Configuration Manual

MSc Research Project
Data Analytics

Aoife Gaffney
Student ID: x19217781

School of Computing
National College of Ireland

Supervisor: Dr. Majid Latifi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aoife Gaffney
Student ID:	x19217781
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Majid Latifi
Submission Due Date:	23/09/2021
Project Title:	Configuration Manual
Word Count:	945
Page Count:	11

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:	23rd 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

Aoife Gaffney
x19217781

1 Introduction

This configuration manual describes the software, environments and settings used in the research project 'An Ensemble Learning Algorithm for ICU Patient Mortality Prediction'. This document can be used to replicate the technical work carried out in the research project.

2 Hardware Used

This research project was conducted on a Macbook Air with the following configuration:

- 1.6GHz dual-core Intel Core i5, Turbo Boost up to 3.6GHz, with 4MB L3 cache
- 8GB of 2133MHz LPDDR3 onboard memory
- Operating System: macOS

3 Environment

Python was used to create the models and Google Colab was used to write and run Python code in an online browser that runs on a hosted online Jupyter Notebook platform. It is cloud based and there is no requirement to install Python packages locally. The code is written in Google Colab and saved to Google Drive. A google account is required for using Google Colab.

4 Implementation

The following section outlines the technical implementation of the project.

4.1 Google Colab environment setup

Several packages were required in this project including: pandas, numpy, seaborn, sklearn, scipy, plotly, mathplotlib and vecstack. These were loaded in as seen in Figure 1 using the pip command if necessary.

```
import pandas as pd
import io
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from numpy import mean
from numpy import std
from pandas.plotting import scatter_matrix
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
import lightgbm
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
from sklearn.metrics import *
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.nae_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
import plotly.graph_objects as go
import vecstack
from vecstack import stacking
from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.feature_selection import mutual_info_classif
```

Figure 1: Packages

4.2 Dataset

The data used in this project were the files 'training_v2.csv' and 'WiDS Datathon 2020 Dictionary.csv' downloaded from Kaggle[1] in csv format. The files were stored on the user's computer and uploaded into Google Colab as seen in Figure 2. Once uploaded, the csv files were read into a pandas dataframe for analysis.

[1]https://www.kaggle.com/c/widsdatathon2020/data

```
]   # choose files to upload
    from google.colab import files
    uploaded = files.upload()
```

```
Choose files  No file chosen        Upload widget is only available when the cell has been executed in the current browser
session. Please rerun this cell to enable.
Saving train_data.csv to train_data.csv
Saving WiDS Datathon 2020 Dictionary.csv to WiDS Datathon 2020 Dictionary.csv
```

```
]   # convert to pandas df
    train_full = pd.read_csv(io.StringIO(uploaded['train_data.csv'].decode('utf-8')))
    dictionary = pd.read_csv(io.StringIO(uploaded['WiDS Datathon 2020 Dictionary.csv'].decode('utf-8
```

Figure 2: File upload

4.3 Data Pre-processing

After uploading, Exploratory Data Analysis was carried out using the pandas package to get overall view of the data and plots were created of categorical variables, Figure 3.

```
[ ]  # all plots
     cat_plot(col_name ='ethnicity', fig_size=(20,10))
     cat_plot(col_name ='age', fig_size=(40,15))
     cat_plot(col_name ='hospital_admit_source', fig_size=(30,12))
     cat_plot(col_name ='icu_admit_source', fig_size=(30,12))
     cat_plot(col_name ='icu_stay_type', fig_size=(30,10))
     cat_plot(col_name ='icu_type', fig_size=(20,10))
     cat_plot(col_name ='apache_3j_bod[illegible]stem', fig_size=(25,12))
     cat_plot(col_name ='apache_2_bodys[illegible]tem', fig_size=(25,12))
```

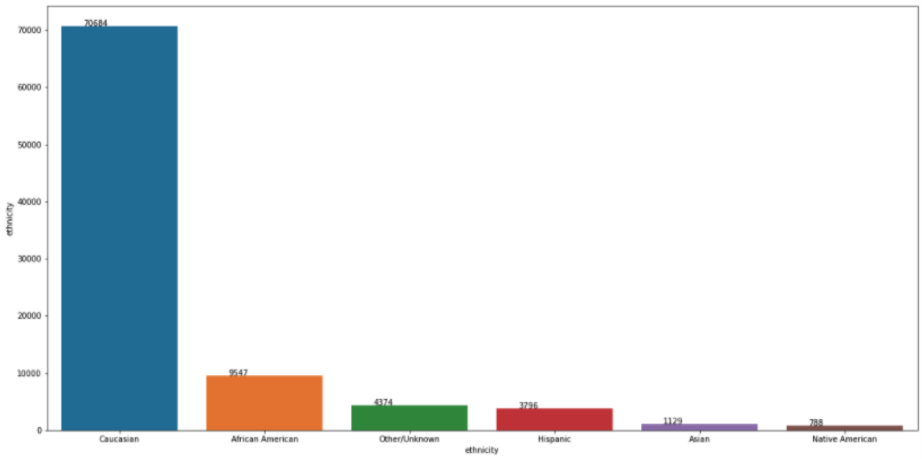

Figure 3: EDA

Feature Engineering was computed on a selection of variables to remove irrelevant features or tidy up groupings within features. BMI was computed with BMI formula due to a high volume of missing values, Figure 4.

```
# tidy up apahce 2 bodysystem
train_full['apache_2_bodysystem'] = train_full['apache_2_bodysystem'].replace({'Undefined diagno
```

```
# compute bmi using bmi formula to fill in missing bmi values
train_full['new_bmi'] = (train_full['weight']*10000)/(train_full['height']*train_full['height'])
train_full['bmi'] = train_full['bmi'].fillna(train_full['new_bmi'])
train_full = train_full.drop(['new_bmi'], axis = 1)
```

```
# drop unneccessary features such as id and stay types that are duplicate features
train_full.drop(['icu_admit_source', 'icu_id', 'icu_stay_type', 'patient_id', 'hospital_id'],axi
```

Figure 4: Feature Engineering

Correlation was computed with the corr() function on non categorical features and all those features with correlation greater than 0.9 were removed, Figure 5


```
# tidy up apahce 2 bodysystem
train_full['apache_2_bodysystem'] = train_full['apache_2_bodysystem'].replace({'Undefined diagno
```

```
# compute bmi using bmi formula to fill in missing bmi values
train_full['new_bmi'] = (train_full['weight']*10000)/(train_full['height']*train_full['height'])
train_full['bmi'] = train_full['bmi'].fillna(train_full['new_bmi'])
train_full = train_full.drop(['new_bmi'], axis = 1)
```

```
# drop unneccessary features such as id and stay types that are duplicate features
train_full.drop(['icu_admit_source', 'icu_id', 'icu_stay_type', 'patient_id', 'hospital_id'],axi
```


Figure 5: Correlation

A search for missing values was computed and all features with greater than 60% missing values were removed. MICE imputation was performance using sklearn package and SimpleImputer() on the rest of missing values using the 'mean' strategy for non categorical features and 'most frequent' strategy for categorical features, Figure 6

```
# apply MICE for numerial values with most frequent strategy
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
# apply to data
train_cat = pd.DataFrame(imputer.fit_transform(train_cat))
train_cat.columns =cat_column_names;
```

Figure 6: MICE

One Hot encoding was preformed on categorical features to get dummy variables with pandas package using get.dummies() function, Figure 7

```
# apply one hot encoding to categorical features to get dummy varibles
train_encoded=pd.get_dummies(train, columns=['ethnicity', 'gender',
       'icu_type', 'apache_3j_bodysystem',
       'apache_2_bodysystem','hospital_admit_source'])
#list dummy variables
train_encoded.columns
```

Figure 7: One Hot Encoding

Standardisation was applied to all numerical data to scale using StandardScaler() from sklearn, Figure 8.

```
# apply scaling to numerical data
features_info = pd.DataFrame()
features_info['unique values'] = train_encoded.nunique()
# scale numerical data only and remove dummy/cat features
columns_to_scale = features_info[features_info['unique values'] > 2].index.values
scaler = preprocessing.StandardScaler()
scaled_columns = scaler.fit_transform(train_encoded[columns_to_scale])
scaled_features_df = pd.DataFrame(scaled_columns, index=train_encoded.index, columns=columns_to_
# replace with scaled features
train_scaled = train_encoded.drop(columns_to_scale, axis=1)
train = train_scaled.join(scaled_features_df)
```

Figure 8: Scaling

Oversampling was applied to minority set (death) by using SMOTE resample() function in sklearn, Figure 9.

```
# apply SMOTE to oversample minority class
train_minority_upsampled = resample(train_minority,
                                    replace=True,     # sample with replacement
                                    n_samples=83798,    # to match majority class
                                    random_state= 303) # reproducible results

# combine upsampled minority with majority class
train_upsampled = pd.concat([train_majority, train_minority_upsampled])

# new class counts
train_upsampled.hospital_death.value_counts()
```

Figure 9: SMOTE

4.4 Feature Selection

A filter feature selection method GenericUnivariateSelect() from sklearn was used to select the top 20 features for prediction of ICU mortality. This was applied to the dataset and a new separate dataset with only the top 20 features was created, Figure 10.

```
# apply festure selection to dataset using k best = 20
trans = GenericUnivariateSelect(score_func=mutual_info_classif, mode='k_best', param=20)
# apply to dataset
feature_selection_X_trans = trans.fit_transform(feature_selection_X, feature_selection_y)
print("We started with {0} features but retained only {1}".format( feature_selection .shape[1] -
```

```
We started with 120 features but retained only 20
```

Figure 10: Feature Selection

The datasets are spilt in 30% test and 70% train using train_test_split function from sklearn, Figure 11.

Spilt

```
[ ]  # Split original dataset into training and test set
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)
```

```
[ ]  # Split top 20 features dataset into training and test set
     X_train_new, X_test_new = train_test_split(X_new, test_size = 0.3, random_state = 1)
```

Figure 11: Data Spilt

4.5 Models

The RF base model was computed using Sklearn package. Randomised search 10-fold cross validation (CV) was then computed to get their optimised parameters with RandomisedSearchCV(). The model with optimised parameters was applied to full dataset without feature selection. Then, a second RF model was then computed with randomised search 10-fold CV for optimised parameters on the dataset with feature selection. Sample code in Figure 12.

```
# hyperparamter search
n_estimators = [100, 300, 500]
max_depth = [5, 8, 15]
min_samples_split = [2, 5, 10, 15]
min_samples_leaf = [1, 2, 5]
# create hyperpraramter dict
rf_param = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
             min_samples_leaf = min_samples_leaf)
# apply randomsearch
rf1_search = RandomizedSearchCV(rf1, rf_param, cv = 10, verbose = 1,
                      n_jobs = -1)
#fit
rf1_best = rf1_search.fit(X_train, y_train)
```

```
# examine the best model
print(rf1_best.best_score_)
print(rf1_best.best_params_)
print(rf1_best.best_estimator_)
rf_best = rf1_best.best_estimator_
```

Figure 12: Sample Random Forest code

Each single classifier, DT, SVM, LR and DT were computed using Sklearn package functions. Randomised search 10-fold CV was then computed to get their optimised parameters with RandomisedSearchCV(). NB is excluded as it does not have hyperparmeters to optimise. For each single classifier, the model with optimised parameters was applied to full dataset without feature selection. Then, for each single classifier a second model was then computed with randomised search 10-fold CV (with the exception of NB) was applied on the dataset with feature selection. Sample code for DT and SVM in Figure 13 and Figure 14.

```
# examine the best model
print(svm1_best.best_score_)
print(svm1_best.best_params_)
print(svm1_best.best_estimator_)
svm1_best = svm1_best.best_estimator_
```

0.7941645786966224

```
# parameters
svm_param = {'C': [0.1,1, 10]}
# randomsied search with 10 cross validation
svm2_search = RandomizedSearchCV(svm2,svm_param, verbose=2, cv=10, n_jobs=-1)
# fit
svm2_best = svm2_search.fit(X_train_new,y_train)
```

Figure 13: Sample SVM Code

```
# parameter grid based
dt_param = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
#randomsied search with 10 fold cv
dt_search = RandomizedSearchCV(dt1, dt_param, cv=10, n_jobs=-1, verbose=1)
# fit
dt_best = dt_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  4.0min finished
```

```
# examine the best model
print(dt_best.best_score_)
print(dt_best.best_params_)
print(dt_best.best_estimator_)
dt1_best = dt_best.best_estimator_
```

Figure 14: Sample DT Code

Stacking was computed of the 4 base single classifiers using the vecstack package and stacking() function. Each model with optimised parameters was applied first to the full dataset without feature selection, Figure 15. The best model (DT) was then reapplied to the stacking function to get the final prediction, Figure 16. This same process is repeated for the dataset with feature selection.

```
# models with optimised parameters
models = ([dt1_best,
           nb1_best,
           svm1_best,
           lr1_best])
```

```
# Stacking model
S_train, S_test = stacking(models,
                           X_train, y_train, X_test,
                           regression=False,

                           mode='oof_pred_bag',

                           needs_proba=False,

                           save_dir=None,

                           metric=roc_auc_score,

                           n_folds=10,

                           stratified=True,

                           shuffle=True,

                           random_state=0,

                           verbose=2)
```

Figure 15: Stacking

```
]  #Best base model
   model2 = dt2_best
   # fit
   model2 = model2.fit(S_train_new, y_train)

   stack2_pred = model2.predict(S_test_new)
   print('Final prediction score: [%.8f]' % accuracy_score(y_test, stack2_pred))

Final prediction score: [0.80954275]
```

Figure 16: Stacking

The LGBM base model was computed using Sklearn package. Randomised search 10-fold CV was then computed to get the optimised parameters using RandomisedSearchCV(). The model with optimised parameters was applied to full dataset without feature selection, Figure 17. As second model was then computed with randomised search 10-fold CV on the dataset with feature selection. Sample code in Figure 18

```
#set parameters
lgbm_params ={'cat_smooth' : sp_randint(1, 50),
          'learning_rate': [0.08, 0.85, 0.09],
          'num_leaves': sp_randint(500, 5000),
          'max_bin': sp_randint(100, 1500),
          'max_depth': sp_randint(1, 15),
          'min_data_in_leaf': sp_randint(500,3500)}
```

```
# set fit parameters
fit_params={"early_stopping_rounds":2,
            "eval_metric" : 'auc',
            "eval_set" : [(X_train, y_train),(X_test,y_test)],
            'eval_names': ['train','valid'],
            'verbose': 300,
            'categorical_feature': 'auto'}
```

```
#Random search
lgbm_search = RandomizedSearchCV(lgbm1, lgbm_params, scoring='roc_auc',cv=10, refit=True,random_
```

Figure 17: LGBM Parameters

```
#CV
CV_NUMBER = 10

excl = ["Class", "datetime"]
features = [f for f in X.columns if f not in excl]

auc_list = []
recall_list = []
precision_list = []
accuracy_list = []
F1_list = []

# Cross Validation
for idx in range(CV_NUMBER):

    # Create dataset for lightgbm
    lgb_train = lightgbm.Dataset(X_train, y_train)
    lgb_eval = lightgbm.Dataset(X_test, y_test, reference=lgb_train)

    bst = lightgbm.train(params=final_params, train_set=lgb_train, num_boost_round=500,
                         valid_sets=lgb_eval, early_stopping_rounds=2)
    y_pred = bst.predict(X_test, num_iteration=bst.best_iteration)
    y_pred = np.round_(y_pred, 0)

    auc_list.append(roc_auc_score(y_test, y_pred))
    recall_list.append(recall_score(y_test, y_pred))
    precision_list.append(precision_score(y_test, y_pred))
    accuracy_list.append(accuracy_score(y_test, y_pred))
    F1_list.append(f1_score(y_test, y_pred))
```

Figure 18: LGBM

5 Evaluation

The evaluation metrics applied to each model were Accuracy, AUC, Recall, Precision and F1 Score using sklearn package. These metrics were computed on both test sets with and without feature selection, sample code in Figure 19

```python
# evaluate model
svm2_best_predict = svm2_best.predict(X_test_new)
print("Confusion Matrix")
print(confusion_matrix(y_test, svm2_best_predict ))
print("AUC Score")
print(roc_auc_score(y_test, svm2_best_predict))
print("Accuracy")
print(accuracy_score(y_test, svm2_best_predict ))
print("Recall")
print(recall_score(y_test, svm2_best_predict ))
print("Precision")
print(precision_score(y_test, svm2_best_predict ))
print("F1 Score")
print(f1_score(y_test, svm2_best_predict))
```

```python
# plots
predictedLabels = (svm2_best_predict).astype(int)
plt.figure(figsize=(13,10))
plt.subplot(221)
sns.heatmap(confusion_matrix(y_test,predictedLabels),annot=True,fmt = "d",linecolor="k",linewidt
plt.title("SVM TOP 20 FEATURES CONFUSION MATRIX",fontsize=10)
predicting_probabilites = svm2_best_predict
fpr,tpr,thresholds = roc_curve(y_test,predicting_probabilites)
plt.subplot(222)
plt.plot(fpr,tpr,label = ("Area_under the curve :",auc(fpr,tpr)),color = "r")
plt.plot([1,0],[1,0],linestyle = "dashed",color ="k")
plt.legend(loc = "best")
plt.title("SVM TOP 20 FEATURES AUC",fontsize=10)
plt.show()
```

Figure 19: Evaluation metrics

Plots of confusion matrix and AUC curve were computed for each model on both test sets with and without feature selection, sample code in Figure 19. Example of plot below in Figure 20

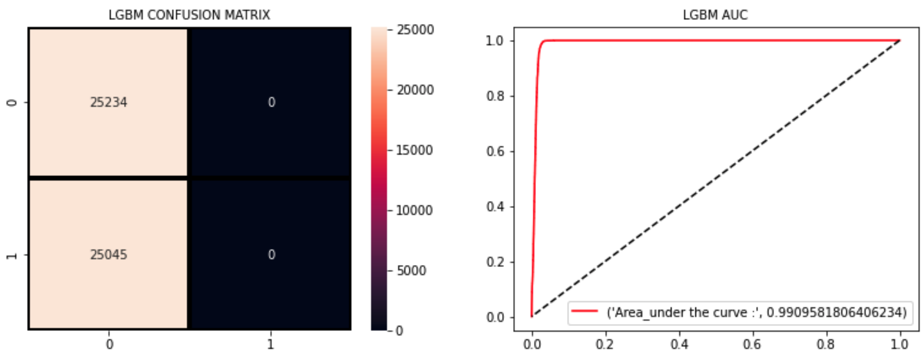

Figure 20: Plots