Introduction to Data Science — Final Project Phase 3

Instructors: **Dr. Bahrak, Dr. Yaghoobzadeh** TAs: محمد امانلو، محمد امین یوسفی، محمدرضا محمدهاشمی، حمید سالمی، متین بذرافشان، امیرمهدی فرزانه

Deadline: 17th Khordad

Introduction

In Phase 3 of the project, you will take the final step in your data science journey by applying all the previous work to develop a model that makes predictions based on the relationships within your dataset. The primary task in this phase is to build and deploy a data-driven model that leverages the features and structure of your data. Your model should be designed to predict a specific target variable based on the engineering and preprocessing steps you completed earlier.

This phase will involve a final evaluation of your pipeline's functionality and an assessment of the model's ability to generate meaningful and actionable predictions. Additionally, you will be required to assess the performance of your model using appropriate metrics and communicate the results in a clear and concise manner.

Tasks and Requirements

Section 1: Model Development and Task Definition

In this section, you need to:

1. **Model Selection:**
   Choose a model or algorithm suitable for your task. To ensure the best model, it's essential to test multiple models and determine which one works best. The choice of models to test, how to conduct the testing, and how many models to evaluate should be coordinated with your mentor. Depending on the discussions with your mentor, one of the models tested may even become your final model. You can use any library of your choice for this task. A brief list of suggested models for each task is provided at the end of the exercise in the attachment section.
2. **Data Splitting:**
   Split your dataset into training and validation and testing sets (e.g., 70/10/20 or 80/10/10) to ensure you can evaluate the model's performance on unseen data.

Section 2: Model Training and Evaluation

1. **Model Training:**
   Train the model using the training data. Ensure that you use proper hyperparameter tuning (if applicable) and techniques like cross-validation, regularization to prevent overfitting.
2. **Evaluation Metrics:**
   In this step, you need to make predictions with the model you built in the previous step on your test data. Then, evaluate the model's outputs and record the results. Choose the appropriate metrics based on the specific requirements of your project. A brief list of suggested metrics for each task is provided at the end of the exercise in the attachment section.

Section 3: Integration into the Data Pipeline (End-to-End Automation)

In this section, your goal is to integrate the trained model into the existing pipeline from Phase 2 and automate the entire process, ensuring smooth execution of each stage in your workflow. The pipeline should consist of two main parts: one for training the model and another for making predictions. The two pipelines will work together to automate both the training and prediction tasks, with the final predictions being saved back to the database.

1. Choosing the best method for your pipeline

To ensure that the entire process, from data loading to predictions, can be executed with a single command or function call, here are three approaches you can use for automation. You can choose one of these methods based on your preference and project requirements.

1. Writing run_pipeline.py:

This is a simple approach where you automate the entire workflow by creating a `run_pipeline.py` script that calls each individual script in sequence. This script will execute all necessary steps: data loading, preprocessing, feature engineering, model training, and generating predictions. This method is suitable for a straightforward and quick automation setup, especially for small projects or internal use.

2. GitHub Actions for CI/CD:

For a more integrated and scalable approach, you can use GitHub Actions to set up continuous integration/continuous deployment (CI/CD) workflows. This ensures that

the pipeline runs automatically whenever there are updates to the repository (e.g., on commits or pull requests). Benefits:

- Automates pipeline execution on every push or pull request.
- Ensures that the pipeline is tested and run in a controlled environment every time code is updated.
- Great for team collaborations and deployment to production.

This method is ideal if you need a robust, cloud-based CI/CD pipeline that integrates with version control systems like GitHub.

3. Using Workflow Automation Tools (Apache Airflow or Prefect):

For advanced orchestration, you can use tools like Apache Airflow or Prefect to define and manage your pipeline in a more structured way. These tools allow you to create Directed Acyclic Graphs (DAGs), where each node represents a task in the pipeline, and dependencies are automatically managed. Benefits**:**

- Visualize and monitor your workflow in real-time.
- Schedule tasks to run at specific intervals, enabling periodic or delayed execution.
- Handle complex dependencies and retries, making it easier to maintain the pipeline.

You have three methods to choose from for automating your pipeline. Each method has its strengths, and you can select one based on your project needs:

1. For a simple, single-command execution:
   Choose Method 1 (Writing `run_pipeline.py`) if you need a straightforward approach to automate your pipeline with minimal setup.
2. For seamless cloud-based automation with CI/CD integration:
   Choose Method 2 (GitHub Actions) if you want to automate your pipeline and integrate it into your Github repository with a robust CI/CD workflow.
3. For advanced orchestration and monitoring of complex workflows:
   Choose Method 3 (Apache Airflow or Prefect) if you need advanced scheduling, visualization, and monitoring features for large-scale or complex projects.

2. Pipeline stages

After you choose one of the above methods to create a pipeline, you need to define the following stages in it. In fact, you should have two separate pipelines: one for training your

model and one for making predictions with your model. These pipelines will include the following steps:

1. Data Loading:

- **Training Pipeline:**
  Load the training data from the database into Pandas DataFrames (or other relevant data structures). This can be automated through a script such as `load_data.py`, which connects to the database and fetches the data needed for model training.
- **Prediction Pipeline:**
  Similarly, for the prediction phase, load new or test data from the database that will be used for making predictions. Use a script like `load_data.py` to fetch the data and prepare it for the model.

2. Data Preprocessing:

- **Training Pipeline:**
  Perform all necessary preprocessing steps on the training data, such as handling missing values, encoding categorical variables, normalizing or standardizing numerical data, and other transformations. This should be automated using a script like `preprocess.py` to ensure that the training data is cleaned and ready for feature engineering and modeling.
- **Prediction Pipeline:**
  The same preprocessing steps should be applied to the new or test data. Ensure that the same transformations used during training (such as feature scaling or encoding) are applied to the test data to ensure consistency when making predictions.

3. Feature Engineering:

- **Training Pipeline:**
  After preprocessing, generate the necessary features for the model. This might involve creating new features, scaling, encoding, or other transformations. Automate this step using a script like `feature_engineering.py` to ensure the features are generated consistently for training.
- **Prediction Pipeline:**
  The same feature engineering steps used for the training data should be

applied to the test or new data before making predictions. This ensures that the input data for the model is consistent with what the model was trained on.

4. Modeling (Training):

- **Training Pipeline:**
  Train the model using the prepared data. This includes selecting the appropriate model, fitting it with the training data, and optionally fine-tuning the model. The training process should be automated through a script like `train_model.py`. Once the model is trained, save the trained model for later use.
  Important: The trained model should not be re-trained on the prediction data. Instead, it should be saved once it is finalized, and used for inference in the prediction pipeline.
- **Prediction Pipeline:**
  After the model has been trained and saved, it should be loaded into the prediction pipeline. The trained model should be used to make predictions on the test or new data. This process is automated through a script like `make_predictions.py`.

5. Prediction:

- **Prediction Pipeline:**
  Once the model is trained and saved, use it to generate predictions on new or test data. After making predictions, ensure that the predicted values are saved back into the database. This can be done through a script like `predictions.py`, which saves the predictions to the relevant database table for future analysis.

  Important: You must save your final predictions back to the database. This ensures that your predictions are stored and accessible for later use, whether for evaluation or for use in production applications.

Using MLFlow for Model Management (10% Bonus Section)

As a bonus, we highly recommend integrating MLFlow into your project. MLFlow is a powerful open-source platform designed to manage the end-to-end machine learning lifecycle, including experiment tracking, model versioning, and deployment.MLFlow allows you to:

- Track and log your machine learning experiments (parameters, metrics, models).

- Version control models.
- Serve models for production deployment.
- Compare multiple models and evaluate them based on metrics.

Steps to Use MLFlow:

1. **Install MLFlow**:
   To begin using MLFlow, you need to install it on your system. You can install MLFlow easily using `pip`. This will allow you to access all the features MLFlow offers, such as tracking experiments, versioning models, and deploying them.

2. **Track Experiments**:
   Once you have MLFlow installed, you can use it to track the progress of your model training. This includes logging important information such as the model, its hyperparameters, and evaluation metrics. You will need to start a run using `mlflow.start_run()`. During the training process, you can log the model, the hyperparameters you're using (e.g., number of estimators, model depth), and the evaluation results (such as accuracy) once the model has been tested on validation data. This process helps you track every aspect of your model's training and performance over time.
3. **Save and Load Models**:
   MLFlow makes it easy to save and load models. After training your model, you can log it so that it is saved in the MLFlow model registry. When you want to use the model again, you can load it from the registry using a unique identifier or run ID. This is important for reproducibility and for using the model at a later time, either in production or for further experimentation.
4. **Model Versioning**:
   One of the key features of MLFlow is automatic model versioning. Every time you log a model using MLFlow, it keeps track of its version. This is extremely useful when you are experimenting with different models and want to compare their performance over time. By having version control for your models, you can track how different versions perform with various datasets or hyperparameters.
5. **Access MLFlow UI**:
   MLFlow offers a web-based user interface (UI) to visualize your experiment results. To access it, you can start a local server. The UI allows you to review the details of all the experiments you've run, compare different models and their respective metrics, and analyze hyperparameters. It serves as a helpful tool for visually inspecting how your models and experiments are progressing.
6. **Integrate MLFlow into Your Pipeline**:
   To make your machine learning pipeline fully automated and trackable, you can

integrate MLFlow into every step of the pipeline. This means logging data loading, preprocessing, model training, and prediction steps. By doing so, you can ensure that every part of your process is documented and reproducible. This includes capturing metrics, model versions, and any changes in your pipeline.

By following these steps, MLFlow helps you organize, track, and manage the complete machine learning lifecycle, from data preprocessing and model training to deployment and monitoring.

Attachments

1. Below is a brief list of the main and well-known models for this type of task. You may use any of the models listed below, although models outside of this list are also acceptable.
    - **Classification:** Logistic Regression, Decision Trees, Random Forests, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Neural Networks
    - **Regression:** Linear Regression, Lasso, Ridge, Decision Trees, Random Forests, XGBoost, Neural Networks.
    - **Time Series:** ARIMA, Prophet, LSTM (Long Short-Term Memory networks), RNNs (Recurrent Neural Networks), GRUs (Gated Recurrent Units).
    - **Clustering:** K-Means, DBSCAN, Hierarchical Clustering, Gaussian Mixture Models (GMM).
    - **Recommendation System:** Collaborative Filtering, Matrix Factorization (e.g., SVD, ALS), Neural Collaborative Filtering (NCF).
    - **Natural Language Processing (NLP):** Bag-of-Words (BoW), TF-IDF (Term Frequency-Inverse Document Frequency), Word2Vec, GloVe (Global Vectors for Word Representation), BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pretrained Transformer), Transformer Models (e.g., T5, RoBERTa), LSTM for Text Classification/Sequence Prediction, Attention Mechanisms, Named Entity Recognition (NER), Topic Modeling (e.g., LDA - Latent Dirichlet Allocation), every kinds of LLMs.
    - **Image Generation and Prediction Models:** Generative Adversarial Networks (GANs), DCGAN (Deep Convolutional GAN), CycleGAN, StyleGAN, BigGAN, Variational Autoencoders (VAEs), Convolutional Neural Networks (CNNs) for Image Classification and Prediction, ResNet (Residual Networks), Inception Networks, YOLO (You Only Look Once) for Object Detection, U-Net for Image Segmentation, Mask R-CNN for Instance Segmentation, Fast R-CNN for Object Detection.

- **Neural Networks:** Feedforward Neural Networks (FNN), Convolutional Neural Networks (CNN) for image and text processing, Recurrent Neural Networks (RNN) for sequence data (text, time series), Autoencoders, Deep Neural Networks (DNN) for complex tasks, Capsule Networks for more accurate image recognition and classification, Long Short-Term Memory Networks (LSTM) for sequential data and time-series predictions, Transformers, Attention-based Models (like BERT).

2. Below is a brief list of the main and well-known metrics for some types of task. You may use any of the metrics listed below, although metrics outside of this list are also acceptable. (such as LLM as judge)

- For **classification** tasks, use metrics such as accuracy, precision, recall, F1-score, confusion matrix, AUC-ROC, ROC curve, log loss, Matthews correlation coefficient (MCC), Cohen's Kappa, balanced accuracy.
- For **regression** tasks, use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared (R^2), Mean Absolute Percentage Error (MAPE), Explained Variance Score, Mean Squared Logarithmic Error (MSLE), Huber Loss, Adjusted R^2.
- For **time series** tasks, use metrics such as Mean Absolute Percentage Error (MAPE), Root Mean Squared Logarithmic Error (RMSLE), Symmetric Mean Absolute Percentage Error (sMAPE), Mean Absolute Scaled Error (MASE), Forecast Bias (FBias), Theil's U-statistic.
- For **clustering** tasks, use metrics like Silhouette Score, Davies-Bouldin Index, Calinski-Harabasz Index (Variance Ratio Criterion), Dunn Index, Adjusted Rand Index (ARI), Normalized Mutual Information (NMI).
- For **recommendation systems**, use metrics such as Precision at K (P@K), Recall at K (R@K), Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Normalized Discounted Cumulative Gain (NDCG), Hit Ratio.
- For **Natural Language Processing (NLP)** tasks, use metrics like BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), Perplexity, F1-score for Named Entity Recognition (NER), Word Error Rate (WER), TF-IDF (Term Frequency-Inverse Document Frequency), Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, GloVe (Global Vectors for Word Representation), BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pretrained Transformer), Transformer Models (e.g., T5, RoBERTa), LSTM for Text Classification/Sequence Prediction, Attention Mechanisms, Named Entity Recognition (NER), Topic Modeling (e.g., LDA - Latent Dirichlet Allocation).
- For **image generation and prediction models**, use metrics such as Inception Score (IS), Fréchet Inception Distance (FID), Peak Signal-to-Noise Ratio (PSNR),

Structural Similarity Index (SSIM), Dice Similarity Coefficient (DSC), Mean Intersection over Union (IoU), Generative Adversarial Networks (GANs), DCGAN (Deep Convolutional GAN), CycleGAN, StyleGAN, BigGAN, Variational Autoencoders (VAEs), Convolutional Neural Networks (CNNs) for Image Classification and Prediction, ResNet (Residual Networks), Inception Networks, YOLO (You Only Look Once) for Object Detection, U-Net for Image Segmentation, Mask R-CNN for Instance Segmentation, Fast R-CNN for Object Detection.

- For **neural networks**, use metrics such as Cross-Entropy Loss, Accuracy for Classification, Mean Squared Error (MSE) for Regression, Area Under Precision-Recall Curve (AUC-PR), Gradient Norm (for training stability), Learning Rate Schedulers, Activation Functions Visualization.
- For **image classification**, use metrics like Top-1 Accuracy, Top-5 Accuracy.
- For **semantic segmentation (image-based tasks)**, use metrics such as Mean IoU (Intersection over Union), Pixel Accuracy.
- For **anomaly detection**, use metrics like Precision/Recall for Anomalies, F1-score for Anomaly Detection.

Notes

- Upload your work in this format on the website: DS_Project_P3_[Std numbers].zip. If the project is done in a group, include all of the group members' student numbers in the name.
- On the final presentation day, the complexity and sophistication of your overall project (including dataset choice, pipeline complexity, advanced feature engineering, preprocessing methods, use of advanced tools, and modeling approaches) will be evaluated by the jury panel. Based on this evaluation, your team can earn up to an additional 10% bonus on your total final project grade.
- If the project is done in a group, only one member must upload the work.
- This phase will not be accepted after its deadline i.e. there will be no late and grace policy!

Good luck!