# Learning journal central document

## Key takeaways per chapter

## Homework 1:

* Machine learning is a branch of artificial intelligence focused on training algorithms to make predictions or decisions based on data.
* There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.
* Scikit-learn is a library that provides machine learning models and tools for data preprocessing, feature engineering, model selection, and evaluation.

## Homework 2

* Machine learning projects consist of various steps, such as data acquisition, data cleaning, data exploration and visualization, feature engineering, model selection, and evaluation.
* Data preprocessing involves dealing with missing values, outliers, encoding categorical variables, scaling numerical variables, and splitting data into training and testing sets.
* Data visualization can help understand the relationship between variables and detect patterns in the data.
* Often it is necessary to visualize the data to be able to build a comprehensive model

## Homework 3

* Linear regression is a powerful algorithm that predicts continuous values.
* Polynomial regression can detect nonlinear relationships between variables.
* How to avoid overfitting
* Logistic regression is a binary classification algorithm that forecasts the probability of a binary outcome.
* Softmax regression is a multiclass classification algorithm that predicts the probability of each class.

## Homework 4

* Decision trees are simple models that can handle both numerical and categorical data.
* Random forests are an ensemble of decision trees that can reduce overfitting and improve performance.
* Gradient boosting is another ensemble method that trains models sequentially, with each new model correcting the errors of the previous one.
* Support vector machines (SVMs) are binary classifiers that find the best separating hyperplane between classes.
* SVMs use kernels to transform data into a higher-dimensional space and identify nonlinear relationships between variables.
* Stochastic Gradient Descent, and Mini-Batch Gradient Descent

## Homework 5

What is a pre-trained NLP model?

A pre-trained NLP model is a type of machine learning model that has been trained on a large text corpus to learn the patterns and connections between words and phrases in natural language. These models can be utilized for various NLP tasks like sentiment analysis, named entity recognition, language translation, and more.

How do I load pre-trained NLP models?

There are multiple libraries and frameworks like Hugging Face's Transformers library, TensorFlow, PyTorch, etc., that you can use to load pre-trained NLP models. Depending on the library, you may have to download the model weights and configuration files and then load them into your code.

What is tokenization?

Tokenization is the process of dividing a piece of text into smaller units called tokens, such as words, subwords, or characters. Tokenization is an essential step in NLP tasks as it transforms text into a format that machine learning models can comprehend.

What does fine-tuning mean?

Fine-tuning is the process of taking a pre-trained NLP model and training it on a smaller, domain-specific dataset to adapt it for a particular task. Fine-tuning can improve the performance of the model on the target task since the model has already learned the general understanding of natural language from its pre-training.

What types of NLP models are there?

There are various types of NLP models, including Bag of Words models, Neural Networks (such as Recurrent Neural Networks and Convolutional Neural Networks), Sequence-to-Sequence models, Attention models, and Transformers (such as BERT, GPT-2, and T5).

What possibilities do I have with the Transformers package?

The Transformers package from Hugging Face is a widely-used and powerful library for working with pre-trained NLP models. Using the Transformers package, you can load and use pre-trained models for a wide range of NLP tasks, fine-tune pre-trained models on your own data, train your own models using the same architectures as popular pre-trained models, generate text using pre-trained models, visualize the attention and output of pre-trained models, and more.

How to use a transformer model in practise?

First, the input data needs to be preprocessed by splitting it into tokens, converting them into integer representations, and grouping them into batches. Depending on the model, special tokens and input masks might need to be inserted.

Next, the model processes the inputs and produces intermediate vector representations of the sentences.

Finally, the post-processing step involves converting the number representation of the final task result back into human-readable format, using different functions depending on the task and model.

The huggingface Transformers library provides a Tokenizer object for handling pre-processing and post-processing steps, as well as a Trainer object for fine-tuning models. Additionally, a pipeline can be created to handle all necessary steps from pre-processing to post-processing, as long as the model head is retained.

## Homework 6

Broadcasting is like a helpful assistant in numpy and PyTorch that makes working with arrays of different shapes much easier. It automatically adjusts the dimensions of arrays so they can be combined and computed together without any hassle.

Imagine you have two arrays with different shapes, and you want to perform some calculations on them. Broadcasting steps in and follows specific rules to make sure the arrays are compatible. It copies and stretches the values along the smaller dimensions, so they match the corresponding dimensions of the other array. This way, you can perform computations smoothly without needing to manually reshape or loop through the arrays.

The great thing about broadcasting is that it simplifies coding. It allows developers to write shorter and more readable code by eliminating the need for complicated loops or reshaping operations. So, when you're dealing with complex math operations, like working with matrices or performing calculations on each element, broadcasting comes to the rescue. It not only makes your code easier to understand but also improves its performance by avoiding unnecessary repetitive work.

In the world of deep learning models, broadcasting is especially valuable. It helps with various operations involving tensors of different shapes, such as calculating gradients or applying activation functions to each element. It also plays a key role in batch processing, where you can process multiple data samples at once, making the training process more efficient.

## Homework 7.1

1. What problem does collaborative filtering solve?

2. How does it solve it?

3. Why might a collaborative filtering predictive model fail to be a very useful recommendation system?

4. What does a crosstab representation of collaborative filtering data look like?

5. Write the code to create a crosstab representation of the MovieLens data (you might need to do some web searching!).

6. What is a latent factor? Why is it "latent"?

7. What is a dot product? Calculate a dot product manually using pure Python with lists.

8. What does pandas. DataFrame. merge do?

9. What is an embedding matrix?

10. What is the relationship between an embedding and a matrix of one-hot-encoded vectors?

11. Why do we need Embedding if we could use one-hot-encoded vectors for the same thing?

12. What does an embedding contain before we start training (assuming we're not using a pretained model)?

13. Create a class (without peeking, if possible!) and use it.

14. What does x[:,01 return?

15. Rewrite the DotProduct class (without peeking, if possible!) and train a model with it.

16. What is a good loss function to use for MovieLens? Why?

17. What would happen if we used cross-entropy loss with MovieLens? How would we need to change the model?

18. What is the use of bias in a dot product model?

19. What is another name for weight decay?

20. Write the equation for weight decay (without peeking!)

21. Write the equation for the gradient of weight decay. Why does it help reduce weights?

22. Why does reducing weights lead to better generalization?

23. What does argsort do in PyTorch?

24. Does sorting the movie biases give the same result as averaging overall movie ratings by movie? Why/why not?

25. How do you print the names and details of the layers in a model?

26. What is the "bootstrapping problem" in collaborative filterina?

27. How could you deal with the bootstrapping problem for new users? For new movies?

28. How can feedback loops impact collaborative filtering systems?

29. When using a neural network in collaborative filtering, why can we have different numbers of factors for movies and users?

30. Why is there an n. Sequential in the CollabNN model?

31. What kind of model should we use if we want to add metadata about users and items, or information such as date and time, to a collaborative filtering model?

1. Collaborative filtering solves the problem of personalized recommendation by predicting user preferences or interests based on the behavior and preferences of similar users or items.

2. It solves this problem by analyzing the historical user-item interactions or ratings to identify patterns and similarities among users or items. The idea is that users with similar tastes or preferences in the past are likely to have similar preferences in the future, and items that are preferred by similar users are likely to be preferred by the target user as well.

3. collaborative filtering relies on the availability of sufficient user-item interactions or ratings data. If there is sparse or insufficient data, it becomes challenging to accurately identify similarities and make meaningful recommendations.

Secondly, collaborative filtering suffers from the cold-start problem. When new users or items are introduced to the system with limited or no interaction history, it becomes difficult to establish meaningful recommendations as there is insufficient data to identify similarities.

Additionally, collaborative filtering tends to recommend popular items or rely on mainstream preferences, which may lead to a lack of diversity in recommendations. It can result in the model overlooking niche or less-known items that could be of interest to certain users.

7. can be use to calculate the similarity of user preferences to items in recommendation systems.

A dot product, also known as an inner product or scalar product, is a mathematical operation that takes two vectors of equal length and returns a scalar value. It is calculated by multiplying the corresponding elements of the two vectors and summing the products.

vector1 = [1, 2, 3]

vector2 = [4, 5, 6]

dot\_product = sum(x \* y for x, y in zip(vector1, vector2))

9. An embedding matrix is a matrix that maps discrete categorical variables, such as words or user IDs, to continuous vectors of fixed dimensions.

11. The use of embeddings over one-hot-encoded vectors offers several advantages. Firstly, embeddings can capture relationships and similarities between categories, as they are represented in a continuous vector space. This allows the model to generalize better to unseen data and handle out-of-vocabulary cases. Secondly, embeddings reduce the dimensionality of the input, making it more efficient to process and train models. Lastly, embeddings can be learned from the data itself, enabling the model to automatically derive meaningful representations for the categories.

This doesn't work for encoded cetegories, as they are only represented by binary 1 or 0.

18. In a dot product model, biases play an important role in capturing additional information and improving the model's ability to make accurate predictions.

The use of bias terms allows the model to account for factors that cannot be fully captured by the dot product of the latent factors alone. While the latent factors capture the underlying interactions between users and items, biases provide an additional adjustment or offset to the prediction based on specific user or item characteristics. In our case the bias is used to take the rating tendency and/or how good a movie is preceived to be in a general perspective.

12. randomly created tensors.

To initialize the embeddings, random values are assigned to each dimension of the embedding vector. The number of dimensions in the embedding vector is a hyperparameter chosen by the model designer, and it determines the expressive power of the embeddings. Typically, the dimensionality of embeddings can range from tens to hundreds, depending on the size and complexity of the dataset.

After initializing the embeddings, the recommendation model is trained using techniques such as matrix factorization or neural networks to optimize the embeddings based on the observed user-item interactions. The model learns to predict user preferences or item relevance based on the learned embeddings, and the embeddings are updated during the training process to improve the accuracy of the predictions.

23. The argsort function is particularly useful for tasks such as ranking or sorting elements based on their values. By utilizing the indices returned by argsort, you can retrieve the original tensor elements in sorted order or perform further operations based on the sorted indices. so you can find the coresponding indexes of a tensor soreted for example. This can be very useful to f.e. filter tensors to a ranking etc.

26. The "bootstrapping problem" in collaborative filtering pertains to the predicament of generating personalized recommendations for novel users or items with limited or no historical data available. Collaborative filtering algorithms heavily rely on past user-item interactions to make accurate predictions and suggestions. However, when confronted with a new user joining the system or the introduction of a fresh item, the absence of data or insufficient user-item interactions poses a significant challenge.

The quandary arises from the fact that collaborative filtering approaches typically rely on discerning patterns and similarities from prior interactions to infer user preferences. Without substantial historical data, accurately assessing the preferences and characteristics of new users or items becomes arduous, impeding the provision of relevant and personalized recommendations.

27. To ask the user questions to categorize him initially when he joins the platform, or initialize him with values for interests like the average person on the platform

## Homework 7.2

User-Based vs. Item-Based Collaborative Filtering: Collaborative filtering can be implemented in two main ways: user-based and item-based. The user-based approach recommends items based on users with similar tastes, while the item-based approach suggests items based on item similarities. The choice between these approaches depends on factors like dataset size, sparsity, and computational efficiency.

Scalability and Efficiency: Efficiently handling large datasets is crucial for collaborative filtering, given the numerous users and items in real-world systems. Optimization techniques, such as data partitioning, parallelization, and approximation algorithms, enhance scalability and efficiency.

Evaluation Metrics: Evaluating the performance of collaborative filtering algorithms is vital. Metrics like precision, recall, mean average precision, normalized discounted cumulative gain, and root mean square error are commonly used. The choice of metrics depends on the specific goals of the recommendation system.

Cold Start Problem: The cold start problem arises when new users or items join the system, lacking sufficient data for accurate recommendations. Collaborative filtering struggles in such scenarios. Hybrid approaches that combine collaborative filtering with content-based or knowledge-based methods help mitigate this issue by leveraging additional information.

Long-Tail Recommendations: Collaborative filtering can address the long-tail problem, where a few popular items overshadow niche items. Leveraging user preferences and item similarity, collaborative filtering algorithms recommend long-tail items, enhancing diversity and user satisfaction.

Sparsity Handling: Collaborative filtering often deals with sparse data, where users rate only a fraction of available items. This sparsity poses a challenge in predicting user preferences accurately. Techniques like matrix factorization, neighborhood-based methods, and dimensionality reduction address sparsity to improve recommendation quality.

Overcoming Bias and Personalization: Collaborative filtering can inadvertently reinforce biases present in the data, leading to homogeneous recommendations and excluding certain user segments. Techniques like diversity-aware recommendation algorithms and hybrid models help mitigate bias and enhance personalization.

## Additional Projects to the class assignments

## Project 1

Since we began with linear regression models, I chose to construct a linear regression model independently (assignment 3.1). However, I didn't want to use pre-existing data; instead, I desired a dataset that I could consistently update and extend. My interest in real-time financial data led me to build a Linear Regression model with the scikit library and yfinance. I aimed to predict the price of Bitcoin by assessing its correlation with other assets in the market.

To utilize the data consistently, I developed a database that stores the predictions, allowing me to create a trading bot that can make informed decisions based on the model's predictions. I also calculated the correlation between assets over various time periods, dynamically implementing a time range that adapts as I retrain the model. To diversify the assets, I included those with positive correlations, such as SPY to Bitcoin, those with negative correlations, such as Gold to Bitcoin, and a broad index of bonds. I utilized as many parameters as possible in the model to account for biased predictions due to the differences in base values of the assets, and converted the dataset to work on percentage change of the assets values.

Project 2

For my second project (assignment 3.2), I constructed a basic LSTM (long short term memory) neural network that predicted Bitcoin's price based on sequential analysis. I utilized historical data from Yahoo Finance via the yfinance library to focus specifically on BTC. I designed and trained an LSTM-based neural network model to predict future BTC prices based on past data. The model's efficiency was evaluated on a test set, and I created functions to determine the optimal training sequence length and visualize the model's predictions. The ultimate goal of this project was to develop a model that can accurately predict BTC prices based on historical data, potentially allowing for more informed cryptocurrency trading decisions.

### Project 3

As a third project (assignment 3.3), I used to create a linear regression model and a random forest to predict the probability of survival for passengers of the titanic. Firstly, I analyzed the dataset, whereas already some clear patterns emerged, converted the data, and selected a couple features for the models. I was quite happy with my third project, although I tried to keep it simple, as it was mainly a review of last week’s lesson.

## Project 4

In the context of my fourth project, specifically assignment 4.1, I employed a pretrained Convolutional Neural Network (CNN) to detect facial features within images sourced from an online platform characterized by rating-based interactions, commonly known as "hot or not." By extracting landmark coordinates from the detected faces, I computed scores based on various facial metrics, including symmetry and adiposity. These metrics served as inputs for constructing a linear regression model aimed at predicting the ratings associated with individuals, as inferred from the aforementioned facial landmarks. To delve deeper into the project and gain comprehensive insights, I encourage perusal of the associated presentation on CLIP and the Tinder-bot, which elaborates further on its intricacies and implications.

### Project 5

In the context of the fifth project, the Tinder API endpoint was reengineered, and a multifaceted bot was developed, incorporating several pretrained neural networks to facilitate swiping and communication within the Tinder application. The deployment of this bot was executed through a docker container within an Ubuntu runtime environment. For further insights into the project and its intricacies, a presentation was delivered jointly by myself and Damiano, shedding light on the various aspects and dimensions of the aforementioned undertaking.

### Project 6

I implemented a version of the Tinderbot, which used a cocoa model, in order to predict what can be found in the image.   
The Cocoa model typically consists of two main components: an image encoder and a language decoder.

1. Image Encoder: The image encoder is responsible for processing the input image and extracting high-level features that capture its visual content. This is typically achieved using a convolutional neural network (CNN). The CNN processes the image through multiple layers of convolution and pooling operations, gradually reducing the spatial dimensions while preserving important visual features. This process results in a compact representation of the image that encodes its visual content.
2. Language Decoder: The language decoder takes the encoded image representation from the image encoder as input and generates a textual description in the form of a caption. This component usually employs a recurrent neural network (RNN) or a transformer-based architecture. The RNN or transformer model processes the encoded image features in a sequential manner, generating words or tokens one at a time. At each step, the model predicts the most probable next word given the previously generated words and the image context. This process continues until an end-of-sentence token is generated, indicating the completion of the caption.

Training the Cocoa model involves leveraging large datasets of images paired with their corresponding captions. During training, the model learns to associate the visual features extracted from the image with the textual descriptions. This is achieved by minimizing the discrepancy between the predicted captions and the ground truth captions using techniques like maximum likelihood estimation or reinforcement learning.