Assignment 1: NumPy Introduction

Mario Döbler

Institute for Signal Processing and System Theory
University of Stuttgart

1 Introduction

NumPy (Numerical Python) is an open source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems. [com21b]

The NumPy library contains multidimensional array and matrix data structures, specifically **ndarrays**; a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices. [com21b]

If you're new to NumPy, we want to refer to the beginners guide `https://numpy.org/doc/stable/user/absolute_beginners.html` and also to the NumPy API reference `https://numpy.org/doc/stable/reference/index.html`.

2 When should you use Python lists and when should you use NumPy arrays?

Python's lists are efficient general-purpose containers. They support (fairly) efficient insertion, deletion, appending, and concatenation, and Python's list comprehensions make them easy to construct and manipulate. However, they have certain limitations: they don't support "vectorized" operations, like elementwise addition and multiplication, and the fact that they can contain objects of differing types mean that Python must store type information for every element, and must execute type-dispatching code when operating on each element. This also means that very few list operations can be carried out by efficient C loops – each iteration would require type checks and other Python API bookkeeping. [com21c]

Needless to say, NumPy arrays are faster and more compact than Python lists. An array consumes less memory and is convenient to use. NumPy also enables "vectorized" computations and operations that can be carried out by efficient C loops. Nevertheless, one thing you shouldn't do with NumPy arrays is extending an array in a loop; that would require creating a new array and copying the data from the previous array each iteration. If you know the shape

beforehand, create an empty array beforehand and simply fill it, e.g. using a loop.

3 Broadcasting

The term broadcasting describes how numpy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is "broadcast" across the larger array so that they have compatible shapes. Broadcasting provides a means of vectorizing array operations so that looping occurs in C instead of Python. It does this without making needless copies of data and usually leads to efficient algorithm implementations. [com21a]

For furhter details have a look at `https://numpy.org/doc/stable/user/basics.broadcasting.html`.

4 Tasks

In this task, we will implement the calculation of the confusion matrix, as introduced in the first chapter of the lecture, using NumPy. Use the provided code structure and simply fill in the gaps in `metrics.py`. You can run the code with `python3 main.py`.

NumPy functions that might be useful:

- `numpy.maximum` (`https://numpy.org/doc/stable/reference/generated/numpy.maximum.html`)
- `numpy.amax` (`https://numpy.org/doc/stable/reference/generated/numpy.amax.html`)
- `numpy.sum` (`https://numpy.org/doc/stable/reference/generated/numpy.sum.html`)

1. Implement the confusion matrix and its normalizations.
2. Based on the confusion matrix, derive the metrics precision, recall, and false alarm rate.

5 Movie Recommender System (optional and not relevant for the exam)

To get further experience with NumPy, we will look at a basic movie recommender system. This is not necessarily part of the *Detection and Pattern Recognition*, but it is a good example to apply NumPy. If you're interested in the details of the theory behind recommender systems, we recommend to attend the course *Matrix Computations in Signal Processing and Machine Learning* offered at our institute by Stefan Uhlich.

Recommender systems seek to predict or filter preferences according to a user's choices. They are utilized in a variety of areas including movies, music, news, books, products, and many more. In this assignment we will use collaborative filtering, which builds a model from a user's past behavior/ratings as well as similar decisions made by other users.

Given a ratings matrix $\mathbf{R} \in \mathbb{R}^{N \times M}$, our goal is to factorize the ratings matrix into the product of a user embedding matrix $\mathbf{U} \in \mathbb{R}^{N \times K}$ and a movie embedding matrix $\mathbf{V} \in \mathbb{R}^{M \times K}$, such that

$$\mathbf{R} \approx \mathbf{U}\mathbf{V}^T = \hat{\mathbf{R}} \text{ with } \mathbf{U} = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} \text{ and } \mathbf{V} = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix},$$

where N is the number of users, M the number of movies, and hyperparameter K denoting the number of factors. R_{nm} is the rating of the m-th movie by the n-th user and is 0 in case a rating does not exist and in the range 1 to 5 otherwise. The ratings matrix $\mathbf{R}$ is often very sparse in practice, meaning we have many zero-entries corresponding to unknown ratings. Matrix factorization "fills" these empty entries. The intuition behind using matrix factorization is that there should be latent features that determine how a user rates a movie. Latent features could be, for example, a movie genre, certain actors/actresses, and so on.

To obtain $\mathbf{U}$ and $\mathbf{V}$, we initialize them randomly and update them by minimizing a loss function iteratively with gradient descent. In our case the loss function L is the sum of squared errors

$$L = \sum_{\{nm\} \in \mathbb{TR}} (r_{nm} - \hat{r}_{nm})^2,$$

where $\mathbb{TR}$ is our train dataset, a subset of all known ratings. We can also formulate our objective in matrix form by introducing a mask $\mathbf{M_{tr}} \in \{0,1\}^{N \times M}$, which is 1 if the element is contained in our train dataset and 0 otherwise. We get

$$L = ||\mathbf{M_{tr}} \odot (\mathbf{R} - \mathbf{U}\mathbf{V}^T)||_F^2,$$

where $\odot$ denotes an element-wise multiplication. The corresponding gradients are

$$\nabla_{\mathbf{U}} = -2(\mathbf{M_{tr}} \odot (\mathbf{R} - \mathbf{U}\mathbf{V}^T))\mathbf{V} \quad \text{and} \quad \nabla_{\mathbf{V}} = -2(\mathbf{M_{tr}} \odot (\mathbf{R} - \mathbf{U}\mathbf{V}^T))^T\mathbf{U}.$$

If you're interested in the gradient calculation, come back after Chapter 4, by then you should be able to compute them.

Finally, our update rules look as follows:

$$\mathbf{U}' = \mathbf{U} - \alpha\nabla_{\mathbf{U}} \quad \text{and} \quad \mathbf{V}' = \mathbf{V} - \alpha\nabla_{\mathbf{V}},$$

where α denotes the learning rate. To measure the performance of our recommender system, we compute the root mean squared error on the test dataset $\mathbb{TE}$.

$$RMSE = \sqrt{\frac{1}{|\mathbb{TE}|} \sum_{\{nm\} \in \mathbb{TE}} (r_{nm} - \hat{r}_{nm})^2}$$

6 Tasks

Use the provided code structure and simply fill in the gaps in `recommender_system.py`. You can run the code with `python3 main.py`.

NumPy functions that might be useful:

- You can also apply relational operators to NumPy arrays. They're applied elementwise and you get an array of type boolean, whether the condition is true or false.
- `ndarray.astype` (`https://numpy.org/doc/stable/reference/generated/numpy.ndarray.astype.html`)
- `numpy.random.rand` (`https://numpy.org/doc/stable/reference/random/generated/numpy.random.rand.html`)
- `numpy.matmul` (`https://numpy.org/doc/stable/reference/generated/numpy.matmul.html`)

1. Derive the mask M from the recommendation matrix R and split it randomly into a train and test mask.
2. Implement the derivative calculation both for the user and movie factorized matrices.
3. Implement the update both for the user and movie factorized matrices.

Optional tasks:

- Use the factorized matrices to make recommendations.
- Additionally use regularization during training to reduce overfitting.
- Implement a sparse variant, which only calculates required entries.

References

[com21a] The SciPy community. Numpy: Broadcasting, 2021. Last updated 31 January 2021.

[com21b] The SciPy community. Numpy: the absolute basics for beginners, 2021. Last updated 31 January 2021.

[com21c] The SciPy community. Scipy: Frequently asked questions, 2021.