CC7182 Programming for Analytics Spring Semester 2021-2022

Coursework Assignment

Student name: **Dinh Viet Tuan** Student Number: **21041406**

Assessors' initials

Table of Contents

Introduction

This coursework involves analysing anonymous data dataset involved in real-world Islington iWork unemployment support data from 2018-to 2021. The technical report on data understanding, preparation, exploration, initial analysis, and data mining will be created using Python programs.

The "Islington iwork anonymous data.csv" data set contains 4788 client records. Each record has 14 variables, including socio-demographic and personal information. Employer, Variable 1 is the target variable, and it indicates that a customer has been employed where the employer has an outcome.

The technical report includes six main steps, which are data understanding, data preparation, data analysis, data exploration, data mining and reflection on the work.

1. Data understanding

1.1 Produce a meta data table and describe missing or error data of each variable

In the first step, we will import the necessary libraries for analysis, as well as reading the data file.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder, MultiLabelBinarizer
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB, CategoricalNB, BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.metrics import roc_auc_score

plt.style.use(['seaborn', 'seaborn-whitegrid'])
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings("ignore")
```

```
dfEmployer = pd.read_csv('Islington_iwork_anonymous_data.csv')
dfEmployer.columns = dfEmployer.columns.str.lower()
dfEmployer.head(3)
```

And then, creating a table of meta data to display the characteristics of each variable will help us have an overview of the data. The table contains information about data type, number of number of unique or distinct values, cardinality ratio, null ration and the level of measurements

variable	dType	nUnique	cardinality	nullRatio	measurement
Employer	object	282	0.0589	0.0002	categorical
Registration_Date	object	853	0.1782	0	interval
Client_Current_Age	int64	70	0.0146	0	interval
Parent_on_Enrolment	object	3	0.0006	0	categorical
Gender	object	6	0.0013	0	categorical
Ethnic_Origin	object	33	0.0069	0	categorical
Has_Disability	object	4	0.0008	0	categorical
Disability_details	float64	0	0	1	categorical
Religion	object	12	0.0025	0	categorical
Sexuality	object	7	0.0015	0	categorical
Highest_Level_of_Education	object	10	0.0021	0	nominal
Claiming_Benefits	object	3	0.0006	0	categorical
Benefits	object	74	0.0155	0.5236	categorical
Ward_Name	object	205	0.0428	0.0015	categorical

Table 1: Characteristics of 14 Variables

Table 1 is summarized characteristics of variables according to the following information as follows.

With data types (pandas'), can be either string (object), integer (int64) or decimal (float64).

```
#data type
dfEmployer.dtypes
```

In term of "nUnique", the number of unique or distinct values and cardinality ratio equals the number of unique values divided by the total number of observations. Neither too high

cardinality (for example, employer's id) nor too low cardinality (for example, all values are the same) does not contribute much information to the analysis.

```
# cardinality ratio
(dfEmployer.apply(pd.Series.nunique) / dfEmployer.shape[0]).round(4)
```

```
# observe distinct values of some variables
dfEmployer.parent_on_enrolment.unique()
```

And so on(please see detail in the python note book)

Null ratio, we first consider only nan values detected by Pandas. Then also look closely at possible values of some variables: Has_Disability, Religion, Highest_Level_of_Education, Claiming_Benefits, Benefits, Ward_Name. Many values can be considered missing, such as "Blanks", "Prefer not to say", "Not Known", and "No response", but we have not touched them yet and will treat them differently. The nullratio of the variable disability_details is 100%, indicating that the dataset does not include specific information about the disability category for any observation.

```
#null ratio
dfEmployer.isna().sum() / dfEmployer.shape[0]
```

The level of measurements can be either ratio, interval, nominal or categorical. There is only one nominal feature in this dataset, two interval features, the rest are all categorical, and there is no ratio feature.

2. Data preparation

2.1 Feature selection.

In this step, we will reduce variables with no influences on the target.

```
df = dfEmployer.copy()

# feature selection
df = df.drop(columns=['registration_date', 'disability_details'])
df = df.fillna('')
```

The "Registration date" variable is a time series variable. In this analysis report, we will concentrate on building a model to predict whether an employer would have an outcome. Therefore, the time series variable will be omitted.

The "Disability details" variable includes 100 per cent null values and hence has no predictive value. This variable was also eliminated.

Other factors were retained, although they have not strongly correlated with the target variable. We suppose that none of them individually influence the goal, but their combinations(like decision path in decision tree or linear combination in logistic regression) may somehow relate to the target.

Some algorithms, such as Decision Tree or Logistic Regression with L1 regularisation, provide techniques to minimise the influence of unimportant characteristics to zero.

2.2 Clean and transform data

First of all,we will transform the target variable Employer into binary No Outcome - 0, has an outcome -1

```
# Target variable Employer into binary No Outcome - 0, has an outcome -1
df['employer'] = df.employer.map(lambda x: 0 if x == 'No Outcome' else 1)
```

Variables (genders, claiming_benefits, parent_on_enrolment, has_disability) require encoding using a specific mapping criteria, so I use the Series.map() method which takes an user-defined function as the argument to transform these variables.

(Gender into ordinal Female - 0, Male -1, Transgender=2, Prefer not to say=3, any others=4)

(Claiming_Benefits into ordinal No=0, Yes=1, Blank=2)

```
# Gender into ordinal Female - 0, Male -1, Transgender=2, Prefer not to say=3, any others=4
def mapGender(x):
    if x == 'Female': y = 0
    elif x == 'Male': y = 1
    elif x == 'Transgender': y = 2
    elif x == 'Prefer not to say': y = 3
    else: y = 4
    return y
df['gender'] = df.gender.map(mapGender)

# Claiming_Benefits into ordinal No=0, Yes=1, Blank=2
def mapClamingBenefits(x):
    if x == 'No': y = 0
    elif x == 'Yes': y = 1
    else: y = 2
    return y
df['claiming_benefits'] = df.claiming_benefits.map(mapClamingBenefits)

# parent_on_enrolment
def mapParents(x):
    if x == 'No': y = 0
    elif x == 'Yes': y = 1
    else: y = 2
    return y
df['parent_on_enrolment'] = df['parent_on_enrolment'].map(mapParents)
# has_disability
def mapDisability(x):
    if x == 'No': y = 0
    elif x == 'Yes': y = 1
    elif x == 'Prefer not to say': y = 3
    else: y = 4
    return y
df['has_disability'] = df['has_disability'].map(mapDisability)
```

Variables (religion, sexuality) need to encode in order to use Machine Learning algorithms. We use the OrdinalEncoder class from Scikit-learn to implement this.

```
#encoding in order to use Machine Learning algorithms
encoderReligion = OrdinalEncoder()
df['religion'] = encoderReligion.fit_transform(df[['religion']])

encoderSexuality = OrdinalEncoder()
df['sexuality'] = encoderSexuality.fit_transform(df[['sexuality']])
```

Variables (ethnic_origin, ward_name) should be ordinal encoded based on value occurrences in the dataset. Firstly, We get the unique values, retain their order or reverse them (depends on the question), and then transform it into a Python dictionary where keys are the categories themselves and values are their indices. This dictionary is used as input for the Series.map() function to get the desired result.

(Ethnic_Origin into ordinal number based on their occurrence in the data set in descending order and WARD_NAME into ordinal numbers based on their occurrence in the data set in ascending order.)

```
# Ethnic_Origin into ordinal number based on their occurrence in the data set in descending order

mapEthnic = df.ethnic_origin.unique()[::-1].tolist()
mapEthnic = {cat: idx for idx, cat in enumerate(mapEthnic)}
df['ethnic_origin'] = df.ethnic_origin.map(mapEthnic)

# WARD_NAME into ordinal numbers based on their occurrence in the data set in ascending order
mapWard = df.ward_name.unique().tolist()
mapWard = {cat: idx+1 for idx, cat in enumerate(mapWard)}
mapWard[np.nan] = 0
df['ward_name'] = df.ward_name.map(mapWard)
```

For variable (highest_level_of_education), we use a regular expression (regex), which is a technique that searches for substrings that match a specific pattern to extract the number after the “Level” substring.

(Highest_Level_of_Education into ordinal numbers based on UK ISCED Level)

```
# Highest_Level_of_Education into ordinal numbers based on UK ISCED Level
eduLevel = df.highest_level_of_education.str.extract('(Level \d+)')[0].str.extract('(\d+)')
eduLevel = eduLevel.fillna(0.5).astype(float)
df['highest_level_of_education'] = eduLevel
```

For variable (benefits), we notice that this is a list-like column; each employer may have multiple statuses, not like other variables. The most efficient way of encoding them is one-hot encoding. We use Scikit-learn’s MultiLabelBinarizer class to handle this.

```
#benifit
df['benefits'] = df['benefits'].str.split(', ')
encoder = MultiLabelBinarizer()
dfBenefits = pd.DataFrame(encoder.fit_transform(df.benefits), columns=encoder.classes_)
df = df.join(dfBenefits)
```

The output of this step looks like this:

raw	->	a	b	c
(a,b)		1	1	0
(a,c)		1	0	1
(c)		0	0	1

3. Data analysis

3.1 Summary statistics of sum, mean, standard deviation, skewness, and kurtosis of age variable

To calculate various statistics, we use the function “describe” from the scipy.stats submodule. Because this function returns multiple outputs, we use the unpacking technique to extract desired values. Then print all these values with a precision of 4 decimal places.

The “age” variable has a skewness of 0.9040 > 0, so it is right-skewed. The distribution mass is concentrated to the left, leading to a longer right tail. It has a kurtosis of 3.1877 > 0, so the distribution is peak.

```
[27] from scipy import stats

     age = dfEmployer.client_current_age
     sizeAge, (minAge, maxAge), meanAge, varAge, skewnessAge, kurtosisAge = stats.describe(age)
     statistics = ['sizeAge', 'minAge', 'maxAge', 'meanAge', 'varAge', 'skewnessAge', 'kurtosisAge']
     _ = [print(f'{statistic} = {globals()[statistic]:.4f}') for statistic in statistics]

     sizeAge = 4788.0000
     minAge = 0.0000
     maxAge = 137.0000
     meanAge = 37.3262
     varAge = 198.4655
     skewnessAge = 0.9040
     kurtosisAge = 3.1877
```

This is true when looking at this variable’s histogram.

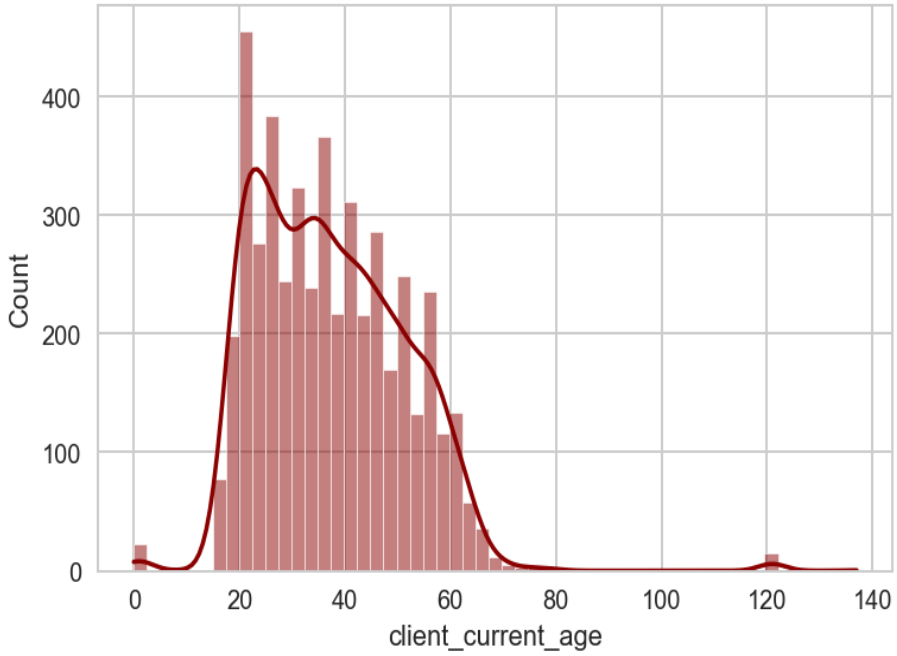


Figure 1: The client_curent_age distribution

3.2 Calculate and show correlation of each variable with the target variable

We use the df.corr() method to calculate correlation coefficients. Then use slicing to extract necessary information only. It is shown that parent_on_enrolment, has_disability, highest_level_of_education, claiming_benefits and not_in_receipt_of_benefits have the highest correlation coefficient with the target variable. However, none of these magnitudes is higher than 0.3. The result illustrates that the correlation is not very high and acceptable.

```
dfEmployer.corr().employer.to_frame().iloc[1:]
```

index	employer
client_current_age	-0.06689800566125428
parent_on_enrolment	-0.29164616044583364
gender	0.022208486379618173
ethnic_origin	-0.009777344895963469
has_disability	-0.28748431275134656
religion	0.024796726074657743
sexuality	-0.029505106077519203
highest_level_of_education	0.2204480356722911
claiming_benefits	-0.2613226785057624
ward_name	0.003691052381952014
carers_allowance	0.009690268956491053
disability_living_allowance_dla	0.06384894887972425
employment_support_allowance_esa	0.035301223214726216
housing_benefit	0.019594639607765998
incapacity_benefit_ib	0.010786357151618723
income_support_is	0.06362722843083272
job_seekers_allowance_jsa	0.12327120270752596
months_on_benefits	0.1669329782330973
not_in_receipt_of_benefits	0.2219396080197633
other_benefit	0.0007384654650156754
personal_independence_payment_pi	-0.01569762830004882
prefer_not_to_disclose_benefits	0.05560339238788256
universal_credit_uc	0.05961196352876675
working_tax_credit	0.02395491816700657

4. Data exploration

4.1 Histogram plot of any user chosen variables

To show histogram plot of any user chosen variables:

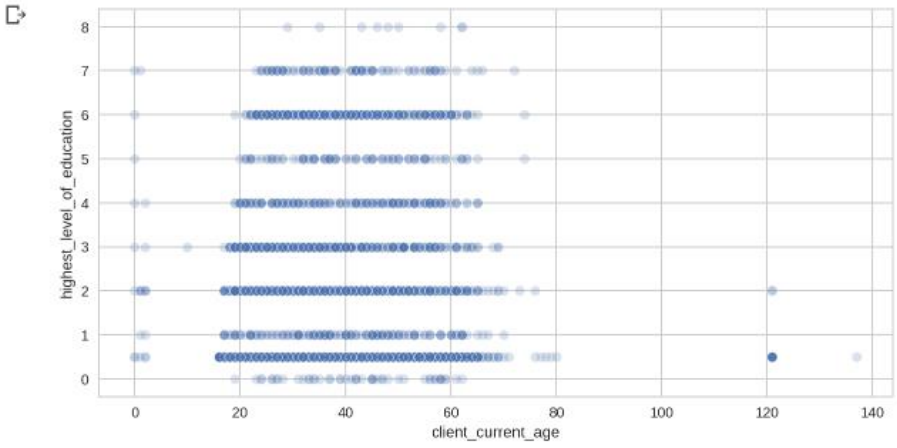
Because histograms work best for univariate data, we plot one for each variable independently. Then we create a for-loop and print out all of the histograms.

```
# input a number of variables
variables = ['client_current_age', 'highest_level_of_education', 'parent_on_enrolment','gender','ethnic_origin','has_disability','religion',
             'sexuality','claiming_benefits','ward_name','carers_allowance','disability_living_allowance_dla',
             'employment_support_allowance_esa', 'housing_benefit','incapacity_benefit_ib','income_support_is','job_seekers_allowance_jsa',
             'months_on_benefits', 'not_in_receipt_of_benefits', 'other_benefit','personal_independence_payment_pi',
             'prefer_not_to_disclose_benefits', 'universal_credit_uc','working_tax_credit']

# plot
for variable in variables:
    plt.figure(figsize=(6,4))
    sns.histplot(df[variable], color='darkred', kde=True, stat='count', edgecolor='w')
    plt.show()
```

4.2 Scatter plot for any two user chosen variables

Scatter plot is designed for continuous variables (continuous-continuous or continuous-categorical). If two categorical variables are used, the plot shows no useful information. So I choose the first variable for the pair to be age. You can replace any other pair to observe their relationship.

```
# input any 2 variables
variablePair = ['client_current_age', 'highest_level_of_education']

# plot
x, y = variablePair
plt.figure(figsize=(10,5))
sns.scatterplot(data=df, x=x, y=y, alpha=0.2)
plt.show()
```



5. Data mining

To determine the optimal method for predicting the goal (whether an employer has an outcome), we consider a variety of algorithms. Only 80 per cent of data is used for training dataset, while the remaining 20 per cent is used for testing dataset. We utilise random splitting to ensure that both sets have the same distribution as the original set. Using a random seed guarantees that the split is deterministic after each call.

```
[ ]  x = df.iloc[:, 1:]
     y = df.iloc[:, 0]
     xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2, random_state=7)
```

```
[ ]  xTrain.shape

     (3830, 24)
```

```
[ ]  xTest.shape

     (958, 24)
```

It is first trained on the train set. Then we use the model to predict given the testing data. A good algorithm must perform well on both sets (not underfitting), as well as the difference between train score and test score must not be very large (not overfitting).

The evaluation metric we use is Area Under the ROC curve (AUROC, or AUC for short). The Receiver Operating Characteristics (ROC) curve is plotted with TPR (True Positive Rate) against FPR (False Positive Rate) at different decision thresholds. It gives an overview of how well the model distinguishes between the two classes. This well-rounded metric takes all four terms tp, fp, tn, fn. The closer AUROC to 1, the better the model is. AUROC should not be less than 0.5

$$\text{TPR} = \text{Recall} = \text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = 1 - \text{Specificity} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

Building model:

```
[ ]  algos = [
         KNeighborsClassifier(n_neighbors=2, weights='uniform'),
         DecisionTreeClassifier(max_depth=4, min_samples_leaf=5),
         GaussianNB(),
         LogisticRegression(),
         BernoulliNB(),
         RandomForestClassifier(max_depth=6),
         ExtraTreesClassifier(n_estimators=300, max_depth=8, max_features=0.2),
         GradientBoostingClassifier(n_estimators=200, learning_rate=0.3, min_samples_leaf=10, max_depth=6, max_features=0.1),
         AdaBoostClassifier(n_estimators=300, learning_rate=0.3)
     ]

     for algo in algos:
         model = algo.fit(xTrain, yTrain)
         yTrainProb = model.predict_proba(xTrain)[:, 1]
         yTestProb = model.predict_proba(xTest)[:, 1]
         aucTrain = roc_auc_score(yTrain, yTrainProb)
         aucTest = roc_auc_score(yTest, yTestProb)
         print(f'aucTrain={aucTrain:.4f} aucTest={aucTest:.4f} [{model.__class__.__name__}]')
```

```
aucTrain=0.9690 aucTest=0.6742 [KNeighborsClassifier]
aucTrain=0.8030 aucTest=0.7647 [DecisionTreeClassifier]
aucTrain=0.7856 aucTest=0.7660 [GaussianNB]
aucTrain=0.7976 aucTest=0.7906 [LogisticRegression]
aucTrain=0.7909 aucTest=0.7664 [BernoulliNB]
aucTrain=0.8850 aucTest=0.8175 [RandomForestClassifier]
aucTrain=0.8914 aucTest=0.8242 [ExtraTreesClassifier]
aucTrain=0.9800 aucTest=0.8279 [GradientBoostingClassifier]
aucTrain=0.8299 aucTest=0.7996 [AdaBoostClassifier]
```

We begin with traditional algorithms such as kNN, Decision Tree, Naive Bayes, and Logistic Regression, since this dataset is quite simple. Logistic Regression, with an appropriate aucTest score of 79.06 percentage, is the best algorithm among these options. Nevertheless, we continue to use ensemble methods.

Theoretically, ensemble methods are ineffective on small data. We use both bagging (Random Forest, Extra Trees) and boosting (Gradient Boosting, Adaptive Boosting), and determine that the worst ensemble model outperforms the best traditional model on this dataset.

Bagging uses voting over several homogeneous weak models in order to reduce variance. Bagging has two components: bootstrapping and aggregating (thus the name). In bootstrapping, the dataset is randomly sampled to generate many weak models. We summarise weak models' results to get the final prediction in aggregating. Bagging works as a committee in which weak models protect others from individual errors. It benefits from the wisdom of the crowd effect. In this dataset, we have tried Random Forest and Extra Trees (Random Forest is very famous, while Extra Trees even goes one step further in randomness between trees). Both impressively performed, with an aucTest of 81.75% and 82.42%, respectively.

Boosting is another ensemble learning technique besides bagging. It trains weak models sequentially, not independently as in bagging. The basic idea of boosting is to tell the machine that "these observations are badly handled, so we can improve it on learning them in the next rounds". We use two types of boosting techniques in this exercise: adaptive boosting and gradient boosting. Adaptive boosting uses a mechanism of automatically re-weighting samples to give misclassification samples higher weights. Gradient boosting borrows the idea from gradient descent, an optimization method which is widely used in Machine Learning and Deep Learning. In gradient boosting, a weak learner is trained using errors made by the previous ones. So that errors become smaller and smaller, after adding up weak models, each model is scaled by a learning rate coefficient. Overall, the best model is Gradient Boosting, with an aucTest of 82.79%.

6. Discussion and reflection

Regarding ordinal encoding, the most discrete features in this dataset are categorical. Their categories are not comparable to nominal features. For example, (bronze medal, silver medal, gold medal) is a nominal feature because bronze < silver < gold, while (apple, orange, banana) is a categorical feature because we can not compare fruits without placing them in a specific context. In this report, we perform ordinal encoding for categorical features in this exercise with a grain of salt. We know that one hot encoding is way better than ordinal encoding in terms of accuracy, but the downside is that it explodes a single categorical feature with n categories into at least n-1 binary variables. This is very inefficient in computational cost and ML training speed, especially when dealing with big data. Therefore, ordinal encoding is an acceptable technique that can work well for this dataset, whatever the order is. A modern technique called EFB (exclusive feature bundling), used in the famous LightGBM algorithm is designed to transform one-hot encoded variables into a single ordinal encoded variable. However, this dataset is not very big, so one-hot encoding is still worth trying.

Regarding Multilabel encoding, we believe this would be a plus point. Using ordinal encoding for the “benefits” feature is technically easy, but it will be a trade-off for much helpful information. Another advantage of using multi-label encoding is that we can analyze feature importances to assess how much each category contributes to the final results after modelling.

About scatter plots, this plot is designed for visualizing two continuous variables. To plot the relationship between a continuous and a discrete variable, we suggest using swarm plot, a variant of scatter plot in which data points do not overlap. Sampling should also be used to

make the plot easy looking. Unfortunately, most variables in this dataset are categorical, so scatter or swam plots do not work very well.

About estimating correlations, since most of the features and the target are categorical, We suggest using the Chi-square test, Kendall's tau or Cramer's V to get a better idea of how a pair of categorical features relates to the other.

Regarding Naive Bayes, this dataset contains features of different levels of measurement. Thus, to use Naive Bayes correctly, we must train a suitable learner (a suitable assumption of distribution) for each group of features, such as Gaussian Naive Bayes for continuous features such as "age" and Categorical Naive Bayes for categorical features and Bernoulli Naive Bayes for binary features. Then stack their results by multiplying the probability prediction of every single model.