Configuration Manual

MSc Research Project
Data Analytics

Krisztina Hapek
Student ID: X17126631

School of Computing
National College of Ireland

Supervisor: Dr. Majid Latifi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Krisztina Hapek
Student ID:	X17126631
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Majid Latifi
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	670
Page Count:	5

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	21st September 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Krisztina Hapek
X17126631

1 Introduction

The goal of this project was to propose a fairness-aware recommender system for peer-to-peer charitable lending platform, Kiva. All pre-processing, model development and evaluation was performed using Python language in Jupyter Notebook.

This configuration manual presents the hardware and system configurations and data source for replication of the project.

2 Hardware

The hardware used for the implementation of this project was a MacBook Pro with macOS Big Sur version 11.5.1 operating system, 2.3 GHz processor and 8GB RAM as shown in figure 1.

Figure 1: Hardware configuration

3 Environment

The project was fully developed in Jupyter Notebook 6.1.4. available through Anaconda Navigator as shown in figure 2.

Anaconda can be downloaded at `https://www.anaconda.com/products/individual`.

RStudio and the Spyder IDE were considered in the initial phase of the project, however, due to the large size of the dataset and the limitations of the computational power, these environments could not handle the dataset efficiently.

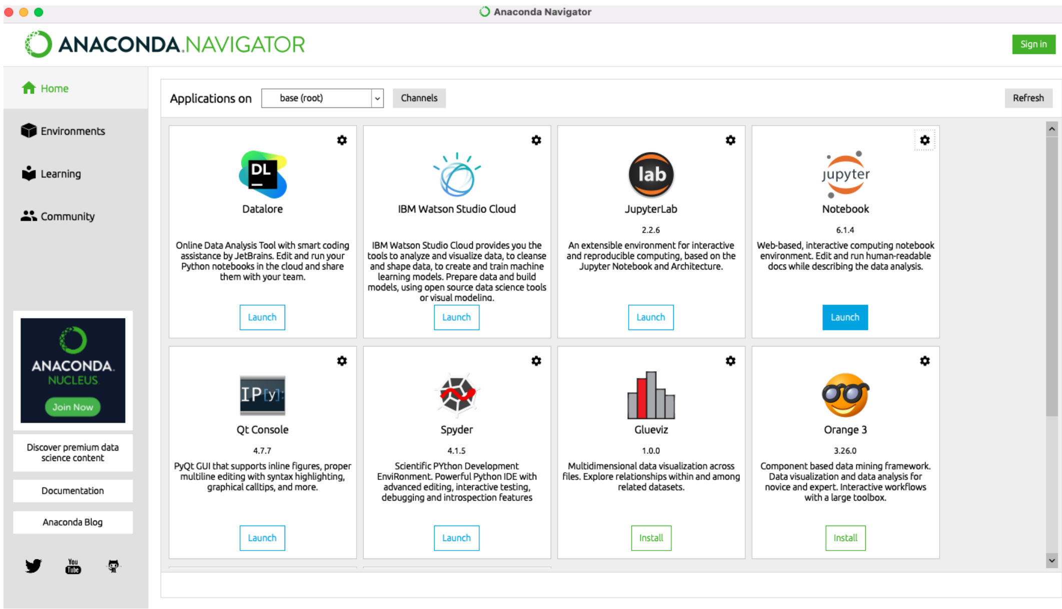

Figure 2: Anaconda Navigator Home

4 Data

The datasets used in this project were downloaded from the Kiva platform's Developer Home on `https://www.kiva.org/build/data-snapshots`.

The platform provides data snapshots in JSON and CSV formats. The latter was used in the development of this project. The snapshots consist of three datasets describing loan characteristics, loan - lender interactions and lenders. As explained in the technical report, the loan - lender interaction and lender datasets were not included in the model development after data exploration revealed that the recommender system could not be built on historical transactions due to the low proportion of returning lenders.

The main raw dataset that the models were developed on, consisted of 34 features shown in figure 3. The final models used 12 independent variables one-hot encoded and normalised. These were presented in the technical report.

The data cleaning and transformation process consisted of various steps including dropping and transforming missing values, dropping features due to large proportion of missing values, changing data types, adding new, calculated features, one-hot encoding of categorical variables and normalising float variables.

5 Python Libraries

Figure 4 shows the libraries used during data preparation, visualisations, model implementation and evaluation and random number generation for the recommender system. The project relied on Scikit-learn's classification implementation and evaluation packages. The recommender system implementation did not require a specific package.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1951124 entries, 0 to 1951123
Data columns (total 34 columns):
 #   Column                          Dtype
---  ------                          -----
 0   LOAN_ID                         int64
 1   LOAN_NAME                       object
 2   ORIGINAL_LANGUAGE               object
 3   DESCRIPTION                     object
 4   DESCRIPTION_TRANSLATED          object
 5   FUNDED_AMOUNT                   float64
 6   LOAN_AMOUNT                     float64
 7   STATUS                          object
 8   IMAGE_ID                        float64
 9   VIDEO_ID                        float64
 10  ACTIVITY_NAME                   object
 11  SECTOR_NAME                     object
 12  LOAN_USE                        object
 13  COUNTRY_CODE                    object
 14  COUNTRY_NAME                    object
 15  TOWN_NAME                       object
 16  CURRENCY_POLICY                 object
 17  CURRENCY_EXCHANGE_COVERAGE_RATE float64
 18  CURRENCY                        object
 19  PARTNER_ID                      float64
 20  POSTED_TIME                     object
 21  PLANNED_EXPIRATION_TIME         object
 22  DISBURSE_TIME                   object
 23  RAISED_TIME                     object
 24  LENDER_TERM                     float64
 25  NUM_LENDERS_TOTAL               int64
 26  NUM_JOURNAL_ENTRIES             int64
 27  NUM_BULK_ENTRIES                int64
 28  TAGS                            object
 29  BORROWER_NAMES                  object
 30  BORROWER_GENDERS                object
 31  BORROWER_PICTURED               object
 32  REPAYMENT_INTERVAL              object
 33  DISTRIBUTION_MODEL              object
dtypes: float64(7), int64(4), object(23)
memory usage: 506.1+ MB
```

Figure 3: Dataframe features

```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
import random
```

Figure 4: Python libraries

6 Models

This project combined classification models and a custom implementation of the ϵ-greedy policy. The models were run with various parameters, different train-test splits and with different target variables to predict either a binary or a 5-class classification. The algorithms presented here show the final implementation of each model. Figure 5 shows the Multinomial Naive Bayes implementation. Figure 6 shows the logistic regression algorithm. Figure 7 shows the ϵ-greedy implementation, which was an adapted version of an implementation proposed by LeDoux (2020).

The original ϵ-greedy policy has a temporal element as the model learns from historical rewards through each iteration. This temporal element was not included in the present project as loan applications have a finite life on the platform and they expire either after getting funded or after the allowed funding period is over, therefore the same loans cannot be recommended infinitely. Furthermore, the goal of the proposed model was to avoid strengthening biases, thus in stead of learning from prior rewards achieved, the loan selection was based on a dummy reward derived from the predicted funding status of the applications. Applications predicted as not funded received a higher dummy reward score than applications predicted as funded.

```
MultiNB = MultinomialNB()
mnb = MultiNB.fit(X_train, Y_train)
print(MultiNB)

Y_expect = Y_test
Y_pred_mnb = MultiNB.predict(X_test)

print(accuracy_score(Y_expect, Y_pred_mnb))
print(precision_recall_fscore_support(Y_expect, Y_pred_mnb, average='binary'))

#print(confusion_matrix(Y_test, Y_pred_mnb))
plot_confusion_matrix(mnb, X_test, Y_test)
plt.show()

print(classification_report(Y_test, Y_pred_mnb))
```

```
MultinomialNB()
0.934402582373639
(0.9596509808911476, 0.9720026306761861, 0.9657873154600657, None)
```

Figure 5: Multinomial Naive Bayes

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver="liblinear", random_state=0).fit(X_train, Y_train)

Y_pred_lr = lr.predict(X_test)

print(accuracy_score(Y_expect, Y_pred_lr))
print(precision_recall_fscore_support(Y_expect, Y_pred_lr, average='binary'))

#print(confusion_matrix(Y_test, Y_pred_lr))
plot_confusion_matrix(lr, X_test, Y_test)
plt.show()

print(classification_report(Y_test, Y_pred_lr))
```

```
0.952522887333479
(0.9528262177332089, 0.9996493695285062, 0.9756763506201616, None)
```

Figure 6: Logistic Regression

```python
def epsilon_greedy_policy(rec_input, arms, epsilon=0.3, slate_size=15, batch_size=5):
    '''
    Applies Epsilon Greedy policy to generate loan recommendations.
    Args:
        df: dataframe. Dataset to apply the policy to
        arms: list or array. ID of every eligible arm.
        epsilon: float. represents the % of timesteps where we explore random arms
        slate_size: int. the number of recommendations to make at each step.
        batch_size: int. the number of users to serve these recommendations to before updating the bandit's poli
    '''
    # draw a 0 or 1 from a binomial distribution, with epsilon% likelihood of drawing a 1
    explore = np.random.binomial(1, epsilon)
    # if explore: shuffle loans to choose a random set of recommendations
    if explore == 1 or rec_input.shape[0]==0:
        recs = np.random.choice(arms, size=(slate_size), replace=False)
    # if exploit: sort loans by "score", recommend loans with the highest score
    else:
        scores = rec_input[['dummy_score', 'loan_id']]
        scores['loan_id'] = scores.index
        scores = scores.sort_values('dummy_score', ascending=False)
        recs = scores.loc[scores.index[0:slate_size], 'loan_id'].values
    return recs

# apply epsilon greedy policy to the rec_input dataset
recs = epsilon_greedy_policy(rec_input, arms, epsilon=0.3, slate_size=15, batch_size=20)

# save recs to df
recs_topd = pd.Series(recs, name = 'loan_id')
recommendations_30 = df.merge(recs_topd,left_on='loan_id', right_on='loan_id')

#concat with cumulative
cumulative_30 = pd.concat([cumulative_30,recommendations_30], axis=0)
print("Cumulative dummy score is ", cumulative_30.dummy_score.sum())
print("Mean cumulative dummy score is", cumulative_30.dummy_score.sum() / 15)
print("Number of batches performed:", cumulative_30.dummy_score.count() / 15 )
```

Figure 7: ϵ-greedy policy

References

LeDoux, J. (2020). Multi-armed bandits in python: Epsilon greedy, ucb1, bayesian ucb, and exp3.
URL: *https://jamesrledoux.com/algorithms/bandit-algorithms-epsilon-ucb-exp-python/*