Generative AI and Visual Synthesis SoSe 24

Prof. Dr. Björn Ommer

Exercise Sheet 2: Classification

Solution

Important Notes:

1. **Email**: Frequently check your email address registered for Moodle. All notifications regarding the course will be sent via Moodle.
2. **Due date**: The exercise sheets will usually be uploaded around 1 week in advance of the due date, which is indicated at the top of the exercise sheet.
3. **Submission**: Your submission should consist of one single ZIP file which includes a PDF file and the corresponding codes. Both the PDF file and ZIP file should contain your surname and your matriculation number (*Surname-MatriculationNumber.zip*) for grading purposes. You may use Jupyter [1] for exporting your python notebooks as PDF, but you still have to hand in your *.ipynb* or *.py* files for us to test your code. For this exercise, please export the *.ipynb* as a PDF file and include that in the ZIP file. **Submissions that fail to follow the naming convention will not be graded.**

This exercise may look intimidating by its length, however, this is because it contains general information about submissions and also many explanations and hints that will help you through the exercise.

General Information:

All programming exercises must be completed in Python. The proposed solution for the exercises will be compiled in Python 3 (3.8 or above). You may use standard Python libraries or Anaconda (open source distribution for Python) to complete your exercises if not mentioned otherwise. We will be using PyTorch[2] as our main deep learning framework.

If you have any problems or questions about the exercise, you are welcome to use the student forum on the lecture Moodle page, as most of the time other students might have similar question. For technical issues about the course (for example, in case you cannot upload the solution to Moodle) you can write an email to the person responsible for the exercise (indicated at the top of the exercise sheet).

[1] https://jupyter.org/

[2] https://pytorch.org/

Task 1: k-NearestNeighbours (10P)

In this task you will make yourself familiar with the k-Nearest Neighbours (KNN) classifier by implementing it from scratch. The file *task1.py* contains code for creating a synthetic dataset you should use during this task. If not mentioned otherwise, your implementation should only rely on pure `numpy` code and make sure it is vectorized accordingly. You can make use of *task1.py* as code skeleton for implementing the subsequent tasks:

1. Visualize the dataset. You can use *matplotlib* or any other plotting library of your choice. (1P)

2. Implement a method `kneighbours` to return the indices and the distance of the k nearest neighbours from the training set for a given query point. Use the euclidean distance as distance metric. (2P)

3. Add a `predict` functionality to your KNN class which returns the predicted label for a given query point. (1P)

4. Fit your KNN model for $k = 5$ to the data. Repeat this step using the `KNeighborsClassifier` provided by `sklearn` and make sure both return the same predictions. (1P)

5. Run your KNN model with different values of $k = 2^i$ for $i = 0, ..., 9$. (1P)

6. Plot the decision boundary for each k. (1P)
   Hint: Evaluate the classifier on a grid within a box. Use around 100 points in each direction and generate the grid via `np.meshgrid`. Visualize the area with a contour plot (`contourf` using `matplotlib`).

7. How does the decision boundary change with k? What would happen if k is equal to the number of train samples? (1P)

8. Report class probabilities $p(c)$ on the train set. Further plot $p(\mathbf{x}_n) = \frac{k}{NV^\star}$ by estimating $V^\star$ as the area of the smallest circle needed to include k nearest neighbours for query point $\mathbf{x}_n$. Use $k = 2, 4, 8, 16, 32$. Repeat the same plots for $p(\mathbf{x}_n|c)$. (2P)

Task 2: Linear Regression (4P)

The goal of this task is to implement a simple linear least squares (LLS) classifier. Let $\mathbf{x}_i$ denote the i-th training sample and $y \in \{-1, 1\}$ its corresponding label. Then the linear score function f with parameters $\theta = [\omega_j, b]$ is given by

$$f(\mathbf{x}, \theta) = \langle \omega, \mathbf{x} \rangle + b = \langle \theta, \bar{\mathbf{x}} \rangle \tag{1}$$

which we could shorten by applying the bias trick and write our data as $\bar{\mathbf{x}} = [\mathbf{x}, 1]$. To fit the parameters to training data, one needs to minimize the following criterion

$$\mathcal{L} = \frac{1}{2} \sum_{n=1}^{N} (f(\bar{\mathbf{x}}_i, \theta) - y_i)^2 \tag{2}$$

which is solved by

$$\theta^\star = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y} \tag{3}$$

1. Complete the `fit` and `predict` routine in *task2.py.* (2P)

The file *task2.py* contains a synthetic dataset for which one can control the magnitude of an outlier n during generation. Generate datasets for $n = 2^i,\ i = 0, ..., 4$ and solve the following tasks:

2. Visualize the dataset (0.5P)
3. Fit your LLS model to the data, report accuracy on the test set and visualize the decision boundary of the classifier. (1P)
4. How is the fit affected by the outlier? Give a short explanation. (0.5P)

Task 3: Softmax Regression & Optimization (6P)

In this exercise we will implement Softmax Regression from scratch using only `numpy` and make use of numerical optimization to find optimal parameters of mentioned classifier. With $\mathbf{x}^{(i)} \in \mathbb{R}^D$ we denote the i-th of N total data points with D dimensions and $y_i \in 1, ..., C$ beeing its corresponding class label. The probability for each class is

$$p(c|\mathbf{x}) = \frac{\exp\left(\theta_0^{(c)} + \sum_{j=1}^{D} \theta_j^{(c)} x_j\right)}{\sum_{k=1}^{C} \exp\left(\theta_0^{(k)} + \sum_{j=1}^{D} \theta_j^{(k)} x_j\right)} \tag{4}$$

To find optimal parameters $\theta_j^{(c)}$ for a given dataset $\mathbf{x}, \mathbf{y}$ the cross entropy loss should be applied

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{c=1}^{C} \log p(c|\mathbf{x}_i) \mathbb{1}[y_i = c] \tag{5}$$

where $\mathbb{1}$ denotes the indicator function which is 1 if y_i is equal to the current class c otherwise 0. You can take *task3.py* as a starting point if you wish.

1. Derive a formula for the gradient $\frac{\partial \mathcal{L}}{\partial \theta_j^{(c)}}$. To do so, just derive the solution for a single example $\mathcal{L}_i$ and apply the chain rule of calculus $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial g}\frac{\partial g}{\partial \theta}$ with $g = \theta_0 + \sum_{j=1}^{D} \theta_j x_j$. For this task it is enough if you report derivatives of each chain rule component alone to get full points. (3P)
2. Load the digits dataset (`load_digits`) from the `sklearn` library and create a train-test split with a75% to 25% ratio.
3. Implement a optimization routine using gradient descent. It is highly recommended to use the stabilization trick from the lecture for the calculation of softmax values. (2P)
4. Run your optimization scheme for 1000 steps with a learning rate of $lr = 0.001$ and plot evolution of cross entropy loss and accuracy on training and test split. Report the final accuracy on the test set. (1P)

Solution

First of all we will look at a single element $\mathcal{L}_i$ from $\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_i$ as the derivatives for each element are just summed up. To simplify the derivative we apply the chain rule of calculus first which gives us

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \mathbf{p}}\frac{\partial \mathbf{p}}{\partial \mathbf{g}}\frac{\partial \mathbf{g}}{\partial \theta} \tag{6}$$

with $g_j = \theta_{j,0} + \sum_{d=1}^{D} \theta_{j,d} x_d$. We start of by calculating the first part of the chain rule of calculus which is the cross entropy part, where we derive the loss towards each class probability p_c, $\frac{\partial \mathcal{L}}{\partial p_c}$:

$$\frac{\partial \mathcal{L}}{\partial p_c} = -\frac{\partial}{\partial p_c}\sum_{k=1}^{C} \log p_k \mathbb{1}[y = k] = -\frac{1}{p_c}\mathbb{1}[y = c] \tag{7}$$

Next we move over to derive the probability $\frac{\partial p_c}{\partial g_j}$: Here the application of the quotient rule may be helpful

$$f(x) = \frac{g(x)}{h(x)} \implies \frac{\partial f}{\partial x} = \frac{\frac{\partial g}{\partial x}h - g\frac{\partial h}{\partial x}}{h^2} \tag{8}$$

and we also need to differentiate between two cases which are:
$c = j$:

$$\begin{aligned} \frac{\partial p_c}{\partial g_j} &= \frac{\exp(g_c)\left(\sum_{k=1}^{C}\exp(g_k)\right) - \exp(g_c)\exp(g_k)}{\left(\sum_{k=1}^{C}\exp(g_k)\right)^2} \\ &= \frac{\exp(g_c)}{\left(\sum_{k=1}^{C}\exp(g_k)\right)}\frac{\left(\sum_{k=1}^{C}\exp(g_k)\right) - \exp(g_j)}{\left(\sum_{k=1}^{C}\exp(g_k)\right)} \\ &= p_c(1 - p_j) \end{aligned} \tag{9}$$

$c \neq j$:

$$\begin{aligned} \frac{\partial p_c}{\partial g_j} &= \frac{0 - \exp(g_c)\exp(g_j)}{\left(\sum_{k=1}^{C}\exp(g_k)\right)^2} \\ &= -\frac{\exp(g_c)}{\left(\sum_{k=1}^{C}\exp(g_k)\right)}\frac{\exp(g_j)}{\left(\sum_{k=1}^{C}\exp(g_k)\right)} \\ &= -p_c p_j \end{aligned} \tag{10}$$

The last two terms we can merge using the Kronecker delta δ_{ij}

$$\frac{\partial p_c}{\partial g_j} = p_c(\delta_{cj} - p_j), \qquad \delta_{cj} = \begin{cases} 1, & \text{if } c = j, \\ 0, & \text{else} \end{cases} \tag{11}$$

The last derivative is short again:

$$\frac{\partial g_j}{\partial \theta_a} = x_a \tag{12}$$

Important Note: Submit exactly one ZIP file via Moodle before the deadline. The ZIP file should contain your executable code and your report in PDF format. Make sure that it runs on different operating systems and use relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The PDF file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends, and captions. Missing plots will result in point reduction.