CS303 Project2 Report

Subtask 1

1. Introduction

You are required to independently train a model using the provided training set. This model should then be utilized to predict labels for the test set. The predicted labels are to be saved in `classification_results.pkl` file. To exemplify this procedure, a Python script named `image_classification_demo.ipynb` has been supplied. You can run `image_classification_demo.ipynb` and observe the results. Additionally, you have the freedom to customize the provided demo by incorporating your own methodologies.

2. Methodology

1. Dynamically adjust the learning rate through warmup.
2. Set the random seed.

3. Experiments

1. Metrics: If the test accuracy of your algorithm exceeds that of the given baseline, you will get all the points for the subtask; otherwise, you get a score of 0 for the subtask.
2. Your Experimental results:

4. Further thoughts

The idea behind a warmup learning rate is to allow the model to initially converge to a reasonable solution with a conservative learning rate and then gradually increase the learning rate to expedite further convergence. This can be especially beneficial when training large and complex models. During the warmup phase, the learning rate is increased linearly or following a predefined schedule until it reaches its target or maximum value. This approach helps stabilize the training process and prevents the model from making large updates to its parameters before it has had a chance to explore the solution space. The specific details of how to implement warmup learning rates may vary depending on the framework or library used for machine learning, such as TensorFlow or PyTorch. It's often considered as a part of the overall learning rate schedule in the training process.

Setting a random seed in deep learning is a common practice to enhance the reproducibility of experiments. While it doesn't directly guarantee improved accuracy, fixing the random seed ensures that the same sequence of random numbers is generated during model training. This reproducibility is valuable for debugging, comparing different model configurations, and achieving consistent results across multiple runs.

Subtask 2

1. Introduction

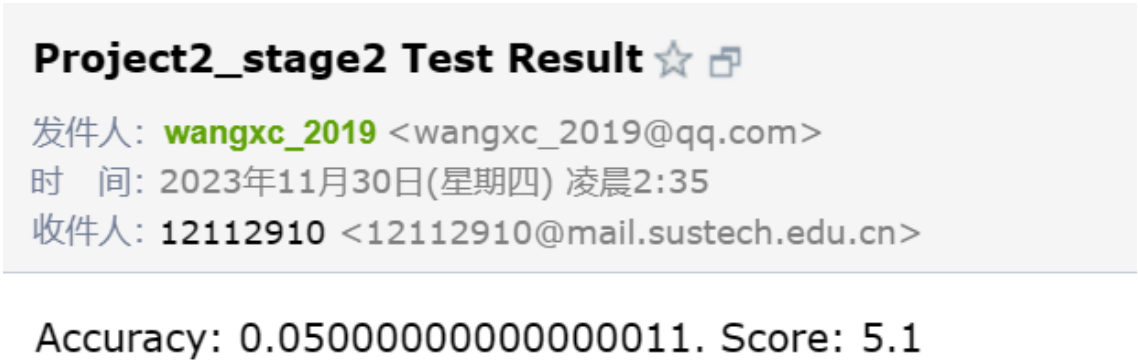
You are required to independently train a model using the provided image repository. This model should then be utilized to respond to image queries from the test set. For each image in the test set, you should find **5** similar images in the image repository. The response will be in the form of a list of image IDs retrieved from the image repository. The resulting answers must be saved in `retrieval_results.pkl` file. To exemplify this procedure, a Python script named `image_retrieval_demo.ipynb` has been supplied. You can run `image_retrieval_demo.ipynb` and observe the results. Additionally, you have the freedom to customize the provided demo by incorporating your own methodologies.

2. Methodology

1. Change the distance function to L1 loss.
2. Introduce random feature masking to enhance robustness.

3. Experiments

1. Metrics: When checking, you need to send `retrieval_results.pkl` to the teacher or SA. For each test image, suppose you submit n similar images (In this subtask, n = 5) and there are m images that are considered to be similar by the evaluation process, the accuracy should be m/n × 100%. The test accuracy is the average accuracy of the whole test set.
2. Your Experimental results:



4. Further thoughts

1. By changing the distance function to L1 loss in KNN, we adopt a different measure for assessing the dissimilarity between data points. Unlike traditional Euclidean distance (L2 norm), which considers the squared differences, L1 loss sums up the absolute differences. This modification often proves effective in scenarios where outliers may disproportionately influence the distance calculation. The L1 loss is less sensitive to extreme values, making the KNN algorithm more robust to outliers and potentially improving its overall performance, especially in situations with noisy or skewed data distributions.

2. By incorporating random feature masking, we introduce a technique that involves randomly selecting and temporarily hiding certain features within our dataset. This process aims to improve the model's robustness by preventing it from relying too heavily on specific features during training. Random feature masking helps the model generalize better to unseen data and enhances its ability to handle variations or uncertainties in the input features. This approach can be particularly effective in scenarios where certain features may be noisy or prone to outliers, contributing to a more resilient and adaptable model.

Subtask 3

1. Introduction

You should write an algorithm to select no more than 30 features from the image features of the classification validation set. Here, a binary vector of feature mask is applied to each input image vector. That means for each input image vector x ∈ R^n, the same binary vector mask x ∈{0,1}^n is applied so that x̌ = x · mask will be the actual input for the trained model in testing. n denotes the dimensionality of the test input vector, and · denotes the inner product of two vectors. For example, if n=3, and there is a test input vector of 〈112,345,321〉 and the mask is 〈1,0,1〉, then the x̌ = x · mask = 〈112,345,321〉 · 〈1,0,1〉 = 〈112,0,321〉. The masked features (the feature value 345 in the above example) have no effect on the trained model which means the corresponding feature dimensions are given up. A demonstration implementation `feature_selection.ipynb` is provided. The test procedure is demonstrated in `image_recognition.ipynb`. It reads the mask vectors from `mask_code.pkl` which is the outcome of running `feature_selection.ipynb`, then uses the mask vectors to mask the features of the classification validation set, predicts the labels using a fixed trained model whose weights are loaded from `image_recognition_model_weights.pkl`, finally calculates the accuracy based on `classification_validation_label.pkl`.

You should re-implement the specific functions in the `feature_selection.ipynb` to construct their feature selection algorithm and then run the `image_recognition.ipynb` to see if the accuracy improves.

2. Methodology

Utilize some feature selection algorithms.

3. Experiments

1. Metrics: When checking, you need to send `mask_code.pkl` to the teacher or SA. The test procedure is similar to `image_recognition.ipynb`. The difference is that when we test, we use the test data. If the test accuracy of your algorithm exceeds that of the given baseline, you will get all the points for the subtask; otherwise, you get a score of 0 for the subtask.
2. Your Experimental results:

Project2_stage3 Test Result ☆

发件人: wangxc_2019 <wangxc_2019@qq.com>
时　间: 2023年11月30日(星期四) 凌晨2:34
收件人: 12112910 <12112910@mail.sustech.edu.cn>

Accuracy: 0.382814844453336, Baseline Accuracy: 0.15094528358507553, Your Sorce: 5.1.

4. Further thoughts

The statement "Utilize some feature selection algorithms" refers to the application of techniques that automatically select a subset of relevant features from the original set of features in a dataset. The effectiveness of feature selection algorithms lies in several key advantages:

1. **Dimensionality Reduction:** Feature selection helps in reducing the number of input features, addressing the curse of dimensionality. This is particularly beneficial when dealing with high-dimensional datasets, as it can lead to simpler, more interpretable models and alleviate computational complexity.
2. **Improved Model Performance:** By focusing on the most informative features, feature selection algorithms can enhance the performance of machine learning models. The reduced feature set often leads to models that are less prone to overfitting and better generalize to new, unseen data.
3. **Enhanced Interpretability:** Simplifying the feature set makes it easier to interpret the model's behavior. Identifying and using only the most relevant features can provide insights into the underlying patterns in the data and improve the model's transparency.
4. **Faster Training and Inference:** Working with a reduced set of features generally results in faster training and inference times. This is crucial in scenarios where computational resources are limited or where real-time processing is required.
5. **Noise Reduction:** Feature selection can help mitigate the impact of irrelevant or noisy features, contributing to the creation of more robust models that are less sensitive to irrelevant information.
6. **Addressing Collinearity:** Feature selection methods can handle multicollinearity by selecting a subset of features that are less correlated, which can lead to more stable and reliable model estimates. In summary, utilizing feature selection algorithms is effective for enhancing model performance, interpretability, and efficiency by focusing on the most relevant features and reducing the complexity of the dataset.