Introduction to Data Science — Assignment 4

Instructors: **Dr. Bahrak, Dr. Yaghoobzadeh**

TAs: Javad Kavian, Mohammad Reza Mohammad Hashemi, Amir Mahdi Farzaneh

Deadline: 6th Khordad

Introduction

This assignment is the fourth exercise of the “Data Science” course at the University of Tehran, designed to enhance students’ understanding of deep learning concepts and equip them with practical skills for tackling real-world challenges. The exercise consists of three distinct projects that cover different domains of machine learning, each aimed at developing expertise in designing, implementing, and evaluating deep learning models. In the first project, students will work with football match data to design and train a Multi-Layer Perceptron (MLP) model to predict the outcomes of matches. By analyzing data from the qualifiers and the Qatar 2022 FIFA World Cup, the model will learn patterns and be used to forecast the results of the World Cup tournament. The second project focuses on image classification, where students will build a Convolutional Neural Network (CNN) to classify flower images. They will design a VGG-style CNN from scratch and fine-tune a pre-trained ResNet model, comparing their performance to understand the benefits and trade-offs of each approach. In the third project, students will use Recurrent Neural Networks (RNNs) to predict Bitcoin prices based on historical data. They will preprocess time-series data, experiment with architectures such as RNN and LSTM, and evaluate their ability to capture trends and forecast future movements in Bitcoin prices. Each project provides students with hands-on experience in essential processes like data preprocessing, model design, hyperparameter tuning, and performance evaluation, helping them build a strong foundation in deep learning and data science.

Task 1 : MLP

In this task, you are supposed to work with a football dataset! This dataset contains the match result and game status of road to Qatar 2022 FIFA World Cup Qualifiers along with the actual matches of Qatar 2022 and you are going to train a Multi Layer Perceptron to predict the result of games. Finally you are going to use your model to predict the entire tournament of Qatar 2022.

1-Dataset Loading

First, let's explain the dataset. This dataset is consisted of multiple attributes which are to be explained:

1. home_team: The national team designated as the home team for the match.
2. home_goals: The number of goals scored by the home team in the match.
3. away_goals: The number of goals scored by the away team in the match.
4. away_team: The national team designated as the away team for the match.
5. wcm: Stands for "World Cup Match" — this is a binary flag (1 or 0) indicating whether this match is part of the actual World Cup tournament or just a qualifier match.
6. dif_inter_match: Difference in the total number of international matches played by the home team minus the away team.
7. dif_inter_match_won: Difference in the number of international matches won (home - away).
8. dif_inter_match_lost: Difference in the number of international matches lost (home - away).
9. dif_inter_match_tie: Difference in the number of international matches tied (home - away).
10. dif_inter_goals_sco: Difference in goals scored in international matches (home - away).
11. dif_inter_goals_con: Difference in goals conceded in international matches (home - away).
12. dif_wc_match: Difference in the total number of World Cup matches played (home - away).
13. dif_wc_match_won: Difference in the number of World Cup matches won (home - away).
14. dif_wc_match_lost: Difference in the number of World Cup matches lost (home - away).
15. dif_wc_match_tied: Difference in the number of World Cup matches tied (home - away).
16. dif_wc_goals_sco: Difference in goals scored in World Cup matches (home - away).

17. dif_wc_goals_con: Difference in goals conceded in World Cup matches (home - away).
18. status: The result of the match, typically encoded as:
    a. 1: home won
    b. 2: tie
    c. 3: home loss

Now let's dive into the steps of this task one by one.

3- Exploratory Data Analysis

In the Exploratory Data Analysis (EDA) phase, start by examining the dataset to understand its overall structure, including the number of rows and columns, data types, missing values, and the distribution of features. Analyze the distribution of match outcomes to identify whether the classes are balanced or imbalanced. Investigate numerical features, such as differences in the number of matches played or goals scored, to uncover potential relationships with match results. Utilize visualizations like histograms, box plots, and correlation matrices to gain deeper insights. Analyze team performance in home and away matches to validate logical patterns in the dataset. As suggested, ensure missing values are handled and exclude features that directly determine the outcome (e.g., goals scored) to avoid data leakage. Finally, check whether numerical features require standardization or normalization to ensure the data is properly prepared for training the model.

2-Data Preparation

First you need to prepare this data to feed it to a model. Load the dataset as a pandas dataframe. The process of training the model is on games whose wcm attribute is 0 because we want to have a final prediction on the winner of the World Cup, we need the world cup matches' data to remain unseen for our model.

Then we will need to select X and y. X is our set of features and y is the label which is to be predicted. In the context of this dataset, X will be all columns except the following columns:

1. home_team
2. home_goals
3. away_goals
4. away_team
5. status

and y will be the status column. The reason for which we omitted the above columns is obvious for columns home_goals and away_goals because the result of the match is specified if we know the number of goals scored by home and away sides. On the other hand, we omit name of teams from our features because we want to force our model to learn the patterns of the match according to its status; otherwise it will be biased to name of

teams; For example if Germany has more wins than Spain, the prediction of the model will be Germany without considering the other criterias.
Then we need to encode labels. You can use sklearn's label encoder for this phase.
Then split your data into train and test sets using a test portion of 30%.
The next thing you should do is standardize features. You can use StandardScalar or sklearn for this phase.

- Warning! One important thing to note here is to avoid data leakage from test set to train set. The StandardScalar should be fit only to train data and fitting the scalar to train and test set jointly will cause leakage.

Then you should convert the data to pytorch tensor. At this phase, your data is ready to be fed to a neural network model.

3- Model Definition

Now you should write your model class. Note that you are only allowed to use pytorch or tensorflow library (but we suggest pytorch) to define and train your model. Your model class should inherit from torch.nn. Module and should implement two methods __init__ and forward. The arguments of init depends on how you define the model. But note that the forward method should always have input X as an argument.
The way you define your MLP is arbitrary and you can use as many layers as you want.

4- Model Training

Let's have a brief description over training a pytorch model for classification. We need to define a set of tools for training a pytorch model.
First you need to specify the criterion by which you want to train your model. For multi class classification, torch.nn.CrossEntropyLoss is a proper choice.
Then you will need an optimizer. Optimizer is instantiated with two important arguments; model.parameters() and learning rate.
Now let's talk about the training loop; you should repeat the training loop for a fixed number of epochs.
In the training loop, you need to clear the gradient cache of the optimizer to avoid summing the gradient of the previous loop with the current loop; this can be achieved by the zero_grad() method of the optimizer.
Then you need to pass the batch of training data to the model and calculate the output; having calculated the output, you can calculate the loss, using the criterion. Then using the backward() method of loss tensor, you will have the gradients calculated at the computation tree of model parameters. Then by calling the step() method of the optimizer, one step of optimization will be taken and repeating this loop, the model will be trained.

5- Evaluating Model

After training the model, you need to evaluate its performance on the test set. Accuracies higher than 50 will achieve full marks. We really don't expect our model to achieve higher than 90% accuracies like other datasets because football is so unpredictable!

6- Run FIFA World CUP!

In this part, you are going to use your model to predict the tournament including group stage and knockout stage. As you know, the first two teams of each group are qualified for the knockout stage. Don't forget to use the real groups of Qatar 2022 as your initial configuration:

Task 2 : CNN

Image classification is a key task in computer vision, but raw images often need preparation before training a model. Differences in size, lighting, and angles can confuse the model and hurt its performance. In this project, you'll use the Flowers Multiclass Dataset from Kaggle to classify flower species. Preprocessing ensures all images are in a consistent format, which helps the model learn effectively. Data augmentation creates more varied training examples, making the model better at handling new images. You'll build a VGG-style CNN from scratch and fine-tune a pretrained ResNet, comparing how these approaches affect accuracy and reliability.

1. Dataset Loading:

Get the Flowers Multiclass Dataset from Kaggle and load it into your coding environment.

- Step 1: Sign up for a Kaggle account if you don't have one. Go to the Flowers Recognition dataset page and click "Download," or use the Kaggle Hub: :

```
import kagglehub
path=kagglehub.dataset_download("alsaniipe/flowers-multiclass-datasets")
print("Path to dataset files:", path)
```

- Step 2: Unzip the downloaded file and load it into Python using PyTorch or TensorFlow.
- Step 3: If the dataset isn't split, divide it into training (70%), validation (10%), and test (20%) sets.

2. Image Preprocessing:

Prepare the images and labels so they're ready for training.

- Resize: Make all images 224x224 pixels (a common size for CNNs like VGG or ResNet).
- Normalize: Scale pixel values to [0,1] by dividing by 255.
- Labels: Choose between two label formats and experiment to see which works better:
    - One-hot encoding: Convert labels to one-hot vectors (e.g., class 2 of 5 becomes [0, 0, 1, 0, 0]).
    - Label encoding: Use class indices directly (e.g., 0, 1, 2 for each class).
      Note: When choosing a label format, ensure your loss function matches.

3. Data Augmentation:

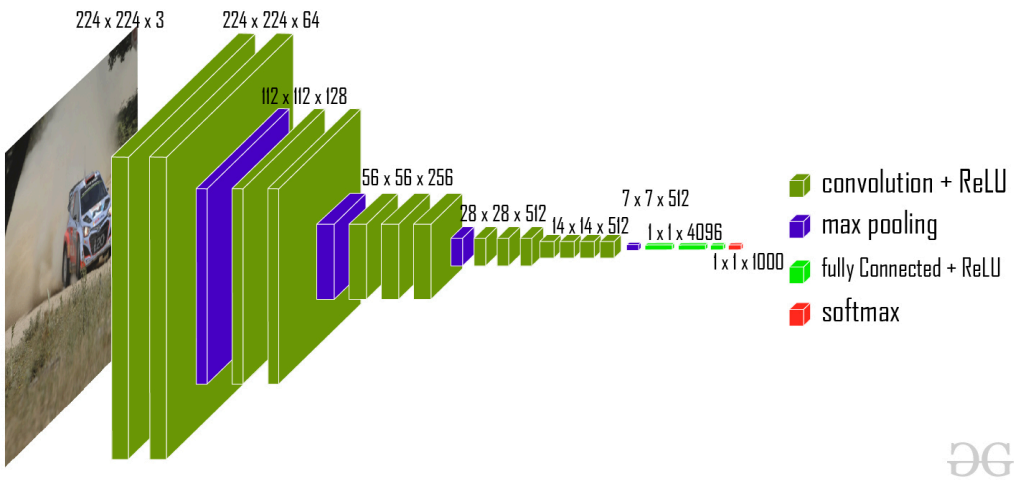
Add variety to the training set only (not validation) to help the model generalize.

- Suggested transformations:
  - Random rotations between -20° and 20°.
  - Horizontal flips (50% chance).
  - Random crops or shifts (e.g., 10% of image size).
  - Slight brightness or contrast changes.
  - other transformations that you found suitable.

4. Building a VGG CNN from Scratch:

Create and train a VGG-style CNN.

- VGG Basics: VGG is a deep network with stacked convolutional layers (e.g., 3x3 filters), max-pooling layers, and fully connected layers at the end. bellow is the VGG-16 architecture also you can use this helper :



- Define the model.
- Train it on the preprocessed and augmented training data for appropriate epochs.
- Evaluate it on the validation set.

5. Fine-Tuning a Pretrained ResNet:

Use a pretrained ResNet (e.g., ResNet50) and fine-tune it in stages.

- Setup: Load ResNet50.

- Freeze Base, Train Head: Freeze all convolutional layers and replace the final layer(s) with a new classifier for flower classes. Train for 5-10 epochs.
- Unfreeze Last Conv Layer: Unfreeze the last convolutional block (e.g., layer4 in PyTorch ResNet50) and train it with the classifier for 5-10 epochs.
- Unfreeze All: Unfreeze all layers and train the whole network for 5-10 epochs.
- Evaluate each method on the validation set.

6. Result Comparison:

Compare the VGG and ResNet models fairly.

- Calculate metrics: accuracy, precision, recall, and F1-score.
- Plot confusion matrices and ROC (Receiver Operating Characteristic) curves using matplotlib or seaborn for deeper insights. Calculate and report the AUC (Area Under the Curve) score for each model. (The ROC curve illustrates the diagnostic ability of a binary classifier by plotting the true positive rate against the false positive rate at various threshold settings, while AUC quantifies the overall performance of the classifier as a single value between 0 and 1, where a higher value indicates better discrimination ability.)

Additional Notes:

- Pick PyTorch or TensorFlow—both work well here.
- Training ResNet fully can take time; use a GPU if possible (e.g., Google Colab's free GPU).
- Play with settings like learning rate (start with 0.001), batch size (e.g., 32), or optimizers (e.g., Adam).
- Track training with loss/accuracy plots (matplotlib) and stop early if the model overfits (e.g., use EarlyStopping in TensorFlow).

Task 3 : RNN

Time series forecasting plays a vital role in many domains, particularly in finance, where anticipating future trends based on historical patterns is crucial for informed decision-making. In this project, you will use a Recurrent Neural Network (RNN) to forecast future Bitcoin prices. RNNs are specifically designed to handle sequential data, allowing them to capture temporal dependencies in financial time series.

You will begin by exploring the historical OHLCV (Open, High, Low, Close, Volume) data and selecting a target feature (specific definition of target mentioned in the data processing section) that serves as an indicator of potential profit or loss. This selected feature will then be used as the prediction target. The goal is to train an RNN model that learns from past sequences and can forecast the target value for future time steps, helping to identify favorable or risky market movements.

Attention: You must analyze each plot in the EDA and Visualization sections.

1- Dataset Loading

Start by loading historical Bitcoin data from the dataset file:

- Dataset name: `BTC-USD.csv`
- Source: The file contains daily Bitcoin prices with columns: Open, High, Low, Close, and Volume.
- Format: Load the data using Pandas or an equivalent tool.
- Explain what each feature is and how calculated.

2- Exploratory Data Analysis (EDA) on OHLCV:

Before training your model, perform exploratory analysis on the full OHLCV dataset to understand trends and relationships.

- Visualize Price Movement: Plot each column (Open, High, Low, Close, Volume) over time.
- Check Correlations: Calculate correlations between columns to see how they influence one another.
- Statistical Summary: Review metrics like min, max, mean, and standard deviation for these features.

3- Data Processing:

Prepare the data from the `BTC-USD.csv` file , focusing on designing a meaningful prediction target and constructing time-aware input sequences.

- Handle Missing Data: Identify and resolve missing or anomalous values.
- Normalization: Normalize features at your discretion.
- Feature Engineering: Using a combination of the OHLCV features , define a custom target that serves as a strong indicator of profit or loss over time. This target could represent a future change or a signal reflecting expected market movement, and it may be a mathematical combination of the extracted features. Your chosen target should be clearly justified based on your initial analysis.
- Sequence Creation: Transform the time series into a supervised format by creating sequences of historical values. For each sample, use a sequence of past data to predict the target at the next time step. This prepares the data for RNN training, where each input sequence represents past market behavior.
- Lookback Period Tuning: The lookback period—how many previous time steps the model considers—greatly affects performance. Try different values (30, 60, 90 days) and evaluate the impact by training models with each setting. This helps identify the optimal window size that captures the right amount of historical context.
- Split the dataset into training, validation, and test sets: Use the validation set for hyperparameter tuning and model selection, and use the test set for final evaluation. In your opinion, why using both a validation and test set is better than relying on just a test set?

4- Model Architecture:

Design a Recurrent Neural Network (RNN) to predict the future value of your target.

- Base Structure: Start with standard RNN layers to model the temporal patterns in your input sequences.
- Output Design: Add one or more fully connected layers after the recurrent layers to produce a single numeric prediction as the model's output.
- Dropout: Incorporating dropout and other regularization techniques is recommended to mitigate overfitting and improve the model's generalization performance.
- Loss and Optimization: Select an appropriate loss function for regression tasks and an efficient optimizer to guide the learning process.

5- Model Training:

Train your RNN model using the prepared training data and evaluate its ability to forecast the selected target.You can use TensorFlow or Torch for implementing your model.

- Training Process: Feed the sequential input data into the model and let it learn to minimize the prediction error over time.
- Monitoring: Track the loss throughout the training process to ensure stable convergence. Watch for signs of underfitting or overfitting.
- Hyperparameter Tuning: Experiment with different model configurations—such as the number of recurrent units, number of layers, batch size, and learning rate—to improve performance. These adjustments can significantly influence the model's ability to generalize to unseen data.

6- Evaluation and Visualization:

Once your model is trained and tested, analyze its performance both numerically and visually to draw meaningful conclusions about its predictive ability.

- Metric reporting: Report key regression metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) on the test set. Discuss the use case of each.
  Next, report the Mean Absolute Percentage Error (MAPE), which expresses the error as a percentage of the actual values, making it especially useful when the target variable's scale varies.
  Finally, report the Cumulative Error (CE), defined as the sum of the differences between actual and predicted values across all test samples. This metric provides insight into the total prediction bias.
- Prediction vs Actual Plot: Visualize the predicted values against the actual target values over time for train and test sets. This comparison helps identify how well the model captures the general trend and turning points in the data.
- Error Trends: Plot the prediction error over time to highlight specific intervals where the model struggles. Analyzing these moments can lead to valuable insights for future refinement.

7- Implementing an LSTM-Based Model (10% Bonus):

As an extension to the standard RNN approach, you may optionally implement an LSTM (Long Short-Term Memory) network. LSTMs are a more advanced type of recurrent

neural network designed to capture long-term dependencies and mitigate issues like vanishing gradients.

- Model Architecture: Replace the base RNN layers with one or more LSTM layers. You can adjust the number of units, stack multiple layers, or add dropout for regularization.
- Training and Evaluation: Train your LSTM model using the same data preparation and sequence setup. Like RNN do Evaluation and Visualization part and then compare its performance—both in terms of metrics and visualizations—with your standard RNN model.
- Insight: Reflect on whether the LSTM improved predictive accuracy or generalization. Discuss possible reasons based on the structure of the data and your model design.

Questions(10% Bonus)

1 - Under what conditions is MLP equivalent to Logistic Regression?

2 - Suppose you have a CNN that begins by taking an input image of size 28 × 28 × 3 and passing through a convolution layer that convolves the image using 3 filters of dimensions 2 × 2 × 3 with valid padding.

A. How many learnable parameters does this convolution layer have?
B. Suppose that you instead decided to use a fully connected layer to replicate the behavior of this convolutional layer. How many parameters would that fully connected layer have?

3 - Explain mathematically the vanishing gradient problem in Recurrent Neural Networks (RNNs). Then, analyze how changing the lookback window size impacts the severity of this phenomenon.

Notes

- Upload your work as a zip file in this format on the website: DS_CA4_[Std number].zip. If the project is done in a group, include all of the group members' student numbers in the name.
- Only one member must upload the work if the project is done in a group.
- We will run your code during the project delivery, so make sure your results are reproducible.
- Only traditional machine learning models are allowed; the use of any neural network models to solve the problem is strictly prohibited.