính kèm.)***

The SVM model with SVC (C=3.87868, gamma=0.001767) achieved impressive accuracy, with about 99.92% on the training set and 94.33% on the MNIST test set. This high level of accuracy demonstrates the model's strong classification and excellent generalization capabilities. The small difference between the accuracy of the training and testing sets shows that the model does not have an overfitting problem. Despite being fine-tuned with resource constraints, the model still shows high performance, which promises practical applicability and scalability to other arithmetic classification tasks.