NOVA ASSIGNMENT

End-to-End Machine Learning Application using a Public Dataset

Svetlana Vojtková

Contents

1 Assignment

This project is designed to assess your skills in machine learning, software development, and basics of DevOps/MLOps. We expect you to build an end-to-end machine learning application that includes data preparation, model building, evaluation, deployment, and monitoring. The application should be production-ready, well-documented, and user-friendly.

1.1 Dataset

We recommend using the Lending Club Loan Data from Kaggle. This dataset contains complete loan data for all loans issued through the time period, including the current loan status and latest payment information. You can find the dataset here: `https://www.kaggle.com/wordsforthewise/lending-club`.

1.2 Assignment Guidelines

1. **Data Exploration and Preparation:** Understand the dataset, clean and prepare it for your machine learning model.
2. **Model Building and Validation:** Develop a model to predict loan defaulters. Choose suitable machine learning algorithms, perform feature selection/engineering, and validate your model rigorously.
3. **Application Development:** Develop a web-based application or API that allows users to input necessary information and get predictions on whether a loan will default or not. The application should also provide some basic interpretation of the result.
4. **Deployment and Monitoring:** Deploy your application on a platform of your choice. Include a simple monitoring system to track the performance of your application and model.
5. **Code Versioning and Documentation:** Use Git (or a similar versioning system) to maintain the version of your application, and provide thorough documentation for your code and the application usage.
6. **Testing:** Implement unit tests and/or integration tests to ensure the robustness and reliability of your application.

1.3 Deliverables

1. A link to the deployed application or API, with instructions for use.
2. A link to the GitHub repository (or equivalent) that includes all your code, tests, and documentation.
3. A brief report explaining your model choice, architecture of your application, deployment strategy, and monitoring system.

2 Data Exploration and Preparation

From the given data, I have selected the following columns:

- **loan_amnt**: The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
- **term**: The number of payments on the loan. Values are in months and can be either 36 or 60.
- **int_rate**: Interest Rate on the loan
- **installment**: The monthly payment owed by the borrower if the loan originates.
- **grade**: LC assigned loan grade.
- **subgrade**: LC assigned loan subgrade.
- **annual_inc**: The self-reported annual income provided by the borrower during registration.
- **verification_status**: Indicates if income was verified by LC, not verified, or if the income source was verified.
- **loan_status**: Current status of the loan.
- **dti**: A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
- **earliest_cr_line**: This column provides the date when the borrower opened their earliest credit line.
- **open_acc**: This column represents the number of open credit lines the borrower currently has.
- **pub_rec**: Number of derogatory public records.
- **revol_bal**: Total credit revolving balance.
- **revol_util**: Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
- **total_acc**: The total number of credit lines currently in the borrower's credit file.

After selecting the columns, the next step was to examine the values within the dataset and perform data cleaning as necessary. One important column for prediction was 'loan_status', which initially contained string values representing different loan statuses. In order to prepare the data for modeling purposes, certain adjustments were made to the 'loan_status' column.

Specifically, the strings 'Fully Paid' and 'Current' were modified to 'Paid' since these indicated successful loan outcomes. On the other hand, all other remaining values in the 'loan_status' column were labeled as 'Default'.

Next operations removed rows with missing values in the 'term', 'dti', and 'earliest_cr_line' columns, while the missing values in the 'revol_util' column are replaced with the median value of that column. These steps help ensure the data used for analysis or modeling is complete and minimizes the impact of missing values on the results. Then followed mapping operations. Categorical variables in the specified columns were transformed into numerical representations. Mapping operations were applies on columns 'term', 'grade', 'sub_grade','loan_status', 'verification_status'.

The dataset was divided into input features (X) and the corresponding target variable (y) - 'loan_status'.

Created dataset was imbalanced therefore I used RandomUnderSampler. The code balances the class distribution in the target variable by reducing the number of instances in the majority class. Final step in data preparation was splitting the dataset for testing and training data.

3 Model Building and Validation

In the context of predicting loan defaults, the Decision Tree Classifier and Random Forest Classifier were chosen due to the nature of the problem, which involves classification. However, during the initial experimentation phase, it was observed that the decision tree classifier yielded lower accuracy compared to expectations based on default parameters.

As a result, the focus was shifted to the random forest classifier. I aimed to enhance the accuracy of default prediction by using the power of multiple decision trees working in parallel.

Random Forest is good for imbalanced dataset. Initially, I ran the model without any undersampling techniques and found that it achieved a high accuracy score. However, when I examined the confusion matrix, it became evident that the model was primarily focusing on the majority class, which indicated overfitting.

After implementing undersampling, I observed that the accuracy score of Random Forest model decreased. However, upon analyzing the confusion matrix, I noticed a improvement in the model's performance on the minority class. The predictions became more equitable across both classes.

Best resutls got Random Forest Classifier with parameters shown in 1.

```
model = RandomForestClassifier(n_estimators= 100,max_depth = 110,
    min_samples_leaf= 2)
```

Listing 1: Random Forest Classifier parameters

Accuracy on test data was 68.11% and in table 1 is the confusion matrix, which provides a comprehensive overview of the model's predictions and their alignment with the true class labels.

True Class	**Predicted Class**	
	Positive	**Negative**
Positive	40503	20197
Negative	18517	42183

Table 1: Confusion Matrix

In order to cope with limited resources, I deployed the model with a smaller size, which resulted in a reduced accuracy of 65.14%. The confusion matrix for this model is as follows:

True Class	**Predicted Class**	
	Positive	**Negative**
Positive	36845	23855
Negative	18459	42241

Table 2: Confusion Matrix

4 Application Development

The application is a React app that utilizes API calls to interact with a backend server where a machine learning model is implemented. The React app is designed to provide a user-friendly interface, featuring a form that users can fill out to obtain a result based on the model's predictions.

Application is deployed on link: `https://reactloan-production.up.railway.app/`

API is deployed on link: `https://flask-production-a60d.up.railway.app/`

5 Deployment and Monitoring

For the deployment of the application, I chose the Railway deployment platform. To set up the deployment, I utilized the template projects offered by Railway for both the frontend and backend components. One of the features of Railway is its seamless integration with GitHub. By connecting Railway to my GitHub repository, any changes made to the codebase on GitHub trigger an automatic deployment to the Railway platform.

I created a workflow on GitHub for floask_loan and react_loan repository that involved building the project on GitHub and deploying it to Railway. Example of workflow is shown in the figure 1

```yaml
on:
    push:
        branches:
          - main
jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.9

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Install Railway
        run: curl -fsSL https://railway.app/install.sh | sh

      - name: Deploy
        run: railway up
        env:
          RAILWAY_TOKEN: ${{ secrets.RAILWAY_TOKEN }}
```

Figure 1: Yaml file for worflow on GitHub

Each application deployed on Railway comes with built-in monitoring metrics for CPU, Memory and Network.

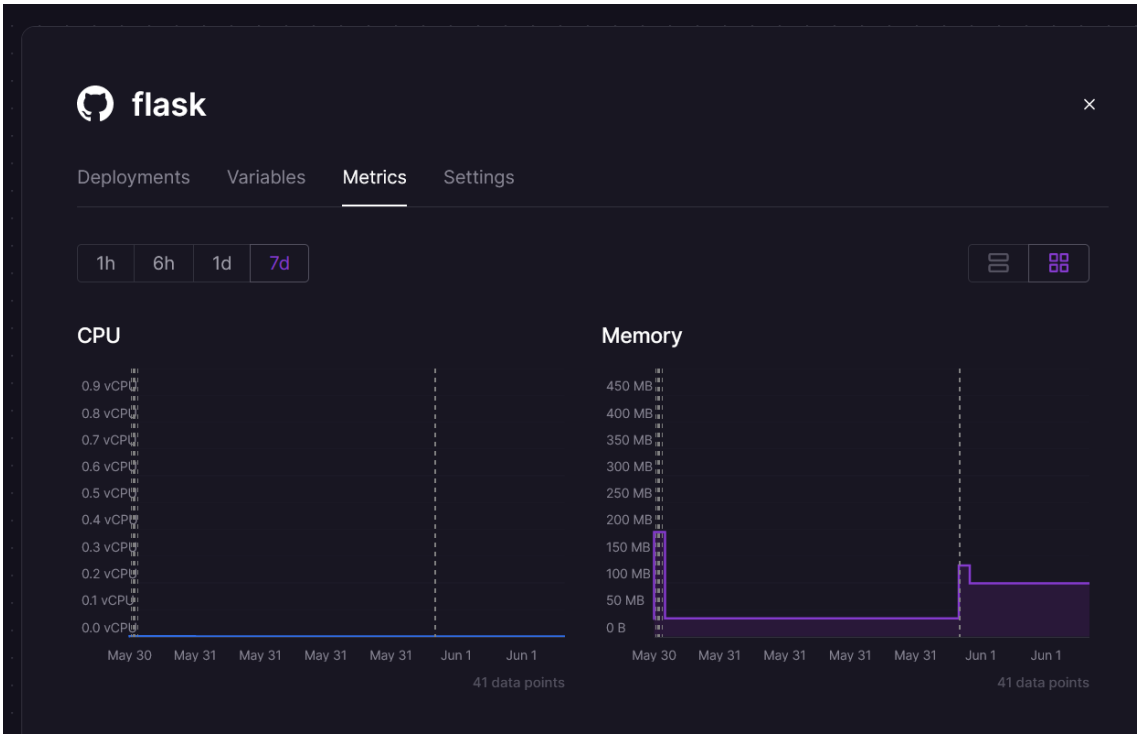

Figure 2: Monitoring for API

6 Code Versioning

Links for GitHub:

- `https://github.com/svojtkova/flask_loan`
- `https://github.com/svojtkova/react_loan`

6.1 API

The backend of the application is implemented using the Python Flask API framework. To make API calls, a JSON payload must be provided. This JSON payload is carefully formatted to ensure compatibility with the underlying model. Once the JSON payload is received, the model processes it and calculates the probabilities associated with the input. This ensures that the application operates reliably and consistently.

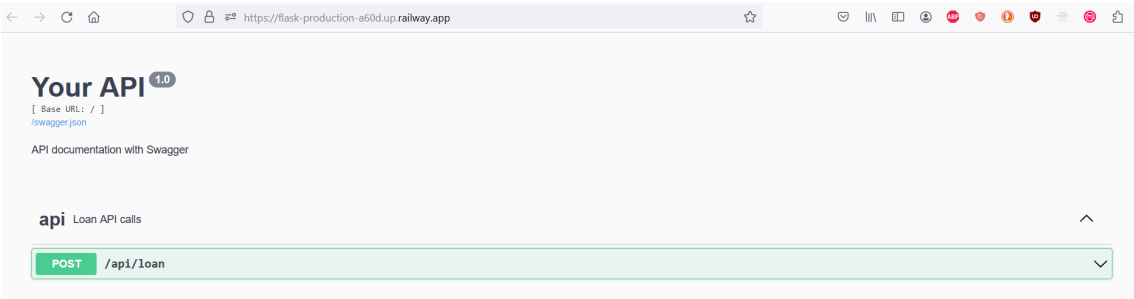

Figure 3: API Python

6.2 Frontend

The frontend of the application is developed using React. The application consists of a single page that contains a form requiring user input. Each input field is equipped with a toolkit feature, which provides additional information about the purpose of that particular input.

Upon submitting the form, an API call is initiated to send the user's input data to the backend. The backend processes the request and generates a response. The response is then displayed in a dialog window, allowing the user to view the result.

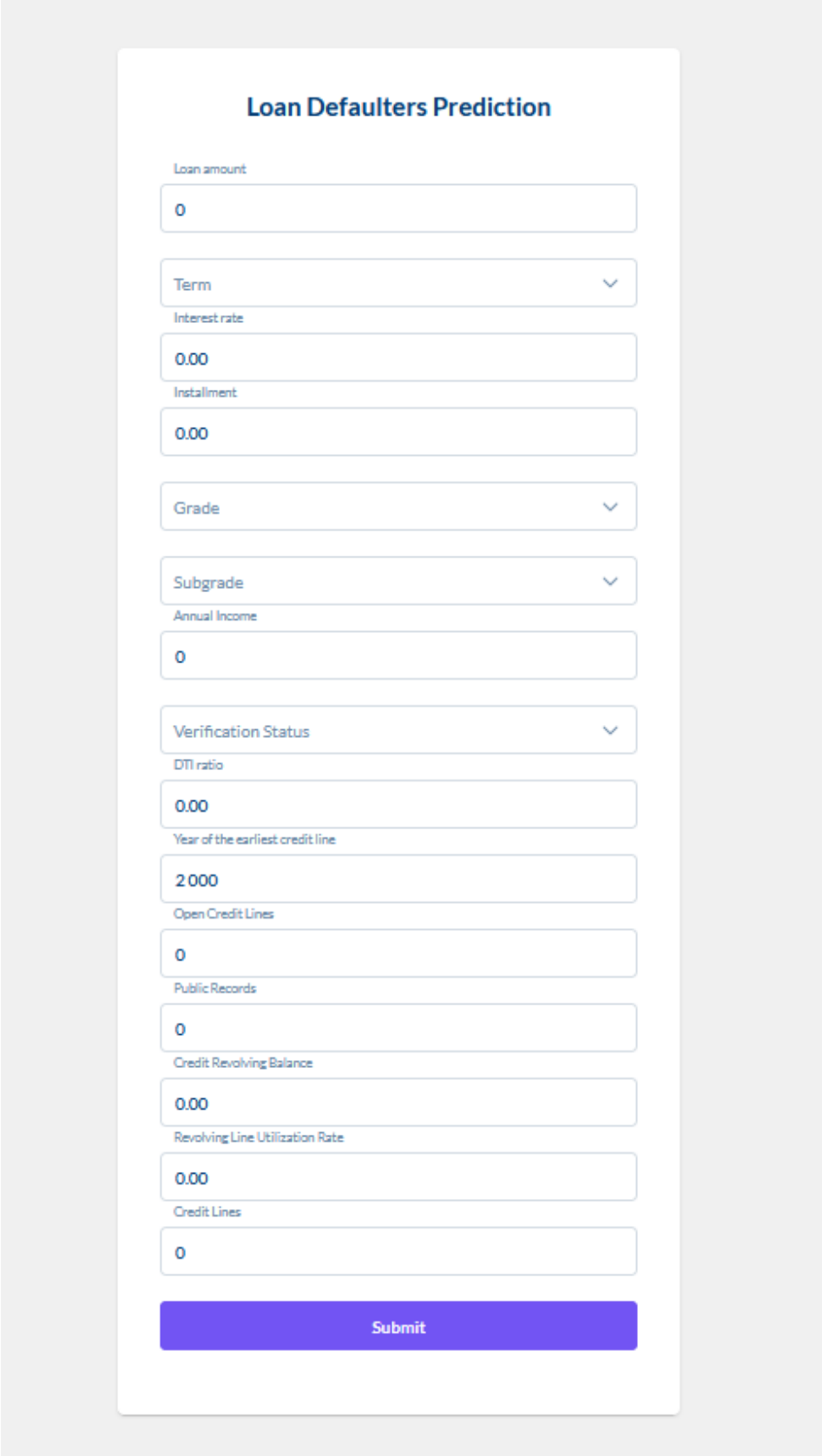

Figure 4: Frontend React