Project Machine Learning
Task 1: Deep Neural Networks

Saurabh Varshneya

saurabh.varshneya@cs.rptu.de,

September 24, 2024

Important: Please read the documentation and the README files in your repository carefully. They contain detailed instructions and hints on how to perform and submit your tasks. If you have any further questions, don't hesitate to contact me.

Introduction

In *Machine Learning* our goal is to learn or extract meaningful patterns from given *data*. A datum in the so called *supervised* setting is a pair $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$, where $\mathbf{x}$ represents the *input* (e.g., an image) and y represents the *label* (e.g., a word like "dog"). A collection of such pairs $D := \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathcal{X} \times \mathcal{Y}$ is called a *dataset* and can also be represented as a vector of inputs $X = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n\end{bmatrix}^\mathsf{T} \in \mathcal{X}^n$ and a vector of labels $\mathbf{y} = \begin{bmatrix}y_1 & y_2 & \cdots & y_n\end{bmatrix}^\mathsf{T} \in \mathcal{Y}^n$. Our goal is to find a function f based on the given dataset D, such that $f\colon \mathcal{X} \to \mathcal{Y}$ with $f(\mathbf{x}) \approx y$ for all $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$.

We call the setting where $\mathcal{Y}$ is finite *classification*, and the elements of $\mathcal{Y}$ are then *classes*. For instance, assume that we have a bunch of 512×512 pixel RGB images showing either cats or dogs and we, as human experts, have already determined for each image whether it shows a cat or a dog. We have $\mathcal{X} = \{0, 1, \ldots, 255\}^{3\times512\times512}, \mathcal{Y} = \{\text{"cat"}, \text{" dog"}\}$ and try to find a classifier f that determines if a given image shows a cat or a dog. Ideally, f should also make correct decisions on new, unseen images, if they are reasonably similar to the already seen images.

1 K Nearest Neighbor Classification

The following content details preliminary information required for carrying out the tasks in the next section.

KNN Definition The classical k-nearest neighbor (KNN) algorithm is perhaps one of the simplest approaches to perform classification. Given a training dataset $D_{\text{tr}} := \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathcal{X} \times \mathcal{Y}$ and $k \in \mathbb{N}$, we assign a label to an unseen test point $\mathbf{x}_{\text{te}} \in \mathcal{X}$ by finding the k closest training points to it and choosing the most frequent label from those points.

Let $\mathcal{N}(\mathbf{x}_{\text{te}})$ be the set of k nearest neighbors for a test point $\mathbf{x}_{\text{te}}$, where

$$\mathcal{N}(\mathbf{x}_{\text{te}}) = \underset{\{(\mathbf{x}_i, y_i): i \in \{i_0, \ldots, i_k\}\}}{\arg\min} \sum_{i \in \{i_0, \ldots, i_k\}} d(\mathbf{x}_{\text{te}}, \mathbf{x}_i) \quad \text{s.t.} \quad \forall \alpha, \beta \in \{i_0, \ldots, i_k\}^2 : \alpha \neq \beta.$$

Here, $d \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a distance function, quantifying the closeness of datapoints in $\mathcal{X}$. This allows us to predict the label of $\mathbf{x}_{\text{te}}$ as

$$y_{\text{te}} = f(\mathbf{x}_{\text{te}}) := \underset{y' \in \mathcal{Y}}{\arg\max} \left| \left\{ y' = y : (\mathbf{x}, y) \in \mathcal{N}(\mathbf{x}_{\text{te}}) \right\} \right|.$$

Note that $\mathcal{N}(\mathbf{x}_{\text{te}})$ and y_{te} might not be well-defined since the min and max are not necessarily unique. In order to fix this, we must employ some kind of tie breaker strategy, a simple approach would be to just randomly select one of the min/max elements. You may check Tasks a) and b) listed below with this information.

Evaluation Techniques To rate the performance of a model, we need to define an evaluation metric. The most common performance metric is *accuracy*: Given some dataset D and a classifier $f \colon \mathcal{X} \to \mathcal{Y}$, for example our KNN classifier, we define the accuracy of f on the dataset D as

$$\text{Acc}(f, D) := \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \mathbb{I}_{f(\mathbf{x}) = y},$$

where the *indicator function* $\mathbb{I}_{\mathtt{p}}$ returns 1 if $\mathtt{p}$ is true, otherwise it returns 0. In words, the accuracy is the fraction of datapoints in D that are correctly classified by f. We also need to find a proper test set D, which should be distinct of the training data and hence contain unseen samples. However, data is often a rare and precious resource in machine learning and access to fresh datapoints is not always available. One possible way to overcome this dilemma is called m-fold *cross validation*:

- Partition the dataset D *randomly* into m equally sized *folds*.
- Choose one fold as the test set and train your classifier on the union of the remaining ones. Evaluate the performance of the classifier on the chosen test set.
- Repeat the training/evaluation process, while choosing each fold as the test set exactly once. Report the average of the performance metrics over all m runs.

You may check tasks c), d) and e) listed below with this information.

Feature Engineering We described the KNN algorithm using a distance function directly on the input space (e.g., space of images). Using pixel intensities of raw images to compute distances for the KNN algorithm will tend to give poor results. One of the reasons is that slight geometric variations such as rotation, translation, or scaling cause a drastic change in the pixel intensities although the semantic information contained in the image stays the same. To remedy this, we can compute distance functions on some *feature* representations of the images that are less sensitive to geometric variations. Typically, in computer vision, such feature representations are obtained by convolving the image with a *filter*. The output of a convolutional operation is a two-dimensional array (i.e., an image) and often referred to as the *feature map*. For example, there are filters that blur or sharpen the image and filters that extract edges, corners, or blobs. Let the image be $I \in \mathbb{R}^{H \times W \times C}$ and our filter be $K \in \mathbb{R}^{F \times F \times C}$. The output of the convolutional operation at location (i, j) is:

$$O_{i,j} = \sum_{l=1}^{F} \sum_{m=1}^{F} \sum_{n=1}^{C} I_{i-(l-1),j-(m-1),n} K_{l,m,n}$$

You may check the Tasks f), g), h), and i) with this information.

1.1 Tasks

a) The template in your git-repository includes utilities for downloading the *Strange Symbols* dataset. It consists of 28×28 pixel grayscale images divided into 15 different classes. Create a plot containing a few images from each class in the dataset and comment about what you think they represent.

b) Implement a generic version of the KNN algorithm—as explained above—that takes a distance function d and a number k as parameters.

c) Before we can use our KNN classifier on the Strange Symbols dataset, we need to find a suitable function d that quantifies the distance between two images. To keep things simple, we can interpret a 28×28 image as a vector in $\mathbb{R}^{784}$ and then use the Euclidean distance between the vectors, i.e., for two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^{784}$ we have

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\langle \mathbf{x} - \mathbf{x}', \mathbf{x} - \mathbf{x}' \rangle} = \sqrt{\sum_{j=1}^{784} (x_j - x'_j)^2}.$$

Estimate the performance of KNN on the Strange Symbols dataset using 5-fold cross validation for all $k \in \{1, \ldots, 10\}$ and the Euclidean distance function. Do not employ a library for this, implement cross validation by yourselves instead. Finally, plot your results in a single figure.

d) Naturally, you would choose k as the value that maximized the accuracy estimated in item c). Do you think that the corresponding accuracy value is a good estimate for the performance of the classifier on fresh, unseen data? Or does it under/overestimate the true performance? Explain your answer.

e) The Euclidean distance used in item c) is not the only possible distance measure between two images from the Strange Symbols dataset and probably not even the most suitable for the task. Find (at least) two more applicable distance measures, run 5-fold cross validation, and compare the results. For this task and all following, use the k found in item d).

f) Implement the convolutional operation, explained above, using Python and NumPy only.

g) Obtain feature representations for the images by applying the convolutional operation with at least two different filters. For example, one filter could be to blur the image and another one to detect edges. Apply the KNN algorithm on the features separately as well as the concatenation of all features maps. Run 5-fold cross validation using at least one distance measure and compare the results. You can check your implementation of the convolution by comparing your results with any existing implementation.

h) So far, our KNN algorithm predicts the most frequent label in the nearest neighbor set with all elements in the set being of equal importance. Extend your implementation of the KNN algorithm so that each neighbor is assigned a weight that is inversely proportional to the distance from a given test point. Run 5-fold cross validation using at least one distance measure on the images and compare the results.

i) For the features obtained from the different filters in item g) and for the raw images, plot each of: the features/images and the nearest neighbors to five random misclassified samples. Indicate the ground truth (label) along with the nearest neighbor image samples. Try to explain why these samples got misclassified.

j) Write a small conclusion comparing everything you've done and include a table of all results. Feel free to add more experiments. For instance, you can try to combine different tasks or try to find the best k when using features or different distance measures.

1.2 Competition

The competition will be about who can *speed up the testing method by the largest margin.* Either way, your tasks are as follows:

1. Make your implementation of KNN deterministic. Specifically, if there are two training points with the same distance to the testing point, consider only the one with a lower index. If there is more than one possible choice when determining the class label from the k nearest neighbors, choose the lowest class number.
2. Perform any changes to the implementation that you deem necessary to make the testing procedure faster. Please don't use any approximate KNN algorithms for that and use the Euclidean distance on images as described in item c).
3. Don't use any additional libraries for your implementation, except for Numpy, PyTorch and their dependencies.
4. The template contains a driver program `knn_challenge.py` that will run our evaluation procedure. You should make the necessary changes to this file and run it to test your implementation.

In our evaluation procedure, we will basically just run the `knn_challenge.py` script with $k = 5$, 100 repetitions and use the reported results. Some general info:

- While your implementation should be as fast as possible, the predictions that it outputs should still be the same as that of any slower KNN implementation. Therefore, we will compare the predictions of your code to those from our reference implementation. If there are less than 95% matches between the two, we must assume that you did not implement KNN correctly and you will be disqualified from this competition.
- We won't run the script using the provided `test_data.pt` as the testing data file. Instead, we will use a different file that contains the same amount of data points.