10907 Pattern Recognition

Lecturers
Prof. Dr. Ivan Dokmanić ⟨ivan.dokmanic@unibas.ch⟩

Tutors
Alexandra Flora Spitzer ⟨alexandra.spitzer@stud.unibas.ch⟩
Roman Fries ⟨r.fries@unibas.ch⟩
Cheng Shi ⟨cheng.shi@unibas.ch⟩
Vinith Kishore ⟨vinith.kishore@unibas.ch⟩
Valentin Debarnot ⟨valentin.debarnot@unibas.ch⟩

Assignment 3

Important:
Deadline: 09.11.2023
Total points: 9

Summary

The math part in this assignment contains exercises related to logistic regression and the discrete Fourier transform. The coding part involves computing new image features using convolution and filter.

Instruction for submission Go to the course page `10907 Pattern Recognition`, you will see two assignment items, `Assignment 3: Coding` and `Assignment 3: Math`.

- For the Math part of this assignment, you upload the solution as a *single* `.pdf` file in the `Assignment 3: Math`. For each exercise, you must specify which pages contain the answer. After submission, please double check that all solutions are uploaded successfully.
- For the Coding part of this assignment, you will find a code template in our course repo[1]. The code template `logistic.py`, `filter.py` and `classifier.py` contain the functions to be completed. Please upload the completed file to the `Assignment 3: Coding`.

Academic integrity You are encouraged to discuss the material with others, but it is essential to document these discussions by writing down the names of the individuals you worked with and any tools that helped you with the assignment. Any instance of cheating will be dealt extremely strictly, potentially resulting in receiving zero credit for the course at a minimum.

FAQ

1. What should I do after uploading my code? Double or triple check! make sure it goes through all the test cases and a score appears. This score is not your final score because you can correct the reported errors and resubmit the code again for a higher score. This is why I highly encourage uploading your code early so there is enough time to debug.
2. What do I do when seeing an error from Gradescope? Let us know as soon as possible. It could be that your implementation take too long to run (timeout error).
3. What to do if the TA makes a mistake grading my solution? If you disagree with the score, you can request a Regrade on Gradescope and we will take a second look as soon as possible.
4. How to use Piazza forum effectively? Open a public thread whenever you have a question regarding the class, homework, Gradescope, etc. You can choose to post anonymously. Some of us will try to answer quickly.

[1] `https://git.scicore.unibas.ch/dokmanic-courses/pr23/-/tree/main/assignments/assignment3/`

Math

Exercise 1 (Logistic Regression 1 point)

Consider a dataset where each data point, $\boldsymbol{x}$, belongs to $\mathbb{R}^2$, and has an associated binary label, $y \in \{+1, -1\}$. The joint distribution of the data and the label is given by:

$$p(\boldsymbol{x}, y) = \frac{1}{4\pi} \exp\left(-\frac{1}{2}\|\boldsymbol{x} - y\mathbf{1}\|_2^2\right) \tag{1}$$

where $\mathbf{1} = [1, 1]^\intercal$ is the all-one vector.

- Determine $P[y = 1|\boldsymbol{x}]$ and $P[y = -1|\boldsymbol{x}]$. (Hint: Bayes' theorem)
- Compute the weight vector for logistic regression which satisfies

$$\begin{bmatrix} P[y = -1|\boldsymbol{x}] \\ P[y = 1|\boldsymbol{x}] \end{bmatrix} = f_{\boldsymbol{w}}(\boldsymbol{x}) := \begin{bmatrix} e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}}/(e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}} + e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}) \\ e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}/(e^{\boldsymbol{w}_1^\intercal \boldsymbol{x}} + e^{\boldsymbol{w}_2^\intercal \boldsymbol{x}}) \end{bmatrix}$$

  Determine all possible values of $\boldsymbol{w} = (\boldsymbol{w}_1, \boldsymbol{w}_2)$ (Hint: Consider the shift invariance.)

Exercise 2 (Discrete Fourier Transform 2 points)

Recall that the discrete Fourier transform (DFT) of a vector $\mathbf{x} = [x[0], x[1], \ldots, x[N-1]]^T$ of length N is defined as

$$X[k] = (F\mathbf{x})[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \quad k \in \{0, 1, \ldots, N-1\}.$$

The inverse DFT of a vector $\mathbf{X} = [X[0], X[1], \ldots, X[N-1]]^T$ as

$$x[n] = (F^{-1}\mathbf{X})[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N}, \quad n \in \{0, 1, \ldots, N-1\}.$$

1. We have seen in class that any linear operator in finite dimensions can be represented by a matrix. Fourier transform is a linear operator. Determine the matrices representing F and F^{-1} such that $\mathbf{X} = F\mathbf{x}$ and $\mathbf{x} = F^{-1}\mathbf{X}$.

2. The inverse DFT is, not surprisingly, the inverse of the forward DFT operator. Show that $F^{-1}(F\mathbf{x}) = \mathbf{x}$ by directly multiplying the matrices. *Hint: use the formula for the sum of a geometric series.*

3. There is another relation between DFT and its inverse. Show that $F^{-1} = \frac{1}{N} F^*$ where F^* is obtained from F by transposing and then applying complex conjugation to all entries. This implies that up to a rescaling, the DFT matrix is orthogonal. Had we defined a slightly different transform $\tilde{F}$ where factors 1 and $\frac{1}{N}$ in the forward and inverse transform are both replaced by $1/\sqrt{N}$ then we would have that $\tilde{F}^{-1} = \tilde{F}^*$, which is to say that $\tilde{F}$ is an orthogonal matrix. Check that this is indeed the case.

4. Show that

$$\sum_{k=0}^{N-1} |X[k]|^2 = N \sum_{n=0}^{N-1} |x[n]|^2.$$

   This fact, that the ℓ_2-norm can be computed in either the spatial domain or the frequency domain, is called Parseval's theorem.

Exercise 3 (DFT and circular convolution 2 points)

Let $\mathbf{x}$ be a vector of length N. We define a circular shift of $\mathbf{x}$ by an amount n_0 as

$$(\text{circshift}_{n_0}(\mathbf{x}))[n] = x[(n-n_0)_N] = \begin{cases} x[n-n_0] \text{ if } n \geq n_0 \\ x[N-(n_0-n)] \text{ if } n < n_0 \end{cases} .$$

Let $\mathbf{X}$ be the DFT of $\mathbf{x}$.

1. Show that when applying the DFT to a circular shifted signal, the new DFT coefficients are just the DFT coefficients of the original signal modulated by a complex exponential:

$$(F(\text{circshift}_{n_0}(\mathbf{x})))[k] = X[k]e^{-j2\pi k n_0/N}.$$

2. Circular convolution of two vectors $\mathbf{w}$ and $\mathbf{x}$ of length N is defined as

$$(\mathbf{w} \circledast \mathbf{x})[n] = \sum_{\ell=0}^{N-1} x[\ell]w[(n-\ell)_N].$$

   Let $\mathbf{W}$ be the DFT of $\mathbf{w}$. Then show

$$(F(\mathbf{w} \circledast \mathbf{x}))[k] = W[k]X[k].$$

   This relation allows us to efficiently compute circular convolution of two signals using their DFTs. You will compare the two implementations—a direct one using the definition and the other based on the fast algorithm to compute the DFT—in the Coding exercise.

Coding

Exercise 4 (Logistic Regression- 2 points)

Logistic regression is a simple and efficient classifier for two-class problems. First, you will complete the functions that are core to the logistic regression classifier and then apply the classifier to a simple dataset.

1. Implement six functions that form the logistic regression classifier in the file `logistic.py`. The functions to be completed are:

   (a) `sigmoid()`: This takes an array as input and gives the sigmoid of each value of the array. The sigmoid function for a point $z \in \mathbb{R}$ is defined as :

   $$\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$$

   (b) `predict_score()`: Takes the weight vector $w \in \mathbb{R}^D$ of size D and a matrix $X \in \mathbb{R}^{N\times D}$, with N data points of size D, and outputs the logistic regression probability (score). For a single data point $x \in \mathbb{R}^D$, the score is defined as:

   $$\text{score}(x, w) = \frac{1}{1+e^{-w^T x}}$$

   Note: The above definition is for a single example. However, the function takes multiple examples as input in the form of a matrix. You can do this without using a `for` loop.

   (c) `predict()`: Predicts the class 0 or 1 using the data matrix X, weight vector w, and a threshold. This is defined as:

   $$\text{predict}(x, w, \text{threshold}) = \mathbb{1}_{\text{score}(x,w)\geq \text{threshold}}.$$

   (d) `cross_entropy_loss()`: Outputs the cross-entropy loss (CE) given the predicted probabilities and the true labels. This is defined as:

   $$CE(y_{\text{pred}}, y) = -\frac{1}{N}\left(\sum_{i=1}^{N} y[i]\log(y_{\text{pred}}[i]) + (1-y[i])\log\left(1-y_{\text{pred}}[i]\right)\right).$$

   Note: We use the natural logarithm.

   (e) `gradient()`: Computes the gradient of the cross-entropy loss with respect to w and returns it. This is defined as:

   $$\begin{aligned} y_{\text{pred}} &= \texttt{predict\_score}(X, w) \\ \texttt{gradient}(X, y) &= \frac{\partial}{\partial w} CE(y_{pred}, y) \end{aligned}$$

   (f) `train()`: Using an initial weight vector w_{init}, learn a new estimate of the weight vector using gradient descent for a predefined number of iterations (`epochs`) and a given learning rate `lr`. The pseudocode for the training algorithm is given in Algorithm 1.

2. Once implemented, you will apply the classifier to the rice dataset provided in the file `Rice.csv`. The dataset consists of various physical features of two types of grains known as 'Cammeo' and 'Osmancik'. The features include: Area, Perimeter, Major Axis Length, Minor Axis Length, Eccentricity, Convex Area, and Extent observed for different grains of rice from the two species. More details about the data is present in [1].

   We have provided you with a skeleton file called `classifier.py` with all the necessary libraries loaded. You will first preprocess the dataset, apply the classifier, and report any preprocessing you do and results (ROC curve and AUC). Your task is divided as follows:

Algorithm 1 Gradient Descent

Require: $\texttt{X} \in \mathbb{R}^{N \times D}, \texttt{w\_init} \in \mathbb{R}^{D}, \texttt{y\_true} \in \{0,1\}^{N}, \texttt{epochs} \geq 0, \texttt{lr} \geq 0$

```
w ← w_init
losses = [ ]
i ← 0
while i ≤ epochs do
    i ← i + 1
    y_pred ← predict_score(X, w)
    loss ← cross_entropy_loss(y_pred, y_true)
    losses.append(loss)
    w ← w − lr ∗ gradient(X,w,y_true)
end while
```

(a) Preprocess: Once the data is loaded, you need to convert the target variable `y` in to binary variable. You can choose either of the two species as class 1. If required, you can preprocess the dataset provided by the variable `X`. Since there are integers and real numbers, some of the preprocessing methods include:

i. Standardization: makes the numerical feature zero mean and unit variance.
ii. Min-Max Scaling: scales features to a specified range (e.g. [0, 1]).
iii. Log Transformation: can be used for features with skewed distributions to make them more normally distributed.

You can choose one of the above or anything else you can think of and apply it to the dataset.

(b) Split the data into training and test sets using the `train_test_split` function from the sklearn library.

(c) Now train the classifier using the training set and test the classifier using the test set. Report the accuracy, loss curve, ROC plot, and the Area Under the Curve (AUC) along with the math part of the assignment. You need to report:

i. The preprpocessing steps used,
ii. Parameters such as learning rate and epochs,
iii. Loss plot, ROC plot, accuracy, and Area under the Curve (AUC).

Exercise 5 (Convolution and Filtering - 2 points)

Convolution and filtering are fundamental operations when working with signals or images. We have provided you with a skeleton code in the file `filter.py` to perform basic 2D convolutions and some simple Gaussian low-pass and high-pass filtering.

1. Convolutions can be implemented in two ways, either in the Fourier domain or the image domain. In both cases, the function takes an image of size $K \times K$ and a filter of size $k \times k$ and `mode` as input. If `mode = 'same'` the output has the same dimension as the input image. In this case, we want to compute linear rather than circular convolution, so you will have to zero-pad the input image accordingly and then crop the result to the target dimensions. If `mode='valid'`, then you should only return that part of the output image that does not depend on zero padding. Thus the output is of size $(K-k+1) \times (K-k+1)$. Your task is to implement:

   (a) `convolve2d()`: performs convolution directly in the image domain, using for loops.

   (b) `convolve2d_fft()`: performs convolution in the Fourier domain.
   Note:

   i. Use NumPy's FFT routine.

ii. While performing convolution in the Fourier domain, you need to first perform the filtering and then crop the result.

(Optional) Compare computation time of `convolve2d` and `convolve2d_fft` on a large image of your choice.

2. A Gaussian low-pass filter retains only smooth features of the image while removing high-frequency, potentially noisy components. For a given standard deviation η, the filter is of spatial dimension $(2m+1)\times(2m+1)$ where $m = \lceil 4\eta \rceil$. The weights of the filter are defined as:

$$W^{\text{low-pass}}[n_1, n_2] = ce^{-(n_1^2+n_2^2)/(2\eta^2)},$$

where $n_1 \in \{-m, -m+1, \ldots, 0, \ldots, m\}$, $n_2 \in \{-m, -m+1, \ldots, 0, \ldots, m\}$ and c is chosen such that $\sum_{n_1=-m}^{m}\sum_{n_2=-m}^{m} W[n_1, n_2] = 1$. The high-pass filter is just the complement of the low-pass, $W^{\text{high-pass}}[n_1, n_2] = \delta[n_1, n_2] - W^{\text{low-pass}}[n_1, n_2]$, which allows only the high-frequency components of the image. The function $\delta[n_1, n_2]$ is an identity filter defined as

$$\delta[n_1, n_2] = \begin{cases} 1 & \text{if } n_1 = 0 \text{ and } n_2 = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Applying this filter doesn't change the image. Your task is to implement:

(a) `gaussian_lowpass()`: Which takes the standard deviation η as input and outputs a Gaussian low-pass filter of size $(2m+1)\times(2m+1)$ where $m = \lceil 4\eta \rceil$.

(b) `gaussian_highpass()`: Outputs a Gaussian high-pass filter.

(c) Using any one of the images in the scikit-image dataset (link), show the effect of a high-pass and low-pass filter by plotting the chosen image along with the low-pass filtered and high-pass filtered images side by side. Report the variance and filter size used. Report the convolution operation used and what is the computational complexity of this convolution?

Note: For a color image, make sure to convert it into a grayscale image first.

References

[1] Rice (Cammeo and Osmancik), UCI Machine Learning Repository, 2019, `https://doi.org/10.24432/C5MW4Z`.