Faculty of Natural and Mathematical Sciences
Department of Informatics

Bush House, King's College London
Strand Campus, 30 Aldwych
London WC2B 4BG
Telephone 020 7848 2145
Fax 020 7848 2851

7CCSMPRJ

Individual Project Submission 2018/19

Name: Smith Kakar

Degree Programme: Intelligent Systems

Project Title: I know you are lying: Predicting truthfulness in data sharing

Word Count: 12450

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☑ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: Smith Kakar **Date:** August 22, 2019

Note: This is a public reference copy after removing the supervisor and data source details to protect user privacy and security. The appendix section is also in concise format in this version.

Department of Informatics
King's College London
United Kingdom

7CCSMPRJ Individual Project

I know you are lying: Predicting truthfulness in data sharing

---

Name: **Smith Kakar**
Course: Intelligent Systems

This dissertation is submitted for the degree of MSc in Intelligent Systems.

Abstract

With digital transformation, data is the most valuable asset for organisations to gain competitive advantage. We are sharing data with companies on an unprecedented scale who in turn share it with other companies. Organisations often provide personalised services to targeted customers after processing this data. Hence there is an ever growing concern about how businesses collect and use personal data. There is yet another perspective on this where users are found to either withhold or falsify the information when they engage in privacy protection behaviours. Therefore, businesses end up with unreliable data that is not useful. What if we knew what type of information and in which context users will be likely to lie about? This would make users feel their privacy is respected by the provider, as they will not be asked to provide that information, and the provider would know that only reliable data will be collected from users, thus having a win-win situation for both users and providers. Unlike existing literature that mainly focuses on *statistical analysis* for solving the similar problem, we found this to be intractable for large data-sets and therefore conducted our research by building *machine learning* models to learn from user data and predict outcomes with high balanced accuracy. In this thesis, we significantly improved the classification performance by incorporating *imbalanced* and *cost-sensitive* learning methods to deal with imbalanced data. We also segmented similar user behaviours into groups using *unsupervised* machine learning techniques. Moreover, structural relationships were identified within data using *probabilistic reasoning* methods.

Nomenclature

AI	Artificial Intelligence
CV	Cross Validation
FN	False Negative
FP	False Positive
FPR	False Positive Rate
KNN	K-Nearest Neighbour
ML	Machine Learning
PCA	Principal Component Analysis
ROC	Receiver Operating Characteristic
std	Standard Deviation
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
TNR	True Negative Rate
TPR	True Positive Rate
$V1.x$	Version 1.x
$XGBoost$	e**X**treme **G**radient **B**oosting

Contents

List of Figures

List of Tables

1 Introduction

Today, data scientists around the globe are working tirelessly to gain real insights into data to make informed decisions for the businesses. With digital transformation and cashless economies on rise, data is the most valuable asset for organisations to gain competitive advantage. Nowadays most people read online articles and create their profiles to access website content, use social media and/ or execute transactions. Clearly, people are sharing data with companies on an unprecedented scale who in turn share it with other companies. Organisations process this data and often provide personalised services to targeted customers.

Consequently, there is an ever growing concern about how businesses collect and use personal data in several domains- e-commerce, finance, online social networks, retail and technology to name a few. Recently, people have also become aware of how Cambridge Analytica used personal data of millions of Facebook users to influence political elections in the UK [1]. Therefore, privacy and security have been the most talked about tech stories for the last two years. Simultaneously, General Data Protection Regulation (GDPR) has also come into effect as data protection is part of the fundamental right to privacy.

Nevertheless, the existing research has also shown that users might either withhold or falsify the information when they engage in privacy protection behaviours. As a result, businesses often end up with unreliable data that is not useful.

What if we knew what type of information and in which context users will be likely to lie about? This would make users feel their privacy is respected by the provider, as they will not be asked to provide that information, and the provider would know that only reliable data will be collected from users, thus having a win-win situation for both users and providers. This motivate us to do research in this area by utilising machine learning techniques to predict in advance whether users are likely to be truthful or not about data. Our main contributions of this dissertation are:

- We perform data analysis and pre-processing on the given data-set before building the machine learning models.
- We apply supervised learning techniques using state-of-the-art estimators to ensemble models to learn from user data.
- The user responses are predicted with high balanced (average) accuracy of truthfulness and falsehood.
- We train and evaluate models using different learning methods- over-sampling, under-sampling and cost-sensitive learning in case of imbalanced data.
- We present the comparison results between different methods in which ensemble samplers (using under-sampling) and cost-sensitive based models turned out to be the clear winner.

- Machine learning models are also trained and evaluated using all features and only informative features for comparison.
- We apply unsupervised learning techniques- K-Means and PCA to identify patterns/ segments in data.
- The probabilistic models are evaluated to learn structural relationships, particularly to draw inferences on how user answers relate across different questions.

1.1 Aims and Objectives

The main aim of this project is to build a machine learning model that learn from existing user data and predict whether new users are likely to be truthful or not about request for the information. The objectives during the course of this project are as follows:

- The given data-set will be analysed to understand the underlying distribution and identify any correlation between variables.
- The design choices will be devised using the outcomes from the data analysis phase.
- The type of machine learning technique and the corresponding workflow will be defined based on our design choices.
- The data will be prepared before training the machine learning model.
- The model will be trained and evaluated using well established procedures.
- Model parameters will be tuned in order to increase efficiency.
- The results will be evaluated while comparing the different methodologies.
- The conclusions and limitations of the current work will be highlighted to discuss possibilities of the future work.

The rest of the paper is structured according to the following narrative. First, we will provide a background and literature survey on this topic in the next section. In Section 3, we will analyse the data using descriptive statistics and visualisation before designing our specifications for the machine learning techniques applicable for this project. We will cover the supervised learning, unsupervised learning and probabilistic reasoning methods to gain deeper insight into data and give a breath to this project. The alternate designs will also be considered at each stage during the process. Moreover, the data preparation is included in this section for better understanding of representation of the data before going into the implementation details.

Section 4 covers the methodologies and implementation for supervised learning, unsupervised learning and probabilistic methods separately. The binary classification as part of supervised learning is carried out step-by-step in three phases- as-is, imbalanced and cost-sensitive learning. The benefits of using this approach are two-fold. First,

the categorisation allows us to compare and contrast the methods in a lucid manner. Second, the lessons learnt during each stage can be applied during multiclass classification and other relevant methodologies. Further, K-Means clustering algorithm, when applied on data with informative features, revealed interesting patterns. On the other hand, Bayesian networks produced astonishing results on learning structural relationships among user answers across all questions. We also present the project structure with respect to code-versioning and Python library usage details at the end of this section.

Furthermore, the results are analysed and evaluated including limitations in Section 5. Our study found significant performance improvement using Under-Sampling and Cost-Sensitive learning methods in case of imbalanced data. The results are also obtained and analysed using all the features and only informative features during classification. Finally, the last section summarises the paper and draws some conclusions from this project.

2 Background and Literature Survey

In this section, we will provide a brief background on Machine Learning before studying the existing literature related to the project work.

2.1 Machine Learning

After two rounds of *AI winter* in 1970s and early 1990s, AI is in hype once again after recent breakthroughs in a number of applications using deep learning. While the venture capital investment in AI has risen to a staggering amount in last five years, big tech companies are also spending millions of dollars in their research departments. Artificial Intelligence is a general field that includes machine learning and deep learning as illustrated in Figure 1.

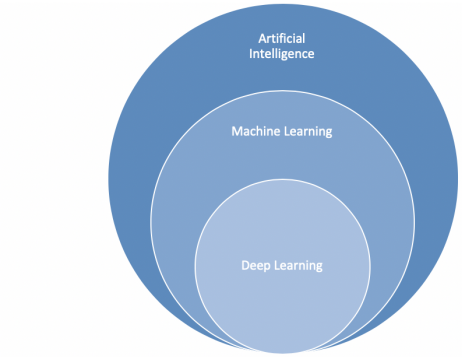

Figure 1: Artificial Intelligence, Machine Learning and Deep Learning

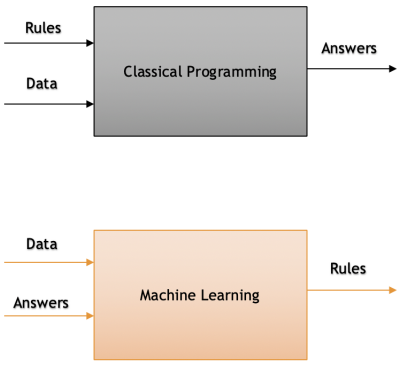

Figure 2: Classical vs Machine Learning Programming Paradigm

The main idea behind AI is to build intelligent machines that automate intellectual tasks performed by humans. In symbolic AI, humans make use of data and rules to get the answers. Programmers would define these data-processing rules manually in the code. However, it is intractable to craft rules by hand as the data grows often with distinct characteristics. The machine learning in this case helps us to input data and answers (expected from data) in order to get the rules. This programming paradigm is shown in Figure 2. Now we can apply these rules on new data to predict the answers.

In essence, a system using machine learning model is trained on many examples of existing data instead of programming the explicit rules. For example, if we wish to automate the process to identify whether it is a cat or dog in a picture, we will train a machine learning model with many examples of pictures labelled as cat and dog, and the model will learn patterns/ rules in data for identifying a cat or dog in pictures.

2.2 Machine Learning Techniques

The machine learning techniques can be grouped by learning method as follows:

Supervised Learning: In supervised learning [2], input data comes along with the output in the form of label (e.g. False/ True, Spam/ Not Spam etc.). It is a function approximation method in which a model is trained on the training data to predict the labels of the unseen data. Further, the model is properly tuned to achieve a reasonable accuracy. A typical supervised learning task would be *classification* or *regression*. A classification is used to classify/ categorise a new observation whereas a regression is typically used to predict a target numeric value, such as the price of a stock, given a set of predictor variables also called features.

Unsupervised Learning: In unsupervised learning [3], input data does not have la-

bels. The main aim is to identify patterns/ segments in the data. These cluster labels can further be utilised to run supervised learning algorithms for prediction. A typical task in unsupervised learning can be *anomaly detection* [4], *association rule learning* [5] or *dimensionality reduction* [6].

Semi-supervised Learning : In semi-supervised learning [7], input data contains both labelled and unlabelled data. The machine learning models in this case are trained mainly on unlabelled data including a small portion of labelled data. This is beneficial for few reasons. First, the process of labelling the data is time-consuming and expensive. Second, there is a chance of human bias being imposed on input data with too much labelling. Thus, this technique is found to improve the accuracy of the model while reducing the cost and time building the model. One example of a service using semi-supervised learning would be *Google Photos*.

Reinforcement Learning: In reinforcement learning [8], the agent is trained to learn in an environment using its own actions and experiences by trial and error. The feedback is provided to the agent in the form of rewards and punishment for positive and negative outcomes. The idea is to let the model learn by itself the best strategy, called a *policy*, that would maximise the total cumulative reward over time.

2.3 Type of Machine Learning Systems

The type of machine learning systems can be classified based on below categories:

Human Supervision: The type of machine learning systems can depend on whether or not they are trained with human supervision. They include supervised, unsupervised, semi-supervised and reinforcement learning methods.

Batch versus Online Learning: In online learning, the machine learning systems learn incrementally on the fly. On the contrary, learning happens on the entire training data at once using batch learning techniques. Therefore, the systems differentiate whether they are trained incrementally using data instances sequentially, either individually or in small groups called mini-batches, or entire training data-set at once.

Instance versus Model-based Learning: The machine learning systems can also be based on whether they work by comparing new problem instances to known trained instances as in instance-based learning. On the other hand, predictive models are built by detecting patterns in the training data using model-based learning. A model based approach in machine learning can also be used to create a custom model tailored specifically to each new application [9].

The above techniques are not exclusive and can be combined while building machine learning systems.

2.4 Comparison of Popular Machine Learning Algorithms

In this section, we will provide an overview of some popular machine learning algorithms while comparing their functionalities.

Decision Trees: Decision trees [10] are versatile machine learning algorithms that are capable of performing supervised learning tasks such as classification and regression. The *Classification and Regression Tree* (CART) algorithm produces only the binary decision trees in which every node except leaf nodes have only two children. The algorithm starts by splitting the training set in two subsets using a feature that will have the least impurity measure such as *gini* or *entropy*. The process repeats for each subset recursively once it reaches the maximum depth or if further split is not possible that will reduce the impurity. A major advantage of using decision trees is that they do not require feature scaling or centring. Therefore, we have to do very little data preparation before using this algorithm or other related algorithms such as ID3, C4.5 and C5.0. We can also visualise the trained decision tree to see how decisions were made at each node. On the other hand, the decision trees also need to be constrained in order to avoid over-fitting the data. This can be achieved by imposing constraints on parameters like *max_depth*, *min_samples_split*, *min_samples_leaf* etc.

Support Vector Machines (SVM): Similar to decision trees, SVMs [11] can also perform classification as well as regression. They are capable of performing linear or nonlinear classification. The idea is to learn a decision boundary, a hyperplane in this case, that maximises the separation (margin) between classes. The non-linear classification is performed by using *kernel trick* that implicitly maps the inputs into high-dimensional feature space. Unlike decision trees, SVMs are sensitive to feature scales and thus perform better after feature engineering. The SVM model is regularised by reducing the value of C hyperparameter. A classifier with high C value makes fewer margin violations due to a smaller margin whereas a low C value makes more margin violations with a larger margin.

Random Forests: Random forests [12] are essentially a group of decision trees trained on different random subsets of the training data. The prediction is done by voting of all individual trees. Henceforth, they predict the class that gets the most votes. Generally, a group of models/ predictors is termed as an *ensemble*. There are several ensemble learning algorithms, including *bagging*, *boosting*, *stacking*, and a bunch of others [13] [14] [15], [16]. The ensemble's accuracy is often higher than the best classifier in the ensemble. We can use diverse classifiers to further improve the performance. Unlike decision trees which are called *white box models* as they can be visualised and their decisions can be interpreted easily, random forests are considered as *black box models*. Furthermore, the random forest algorithm searches for the best feature among a random subset of features rather than all the features for splitting a node and thus produces an overall better model due to more diversity.

Extra Trees: Extra trees is a short name for *Extremely Randomised Trees* [17] which is a forest of extremely random trees. While random forests consider the best feature among a random subset of features to split a node, extra trees opt for random thresholds for each feature rather than looking for the feature with the best possible thresholds. Therefore, these models run faster in comparison to others during the training time.

AdaBoost: This is one of the most popular boosting algorithms. Unlike the bagging methods, models in any of the boosting methods are trained sequentially while trying to improve their predecessors' predictions. In Adaptive Boosting, the weight of misclassified (or poorly predicted in case of regression) training instances are increased at every iteration after each classifier makes its predictions on the training set. Finally, all of the successive models are weighted according to their overall accuracy on the training set and the prediction is made by weighted voting (for classification) or averaging (for regression) of the trained models. The regularisation in this case can be achieved by reducing the number of estimators in the ensemble or further regularising the base estimator.

XGBoost: XGBoost [18] is the one of the most popular algorithms among Kaggle [1] competitors. This has been considered as the most powerful algorithm for structured data in recent years. It stands for e**X**treme **G**radient **B**oosting. It can perform *gradient boosting*, *regularised boosting* as well as *stochastic boosting*. While boosting algorithms like AdaBoost put more weight on the misclassified examples in every iteration, gradient boosting algorithms fit new models to predict the residuals or errors of prior models and then they are added together to make the final prediction. A gradient descent algorithm is used to minimise the loss when adding new models. A major advantage of this algorithm is that it provides improved model performance and computational speed over other boosting algorithms by allowing automatic parallel computation using multiple CPU cores on a single machine. This algorithm was designed specifically for decision tree boosting with custom regularisation term in the objective function. Similar to decision trees and random forests, it also supports both classification and regression problems.

This is not a complete list of machine learning algorithms. There are several other widely used algorithms such as *K-Means* [19], *KNN* [20], *Linear Discriminant Analysis* [21], *Linear Regression*, *Logistic Regression* [22] (which is confusingly a classification algorithm), *Naive Bayes* [23], *PCA* [24] and more. An excellent empirical evaluation of classifiers has been carried out in [25].

2.5 Deep Learning

There is an empirical evidence that basic machine learning algorithms work well on structured data. However, they do not provide state-of-the-art performance for complex

[1] https://www.kaggle.com/

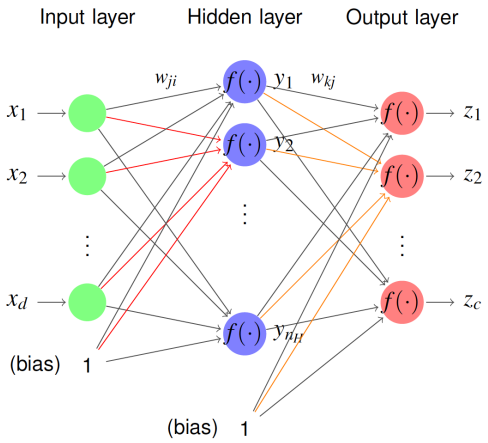

Figure 3: An Example of a Three-Layer Fully Connected Neural Network

problems such as image classification, speech recognition and natural language processing to name a few. Deep learning [26] is a sub-field of machine learning that learn meaningful representations of data via successive layers i.e. in a multistage way. Each layer is associated with weights and the learning process involves finding a set of values for these weights in a network with the end-goal to map example inputs to their associated targets. One such example from [27] is illustrated in Figure 3.

Further, the output of neural network is controlled by the objective function which measures how far the predicted output is from actual target and computes the distance score. This score is then used to slightly adjust the value of the weights in order to reduce the loss for the particular example. The process runs in a training loop with several iterations over many examples which yields near optimal weight values that minimise the loss function. A trained network is the one for which the predicted outputs are as close as possible to the target values.

Although we have not implemented any deep neural networks in this project but it is worthwhile to talk about them as AI is accelerating quickly across different sectors only due to major breakthroughs with deep neural networks. A more comprehensive survey of machine learning algorithms can be studied in [28], [29].

2.6 Related Work

In this section, we will review some of the existing literature in the context of data sharing and privacy.

"Privacy is the claim of individuals, groups or institutions to determine for themselves when, how, and to what extent information about them is communicated to others" [30]. Malhotra et al. [31] developed a theoretical framework based on social contract theory [32] to examine Internet users' information privacy concerns (IUIPC). Their findings reveal that the IUIPC factors - *collection*, *control* and *awareness* exhibit desirable

properties in the context of online privacy and explain a large amount of variance in consumers behavioural intention to disclose or not disclose their personal information.

However, Grossklags and Alessandro [33] argued that the information protection is not based on data disclosure but depends on behavioural processes and different marketplace activities. The empirical evaluation of willingness-to-protect and willingness-to-accept for protecting and selling information respectively reveals that users tend to accept a proposal to sell their information for economic benefits rather than protecting their personal information. On the contrary, individuals may also either withhold [34] or falsify [35] the information in order to protect their privacy in case of no monetary advantage.

Malheiros et al. [36] describe an online field-experiment in a financial context detailing the impact of perceived *effort*, *fairness*, *relevance* and *sensitivity* on personal data disclosure. Although the effect of perceived fairness was not thoroughly examined in privacy research, their results show that the fairness has a significant positive effect on the disclosure and truthfulness of data items. Contrary to popular belief, the effect of perceived relevance is not found to be significant apart from few data items. Disclosure and Truthfulness ratings of each data item were regressed on these factors using Nagelkerke's R^2 model [37]. This model summarises the proportion of variance in the dependent variable associated with the predictor variables.

Differential privacy [38] is another interesting aspect which is a constraint imposed on algorithms to limit the disclosure of private information of users whose information reside in the statistical database. In essence, we will not be able to tell from the output whether any individual's information was part of the data-set or not. This guarantees that the individual's information is not leaked from the database. Hence, it enables social scientists to share meaningful statistical findings about sensitive data. [39] defines this notion and highlights some interesting applications. The differential private algorithms quantify the *privacy loss* using *epsilon* and *delta* parameters and are being applied not only in statistical analysis but also machine learning, game theory and more.

A significant amount of work has been done in the context of privacy-preserving machine learning. Abadi et al. [40] train deep neural networks with new algorithmic techniques based on a deferentially private version of stochastic gradient descent (SGD). They employ advanced privacy-preserving mechanisms by using several layers and thousands of parameters with non-convex objective functions during the training process. Consequently, the machine learning system using deferentially private algorithms will provide protection against those attacks which have access to the model's parameters and knowledge of the training methods.

While most of the existing literature focus on statistics tools to support or reject the hypotheses for the similar problem, the main limitation of statistical inference is that it is impractical to deal with large, complex data-sets in which each sample further consists of large number of features. Subsequently, our goal is to use machine learning techniques to predict whether a user is likely to be truthful or not about sharing data.

3 Objectives, Specification and Design

Our business objective is to predict whether users are likely to be truthful or not about request for the information using Machine Learning techniques. The solution, the type of information users will be likely to lie about, can be used to honour user privacy and not request that piece of information to avoid getting unreliable data in future. The steps that we will perform in this project are inspired from [41] and are illustrated in Figure 4. This is a high-level overview of our machine learning pipeline and particularly applicable to supervised learning technique. The details will be covered in subsequent sections.

Data Collection: The first step in any machine learning project is to gather the data. The data might be coming from different sources or any other component in the overall system. We will discuss more on this including the data analysis part later in this section.

Data Preparation: This usually involves data cleaning, feature selection and feature engineering [42] tasks to be applied to a given problem. The data-set is first split into training and test subsets. It is usually done in the ratio of 80:20 or 70:30 as such. Then the training data is prepared/ transformed using appropriate methods as covered in Section 3.4.

Choose a Model: The next step in our ML pipeline is to choose a model. We have several state-of-the-art estimators available at our disposal. The models vary according to the type of learning- supervised, unsupervised or other machine learning techniques applicable for the given problem. The choice further depends on the goal within each learning method. For instance, the models may vary according to the binary or multi-class classification in supervised learning.

Training: Once the model is selected for the task in hand, the next thing to do is to train that model on the training data.

Model Evaluation: After training, the model is evaluated on the validation data. The validation data can be obtained after splitting the training data. However, if the training data is less, then the evaluation can be carried out using *cross-validation*. In k-fold cross-validation, the training data is split into k smaller sets. For each k-fold, a model is trained using $k-1$ folds considered as training data and the trained model is further validated on the remaining part of the data. The performance score is the average of the values obtained from these folds. An example of 5-fold cross-validation [43] is illustrated in Figure 5.

Hyperparameter Tuning: Each model comes with parameters which need to be tuned to avoid over-fitting on the training data. The idea is to generalise the model to make it perform better on unseen data. This is one of the critical steps to obtain good results

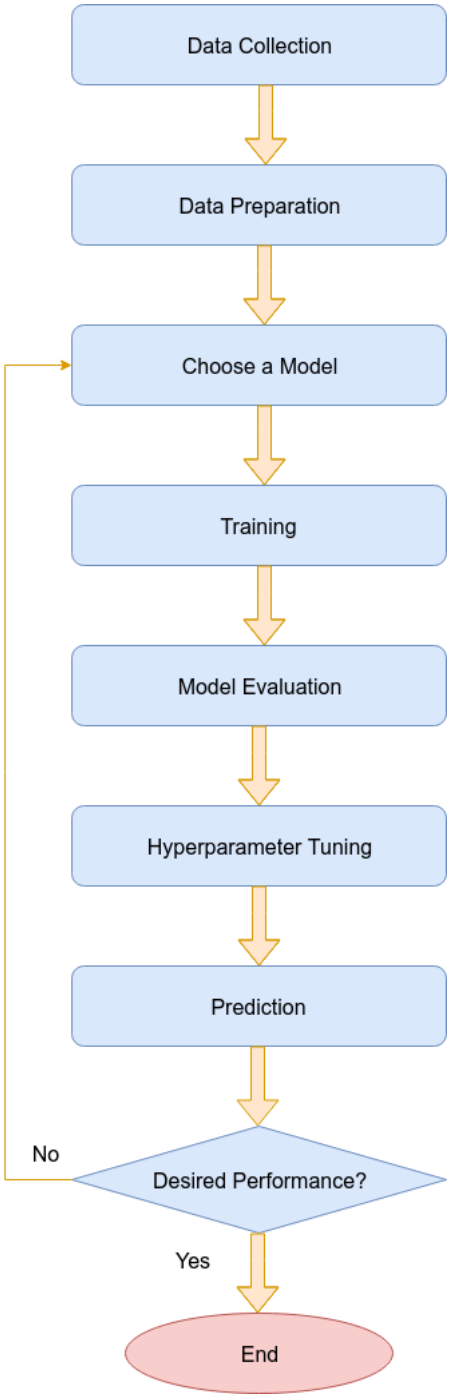

Figure 4: ML Pipeline

on validation data.

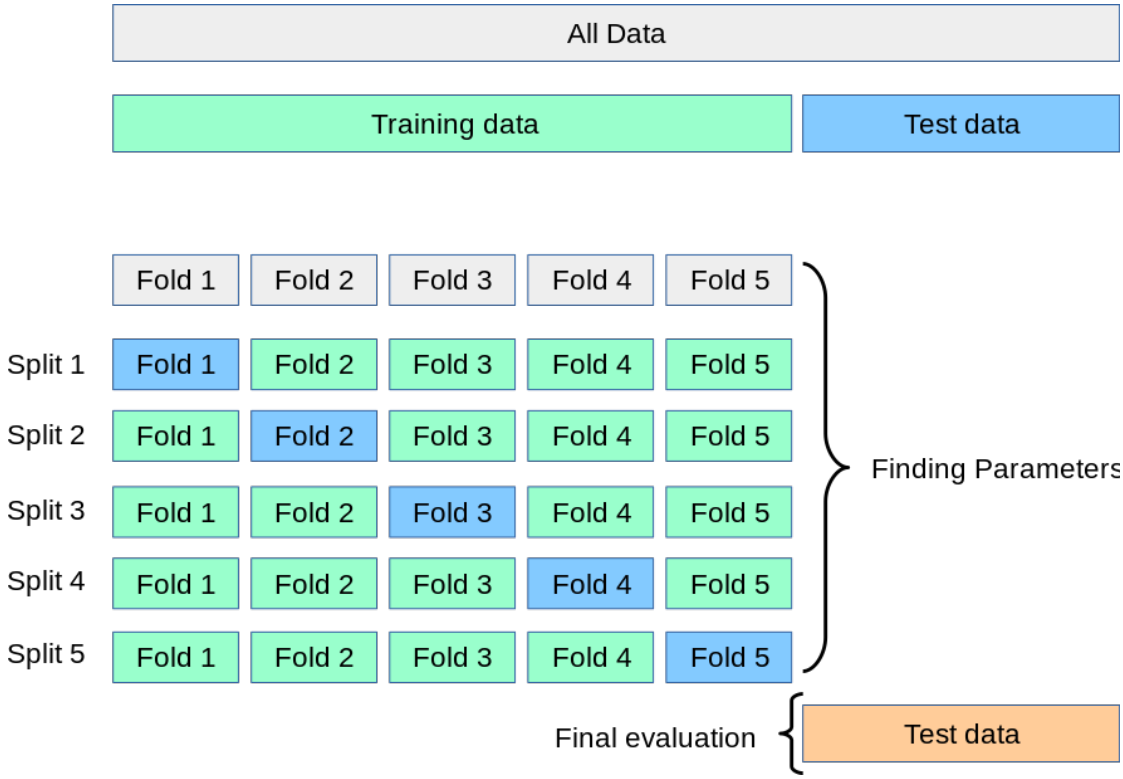

Figure 5: 5-fold Cross-Validation

Prediction: The next step is to do prediction on unseen data using trained model. Before prediction, the test data needs to be transformed exactly in the way training data was prepared in the second step of this process.

Finally, if this model does not perform well on the test data, we go back to step 3 to choose a different model and execute the further tasks in the pipeline.

3.1 Data Collection

The data-set was provided for this project. It was collected through a field-experiment in the context of purchasing discounted movie tickets. The participants were presented a questionnaire with different sections. In one section, the questions were about the data asked for (e.g. Name, DoB etc.) and the feedback for each question in the form of - *effort*, *relevance*, *uncomfortable* and *truthfulness* measures. We suspect that the first three factors influence the truthfulness of data items. The questions in other section were about demographics information and general privacy concerns. A total of six surveys

Table 1: Question Block-1

No.	Question
3	DoB
4	Birth Country
8	Home Address
12	Workplace Travel
16	Ethnicity
17	Politics
22	Memorable Event
25	Study/Education
28	Alcoholic Beverages
30	Favourite Mobile Brand
34	Author
37	Government Banned Movie
38	Web Browser
40	Favourite Actor/Actress
44	Last movie at cinema
48	Go to cinema with
49	Rent adult movies

with different order of questions were maintained in order to increase the quality of the data-set to avoid questions being asked in the same order.

The actual set of questions was distributed across three chunks with 17 questions each in Block 1 and Block 2 and remaining 16 questions in Block 3. Each user participating in the survey was given seven questions each from Block 1 and Block 2 and six questions from Block 3 respectively. The questions in each block are shown in Table 1, Table 2 and Table 3. The sequence of questions in each survey were presented according to the block order maintained for that particular survey. This is laid out in Table 4. For example: In "MovieUp - 2" survey, first seven questions were from Block 1, then next six questions from Block 3 and remaining seven questions from Block 2. This was randomised only to get quality data.

3.2 Data Analysis

Data analysis for this project is performed using open-source libraries in Python. We explored the second data-set that includes survey responses without the actual answers. That is we have the anonymised version of the data-set. The data-set includes *Age* (in years), *Gender* (Female=1, Male=2, Other=3), *Online Presence* (number of hours

Table 2: Question Block-2

No.	**Question**
2	Gender
6	City of Residence
10	Workplace Postcode
13	Personal Email
14	Professional Email
18	Religion
19	Sexual Orientation
24	Illnesses
27	Hobby/Pastime
31	Hurt Sentiments - Movie
32	Holiday Destination
35	Music Genre
39	Age for Adult movie
41	Favourite Movie
45	Money on cinema weekly
47	Illegal streaming/downloading
50	Favourite Pornstar

Table 3: Question Block-3

No.	Question
1	Name
5	Country of Residence
7	Home Postcode
9	Employer Name
11	Work Address
15	Phone Number
20	Relationship Status
21	Lied to Partner
23	Languages
26	Annual Income
29	Shared X-rated movies
33	Lied about Age
36	Musician
42	Favourite Movie Genre
43	Favourite Soundtrack
46	Online rental subscriptions

Table 4: Question Block Orders

Survey Name	First	Second	Third
MovieUp - 1	1	2	3
MovieUp - 2	1	3	2
MovieUp - 3	2	1	3
MovieUp - 4	2	3	1
MovieUp - 5	3	2	1
MovieUp - 6	3	1	2

Table 5: Summary Statistics

Measure	Age	Gender	IUIPC-Awareness	IUIPC-Collection	IUIPC-Control	Online-Presence	Personal-Stability	Reciprocity
count	827	827	827	827	827	827	827	827
mean	31.58	1.56	1.88	2.28	2.33	6.63	4.20	4.02
std	10.75	0.53	1.53	1.45	1.39	3.85	0.87	0.82
min	18	1	1	1	1	0	1.8	2
25%	24	1	1	1	1	4	3.6	3.33
50%	29	2	1	1.75	2	6	4.2	4
75%	36	2	2	3.125	3	9	4.8	4.5
max	98	3	7	7	7	24	7	6.83

spent online in a day) and scores (7-point Likert scale) for *Reciprocity*, *Personal Stability*, IUIPC factors [31] consisting of *IUIPC-Control*, *IUIPC-Awareness* and *IUIPC-Collection*. There are no missing values for these variables. The statistics measures of this data are shown in Table 5. Further, the scores for feedback questions measured on 7-point Likert scale are also available in this data-set. The summary statistics of these variables are not presented here for brevity as there will be a total of 200 columns against 50 questions. We have significant number of missing values for all these 200 variables. This is due to the reason that only 20 questions were originally asked in the survey. Accordingly, we have the feedback for 20 questions from each user. Lastly, the records were discarded for users who did not complete the survey correctly. Therefore, such records are not part of this data-set.

3.2.1 Type of Data

The data type of each variable except *Age*, *Gender*, *Online Presence* and *Prolific ID* is *float64*. The category and data type of variables are listed in Table 6. We have missing values for only feedback questions as discussed in the previous section. The target attribute/ variable from supervised learning perspective would be *01-Truthfulness* for 1st question, *02-Truthfulness* for 2nd question and so on.

3.2.2 Data Distribution and Noise

In this section, we will explore the demographics data first and then perform high-level analysis on IUIPC factors and feedback question data. The distribution of data excluding feedback questions using histogram plot is shown in Figure 6.

The distribution is skewed to the right for *Age* and *Online Presence* data. This implies we will have to transform this data to make it normalised in order to get reasonable performance with ML algorithms as some models are sensitive to the skewed distributions. The data for all other variables is not normally distributed by definition as they are ordinal variables except *Gender* which is a categorical variable. Figure 7 displays the histogram plot of feedback answers for Question 4. We have plotted the histograms of the target variable (*01-Truthfulness*) with respect to other variables in Figure 8, 9, 10, 11 and 12. This visualisation is very helpful when we are working on the data-set with one or two target variables as it can help in eliminating irrelevant attributes and

Table 6: Type of Data

Variable	**Category**	**Data Type**	**Missing Values**
Prolific ID	Alphanumeric	object	No
Age	Continuous	int64	No
Gender	Categorical	int64	No
IUIPC-Awareness	Ordinal	float64	No
IUIPC-Collection	Ordinal	float64	No
IUIPC-Control	Ordinal	float64	No
Online-Presence	Continuous	int64	No
Personal-Stability	Ordinal	float64	No
Reciprocity	Ordinal	float64	No
01-Effort	Ordinal	float64	Yes
01-Relevance	Ordinal	float64	Yes
01-Truthfulness	Ordinal	float64	Yes
01-Uncomfortable	Ordinal	float64	Yes
02-Effort	Ordinal	float64	Yes
...	...	...	...
...	...	...	...
...	...	...	...
50-Uncomfortable	Ordinal	float64	Yes

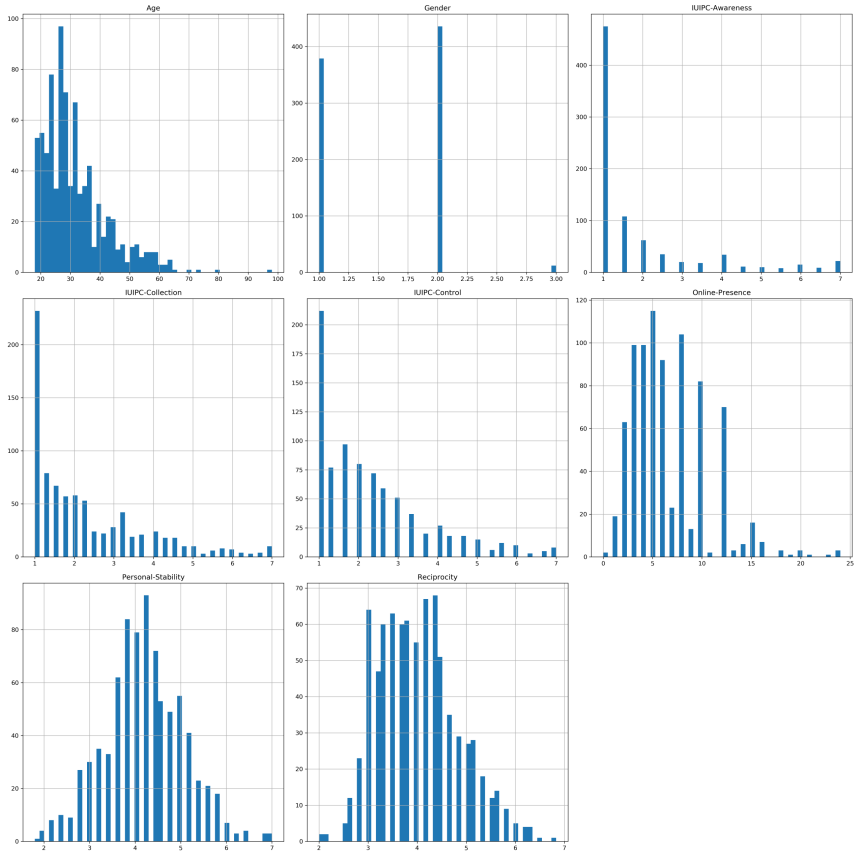

Figure 6: Histogram Plot of Demographics and IUIPC Factors

can also tell us the ideal accuracy we should achieve during prediction. However, this will be cumbersome in our case as we have 50 target variables against 50 questions.

Some other observations from Table 5 are as follows:

- 45.83% Females and 52.72% Males filled out the survey with 1.45% belonging to Other category.
- The mean age of people filling out the surveys is found to be 31.58 years with std of 0.53 as shown in Table 6.
- The average number of hours spent online by males, females and others are found to be 7.09, 5.97 and 10.92 hours respectively.

3.2.3 Data Correlation

Correlation statistical method determines the association between random variables. Our purpose is to identify if there are any highly correlated features. The highly corre-

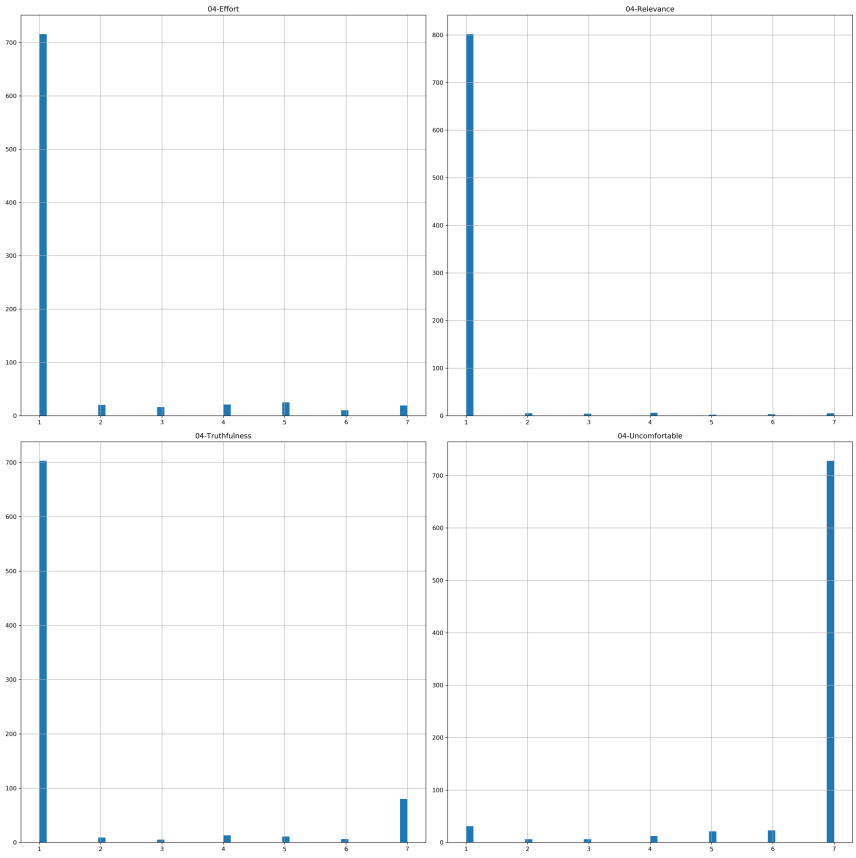

Figure 7: Histogram Plot of Feedback Q04

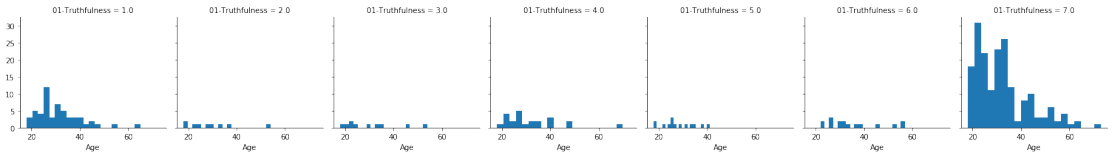

Figure 8: Histogram Plot of Q01-Truthfulness by Age

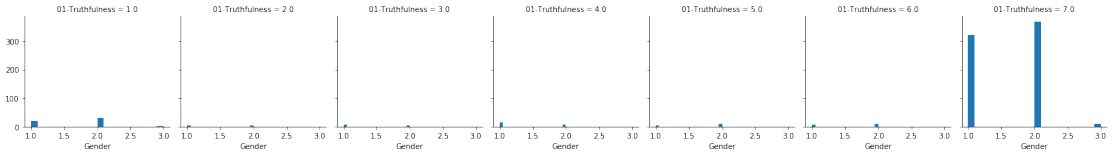

Figure 9: Histogram Plot of Q01-Truthfulness by Gender

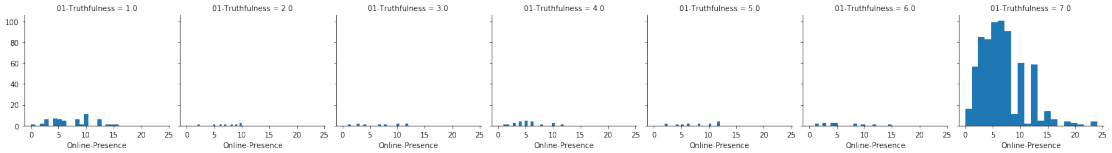

Figure 10: Histogram Plot of Q01-Truthfulness by Online Presence

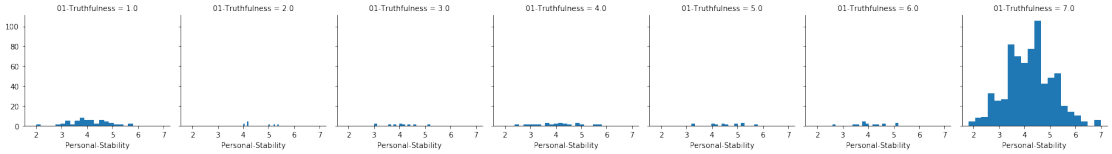

Figure 11: Histogram Plot of Q01-Truthfulness by Personal Stability

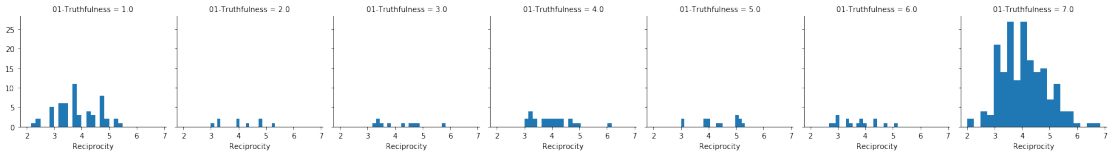

Figure 12: Histogram Plot of Q01-Truthfulness by Reciprocity

lated features are not found to be useful during prediction. Instead, they add unnecessary complexity and make it hard for estimators to predict output with high accuracy.

First, we have plotted the scatter plot matrix of each column against every other column. Figures 13 and 14 indicate that there are no high positive or negative correlations between any variables. The diagonal here represents the density plot of the particular variable. However, we will also take another measures before making any conclusions.

Pearson correlation coefficient [44] identifies the linear correlation between continuous variables. Rodgers and Nicewander [45] explain thirteen ways to look at the correlation coefficient. Since we have mostly ordinal data, Spearman or Kendall's Tau [46] coefficient will be a better statistical measure in this case. Figure 15 displays the correlation heatmap for nine questions. The correlation between *IUIPC-Awareness* and *IUIPC-Control* is found to be around 0.6 and consistent across questions. Similarly, the correlation between *effort* and *truthfulness* is found to be around -0.6 for most of the questions. However, none of the variables are found to be highly correlated ($>= 0.8$ or $<= -8$) across all questions. Therefore, we will not remove any variables from our data based on these findings.

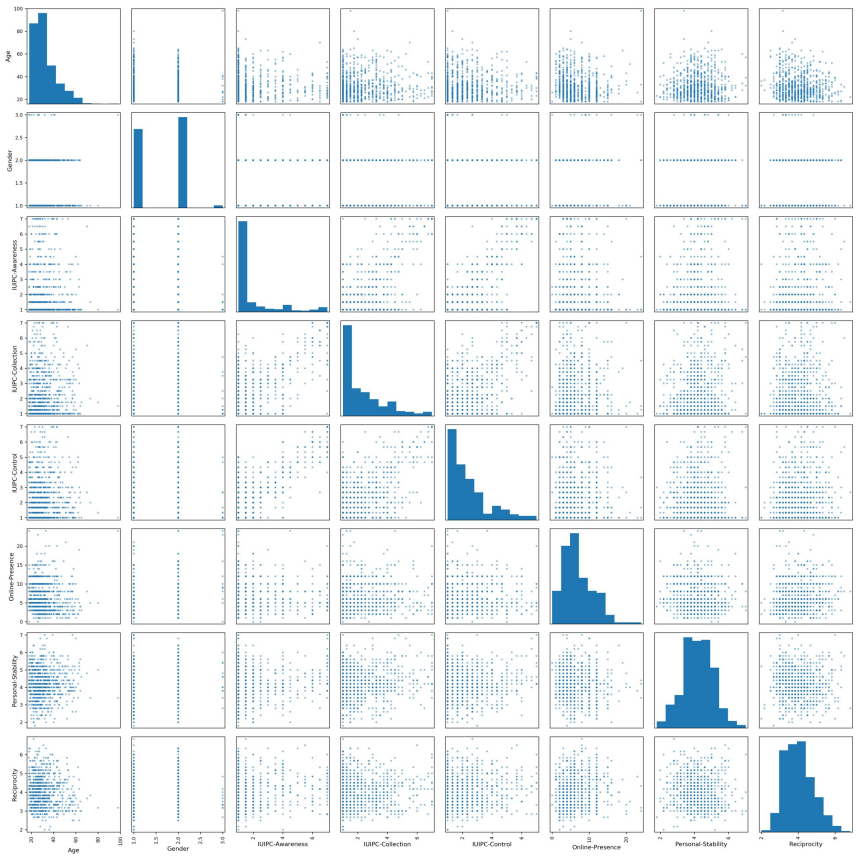

Figure 13: Scatter Plot Matrix of Demographics and IUIPC Factors

3.3 Design

The study of existing literature and data analysis are found to be extremely helpful to make better design choices. In this section, we will discuss our proposed design for the given problem.

3.3.1 Supervised Learning

The choice of model depends on the problem as well as the learning method. During data analysis, we found out that there is a target variable (label) for each question. In other words, there is a category label for each feature vector in the given data-set. Therefore, the current problem to predict truthfulness in data falls into what supervised learning methods imply in the ML toolbox.

In classification problem, we assume that there is an unknown target function (f)

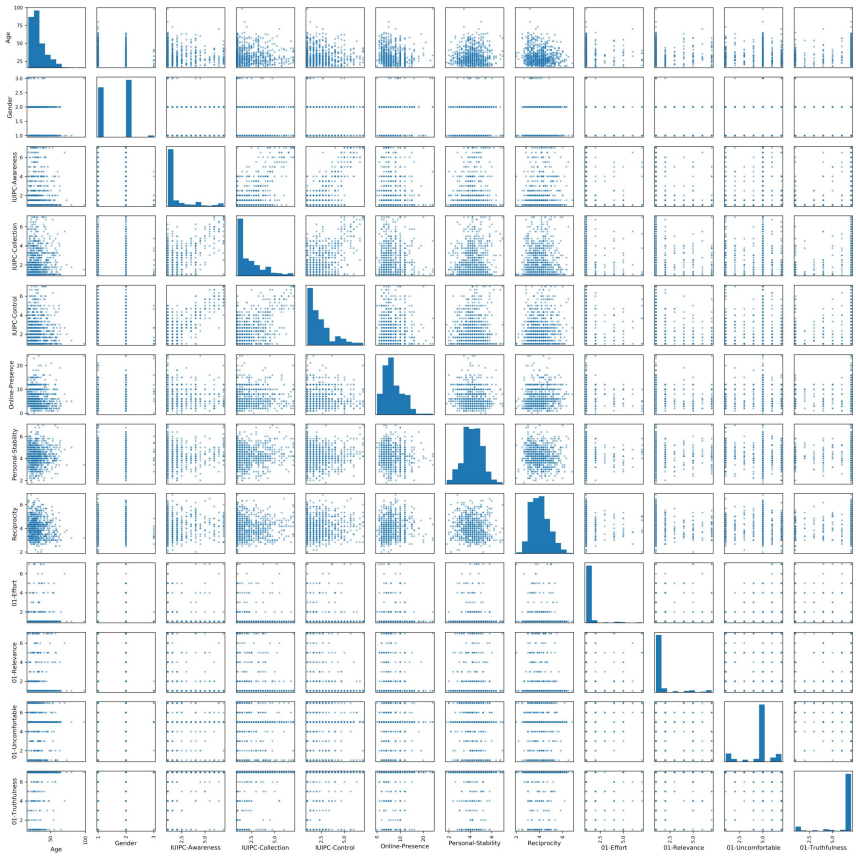

Figure 14: Scatter Plot Matrix of All Features for Q01

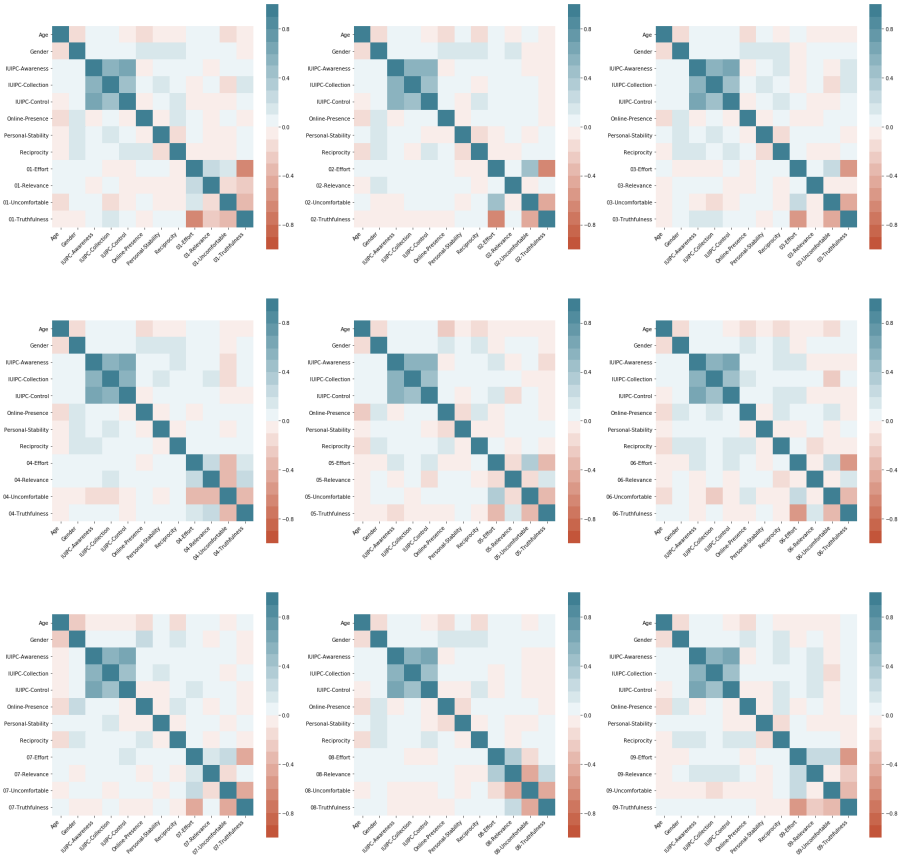

Figure 15: Correlation Heatmap for Q01-Q09

that will map feature vectors (x) to class labels (y) as:

$$y_j = f(x_j) \tag{3.1}$$

Formally, the data-set consist of feature vector and class label pairs:

$$\{x_1, y_1\}, \{x_2, y_2\}, ..., \{x_n, y_n\} \tag{3.2}$$

The learning algorithm finds a model (g) that approximates the target function (f):

$$g \approx f \tag{3.3}$$

Then the classifier allows to predict class labels of new instances as:

$$y_{n+1} = g(x_{n+1}) \tag{3.4}$$

For the given problem, the target variable is an ordinal variable which has the values- 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0. Therefore, the prediction can be performed using binary as well as multiclass classification. We will discuss the design for these methods on-the-go as we explain the process.

Further, the data analysis is now being performed on the training data instead of the whole data-set as we delve more into data preparation and model selection tasks. We have split the data into training and test set with 7:3 ratio in a stratified fashion [47] using scikit-learn [43] library in Python. The stratified split makes sure that the proportion of samples with respect to class labels is maintained in both training and test set as it appears in the original data-set.

The proposed workflow for classification is illustrated in Figure 16. Since the number of samples to train the model will reduce drastically if the validation set is generated from the training data, the cross-validation [48] technique will be used for model evaluation. The tuning of hyperparameters is necessary to regularise the model in order to avoid over-fitting the data and make it generalise to unseen data. However, the tuning of hyperparameters for each question during model evaluation will be cumbersome. Therefore, we propose to build two essential components- *Estimator Finder* and *Predictor*.

The benefits of Estimator Finder component are two-fold. First, this will help in model evaluation and selecting the best model (predictor) for each question. Second, the component can be used for as many models and hyperparameters as we want and hence this speeds up the overall process considering our problem set with 50 target variables. The output of this component will be a text file which includes the best model along with hyperparameters for each question. On the other hand, the Predictor component will be used to do classification using default parameters as well as with the best model identified by the Estimator Finder. The reason for evaluating multiple models is that there is no predefined model that is guaranteed to generalise better for any problem. David [49] demonstrates this by *No Free Lunch* (NFL) theorem.

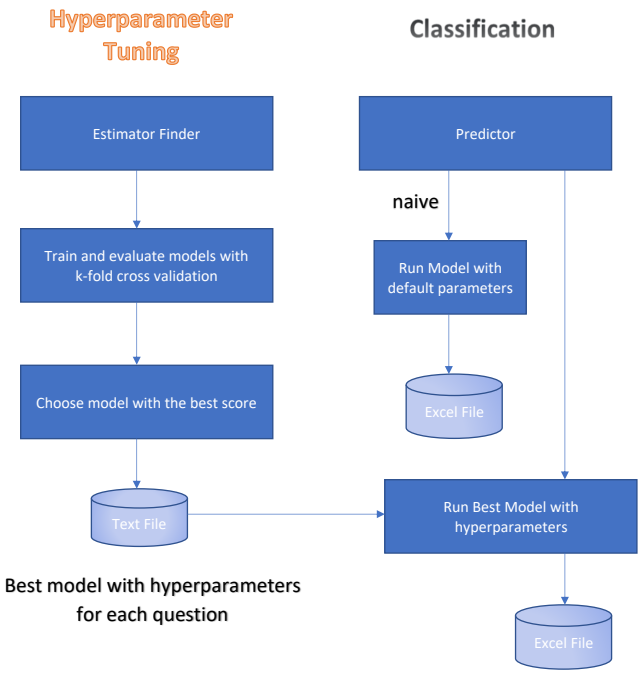

Figure 16: Proposed Workflow for Classification

3.3.1.1 Binary Classification

In binary classification, the label has to be binary with only two unique values. Therefore, the label for each question will be converted to binary (0/ 1). Two options are listed in Table 7 with sample proportion in each class of the training data-set (70% data). There are more than 50% questions with less than 10% samples in class 0 with Option(a). The overall sample proportion is better in Option(b) as compared to Option(a). Therefore, we will convert each label value between 1 and 6 (inclusive) to 0 (*Class 0*) and 7 to 1 (*Class 1*) in order to carry out binary classification.

Table 7: Training Sample Proportion (%) for Binary Classification

Question	Option(a)*		Option(b)*	
	Class 0	Class 1	Class 0	Class 1
Q01	30.88	69.12	77.68	22.32
Q02	1.23	98.77	59.69	40.31
Q03	18.41	81.59	69.2	30.8
Q04	71.43	28.57	90.31	9.69
Q05	3.79	96.21	66.61	33.39
Q06	7.3	92.7	67.65	32.35
Q07	26.39	73.61	76.82	23.18
Q08	53.14	46.86	85.64	14.36
Q09	19.63	80.37	75.09	24.91
Q10	25.42	74.58	75.26	24.74
Q11	40.65	59.35	82.18	17.82
Q12	70.76	29.24	91	9
Q13	46.78	53.22	83.39	16.61
Q14	50.21	49.79	84.08	15.92
Q15	57.21	42.79	86.51	13.49
Q16	1.62	98.38	60.03	39.97
Q17	5.58	94.42	70.24	29.76
Q18	2.93	97.07	63.67	36.33
Q19	5.02	94.98	63.32	36.68
Q20	2.76	97.24	66.26	33.74
Q21	6.76	93.24	69.72	30.28
Q22	11.57	88.43	73.36	26.64
Q23	2.75	97.25	66.44	33.56
Q24	12.5	87.5	69.72	30.28
Q25	3.36	96.64	64.01	35.99
Q26	21.3	78.7	81.14	18.86
Q27	5.44	94.56	68.34	31.66
Q28	2.48	97.52	62.63	37.37
Q29	5.09	94.91	69.38	30.62
Q30	41.53	58.47	79.58	20.42
Q31	13.69	86.31	75.78	24.22
Q32	6.3	93.7	69.55	30.45
Q33	2.25	97.75	65.92	34.08
Q34	10.64	89.36	74.22	25.78
Q35	11.02	88.98	74.05	25.95
Q36	7.55	92.45	72.15	27.85
Q37	3.78	96.22	68.51	31.49
Q38	3.4	96.6	64.36	35.64
Q39	20.5	79.5	80.1	19.9
Q40	10.5	89.5	74.57	25.43
Q41	5.88	94.12	69.2	30.8
Q42	3.69	96.31	70.76	29.24
Q43	8.68	91.32	75.09	24.91
Q44	5.15	94.85	65.22	34.78
Q45	4.62	95.38	73.36	26.64
Q46	36.11	63.89	81.14	18.86
Q47	6.25	93.75	65.74	34.26
Q48	3.81	96.19	65.05	34.95
Q49	9.66	90.34	72.15	27.85
Q50	77.02	22.98	92.91	7.09

*Option(a): labels (1-4) categorised to Class 0, labels (5-7) categorised to Class 1

*Option(b): labels (1-6) categorised to Class 0; label (7) categorised to Class 1

Table 8 shows the stratified split of 25 questions with Option(b). Clearly, the sample proportion is equivalent in train and test sets for each class label. Hence, we have representative samples from our original data-set which make sure our models will be trained on the accurate training set. The same approach will be followed throughout this project whenever necessary.

3.3.1.2 Multiclass Classification

In multiclass classification, we propose to perform classification with three classes as sample proportions are not found to be significant with more than three classes. All target variables with values between 1 and 3 (inclusive) will be converted to 0 (*Class 0*), 4 and 6 (inclusive) to 1 (*Class 1*) and the remaining value 7 to 2 (*Class 2*). The sample proportion with this conversion is listed in Table 9.

3.3.2 Unsupervised Learning

Unsupervised learning [3] is another machine learning technique which is used to identify any patterns in data when we do not know the ground truth. That means, labels are not available in this case. Although we have a label for each question, there is no separate label for the overall data-set. We propose to apply two clustering techniques to identify any natural groupings of user responses. First, we will apply PCA [24] which is a visualisation and dimensionality reduction technique. This can be helpful to visualise data in 2-D or 3-D space while trying to preserve as much data as possible. Second, we will run K-Means clustering algorithm [19] on data with silhouette coefficient to be used for evaluation. The main aim of this study will be to identify segments of user behaviours.

3.3.3 Probabilistic Reasoning

The main idea for this part is to draw some meaningful inferences from the survey responses. We want to identify any causal relationships among truthfulness of answers. We wonder if answers relate to each other in a sense that whether truthfulness of one answer has an influence to other answer in this context. The structural relationships will be deduced by learning Bayesian networks from data [50].

The Bayesian networks model probability distributions according to the below definition.

"Each variable is conditionally independent of all its non-descendants in the graph given the value of all its parents."

$$P(X_1, ..., X_n) = \sum_{i=1}^{n} P(X_i | Parents(X_i)) \tag{3.5}$$

One can also draw inferences using the overall structure given some evidence about data. For example: Given the evidence that an unknown person (not from the given

Table 8: Stratified Split (%) for Binary Classification

Question	Train (70%)		Test (30%)	
	Class 0	Class 1	Class 0	Class 1
Q01	77.68	22.32	77.51	22.49
Q02	59.69	40.31	59.44	40.56
Q03	69.2	30.8	69.48	30.52
Q04	90.31	9.69	90.36	9.64
Q05	66.61	33.39	66.67	33.33
Q06	67.65	32.35	67.47	32.53
Q07	76.82	23.18	76.71	23.29
Q08	85.64	14.36	85.54	14.46
Q09	75.09	24.91	75.1	24.9
Q10	75.26	24.74	75.5	24.5
Q11	82.18	17.82	81.93	18.07
Q12	91	9	91.16	8.84
Q13	83.39	16.61	83.13	16.87
Q14	84.08	15.92	84.34	15.66
Q15	86.51	13.49	86.75	13.25
Q16	60.03	39.97	60.24	39.76
Q17	70.24	29.76	70.28	29.72
Q18	63.67	36.33	63.45	36.55
Q19	63.32	36.68	63.05	36.95
Q20	66.26	33.74	66.27	33.73
Q21	69.72	30.28	69.48	30.52
Q22	73.36	26.64	73.49	26.51
Q23	66.44	33.56	66.67	33.33
Q24	69.72	30.28	69.48	30.52
Q25	64.01	35.99	64.26	35.74

Table 9: Train & Test Sample Proportion (%) for Multiclass Classification

Question	Train (70%)			Test (30%)		
	Class 0	Class 1	Class 2	Class 0	Class 1	Class 2
Q01	23.5	17.05	59.45	23.4	17.02	59.57
Q02	1.23	2.88	95.88	0.95	2.86	96.19
Q03	15.06	10.88	74.06	14.56	10.68	74.76
Q04	67.65	8.82	23.53	67.96	8.74	23.3
Q05	2.37	6.16	91.47	2.2	6.59	91.21
Q06	6.01	13.3	80.69	6	14	80
Q07	22.22	15.74	62.04	21.51	16.13	62.37
Q08	45.19	20.08	34.73	45.63	19.42	34.95
Q09	16.44	17.81	65.75	17.02	17.02	65.96
Q10	23.31	16.53	60.17	22.55	16.67	60.78
Q11	33.64	18.22	48.13	33.33	18.28	48.39
Q12	66.1	11.86	22.03	66.67	11.76	21.57
Q13	41.2	17.6	41.2	41	17	42
Q14	46.41	15.19	38.4	46.08	14.71	39.22
Q15	54.42	9.77	35.81	53.76	9.68	36.56
Q16	1.21	5.26	93.52	0.94	5.66	93.4
Q17	1.72	24.46	73.82	1.98	24.75	73.27
Q18	1.67	10.46	87.87	0.97	10.68	88.35
Q19	3.35	7.95	88.7	2.91	7.77	89.32
Q20	1.84	8.29	89.86	1.08	8.6	90.32
Q21	3.6	17.57	78.83	3.12	17.71	79.17
Q22	6.61	29.75	63.64	6.73	29.81	63.46
Q23	1.38	10.09	88.53	1.06	9.57	89.36
Q24	7.92	19.17	72.92	7.69	19.23	73.08
Q25	1.68	11.34	86.97	0.97	11.65	87.38
Q26	14.35	35.19	50.46	15.05	34.41	50.54
Q27	2.93	20.5	76.57	2.91	20.39	76.7
Q28	0.41	10.33	89.26	0.95	10.48	88.57
Q29	2.78	15.28	81.94	3.23	15.05	81.72
Q30	33.05	16.95	50	33.33	16.67	50
Q31	8.3	33.61	58.09	7.69	33.65	58.65
Q32	4.62	21.43	73.95	3.92	21.57	74.51
Q33	1.35	9.91	88.74	2.08	9.38	88.54
Q34	4.26	32.34	63.4	4.9	32.35	62.75
Q35	4.66	31.78	63.56	4.9	31.37	63.73
Q36	4.72	19.34	75.94	4.4	19.78	75.82
Q37	3.36	20.17	76.47	3.88	19.42	76.7
Q38	1.7	10.64	87.66	1.96	10.78	87.25
Q39	10.88	41	48.12	11.65	40.78	47.57
Q40	4.2	34.03	61.76	3.92	34.31	61.76
Q41	1.68	23.95	74.37	1.94	23.3	74.76
Q42	0.92	21.2	77.88	1.06	21.28	77.66
Q43	5.02	29.22	65.75	4.26	29.79	65.96
Q44	2.58	11.16	86.27	1.98	11.88	86.14
Q45	1.26	34.45	64.29	0.97	33.98	65.05
Q46	30.56	18.98	50.46	31.18	18.28	50.54
Q47	3.33	14.58	82.08	2.88	14.42	82.69
Q48	1.27	13.14	85.59	1.96	12.75	85.29
Q49	5.04	26.89	68.07	5.88	26.47	67.65
Q50	73.62	8.94	17.45	73.53	8.82	17.65

Class 0 includes labels between 1 and 3 (inclusive); Class 1 includes labels between 4 and 6 (inclusive); Class 2 includes the remaining label 7

data-set) answered truthfully in some questions, what is the likelihood that he/ she would be truthful in other questions?

3.4 Data Preparation

As we have performed the data analysis and identified the key components for our proposed system, it is a good time to discuss the data preparation tasks before moving on to methodology and implementation.

3.4.1 Data Cleaning

In this section, we will discuss measures to clean the data and fill-in any missing values. The *Prolific ID* variable will be dropped as it is not required for classification and/ or any other tasks. Any additional space characters from the variable names will be truncated and the columns in Pandas [51] dataframe will be ordered in alphabetical order. The outliers are identified in our data-set using box-plot. Figure 17 shows that we have many outliers in demographics data and privacy factors except *Gender* feature. Similarly, a significant number of outliers are observed in feedback answers for Q01-Q09 as in Figure 18. We suspect that the variables with high number of outliers will not be helpful during classification as some classifiers are sensitive to outliers for prediction. However, we might lost important information by eliminating all such data with outliers. The best way to deal with outliers is to measure the classification performance with and without outliers and then make a decision.

The imputation of missing data [52] [53] will be performed using mean of the particular variable in the data-set. There are many sophisticated methods defined in existing literature but the mean should be reasonable for the problem under discussion.

3.4.2 Feature Engineering

For feature engineering, we will discretise the continuous features. The conversion of *Age* and *Online Presence* features is outlined in Table 10 and 11 respectively. One more efficient way to do this would be to first divide the variable into bins using quantile through which each bin will have the same fraction of samples and then apply one-hot encoding to produce binary feature vectors. Each feature vector will have 1 for the the bin under consideration and 0 for all others. For example: The one-hot feature vectors for the *Age* variable will be [1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0] and so on corresponding to the bins. This makes sure that the model learns different weights for each feature vector. One-hot encoding in our case will not be necessary as each sample will belong to only one bin at a time. As all other variables are ordinal, further feature engineering will not be necessary.

3.4.3 Feature Scaling

During data analysis, it has been observed that the data is not normalised in our case. Therefore, the features will have to be standardised by subtracting the mean and scaling

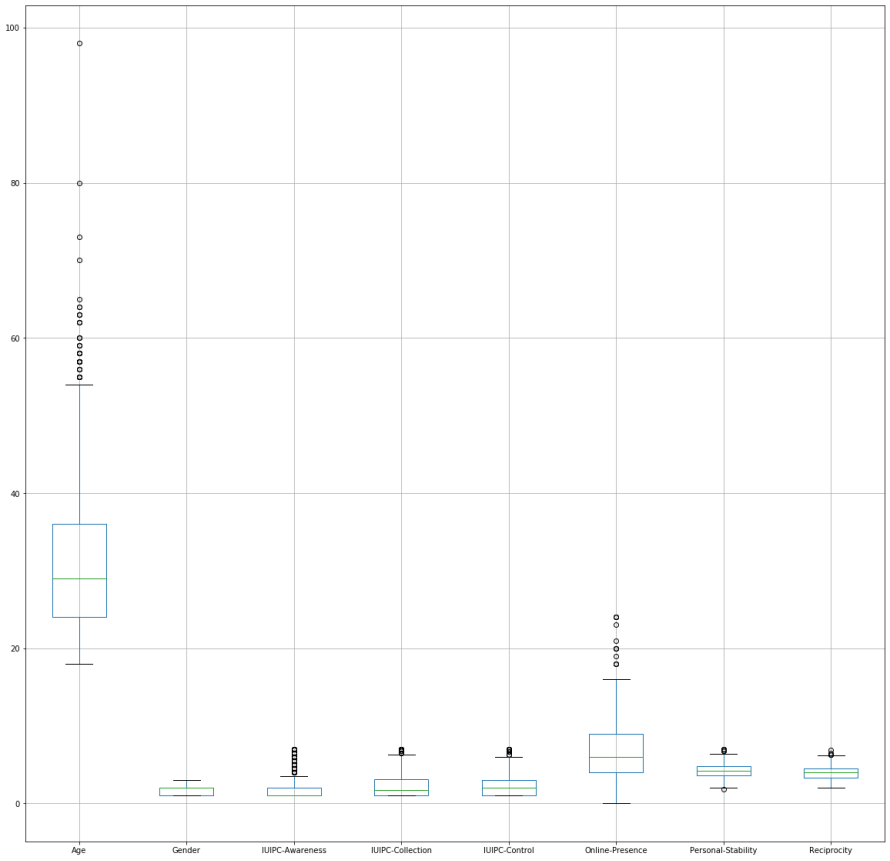

Figure 17: Boxplot of Demographics and IUIPC Factors

Table 10: Discretisation of *Age* Feature

Bin	**Value**
<=17	0
18-24	1
25-34	2
35-44	3
45-54	4
55-64	5
>65	6

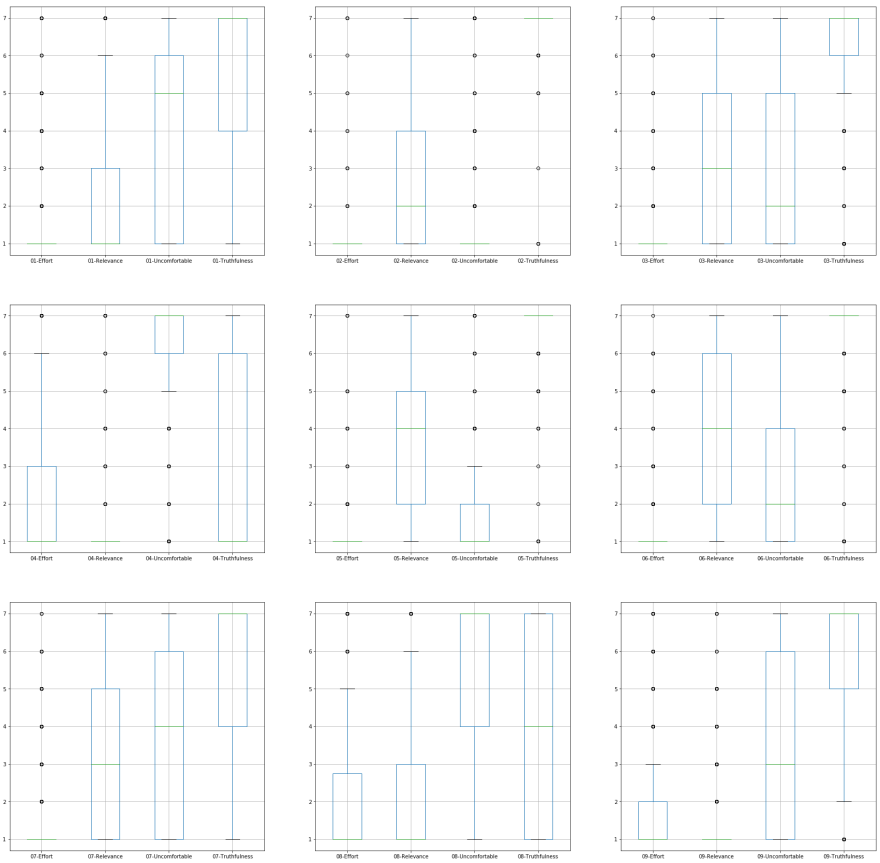

Figure 18: Boxplot of Feedback Q01-Q09

Table 11: Discretisation of *Online Presence* Feature

Bin	**Value**
<=5	0
6-10	1
11-15	2
16-20	3
21-25	4

to unit variance. The performance of machine learning algorithms might be worse if the individual features do not more or less represent the normal distribution. The features with large variance in this case will dominate and subsequently the model will not be able to learn from features with less variance. The data for this project will be normalised using scikit-learn's [43] pre-processing package.

4 Methodology and Implementation

Currently, the next step in our ML pipeline (Figure 4) is to choose a model while following the proposed workflow (Figure 16) for classification. Please note that the output of the Estimator Finder component (which is a text file including model along with hyperparameters for each question) is copied manually and being used as-is in Predictor component in our current implementation.

4.1 Supervised Learning

We have defined and implemented our methodologies for supervised learning in three phases. The reason to perform implementation in phases was to track the progress and maintain separate code versions. This also helped us to compare and contrast the results obtained during each phase. Further, we decided to implement our methodologies for binary classification first so that the lessons learnt during the process can be applied further during multiclass classsification and/ or remaining tasks. In this paper, we call "naive classification" as the one which is carried out with default parameters. The results for each phase will be discussed in detail in Section 5.

4.1.1 Phase 1 - Unweighted Learning

Since there were multiple subsets corresponding to 50 questions, we selected a mix of basic and ensemble classifiers for training and evaluation. We trained these classifiers with default parameters first and measured their performance using cross validation. Then, the estimator finder component was used to tune the hyperparameters of every classifier. The scikit-learn's model selection class *GridSearchCV* was used for this purpose. It is to be noted that the time complexity of training and evaluation depends on the number of combinations to try for estimators and hyperparameters. One can use *RandomizedSearchCV* class to trade accuracy for speed. Finally, the best classifier was trained and used for predicting the truthfulness measure of each question.

Initial results with default parameters motivated us to tune hyperparameters. The naive classification scores obtained through cross-validation for the first five questions are listed in Table 12.

We will not include the "naive classification" scores for any phase here in this paper for brevity. However, the program output will be the EXCEL file which will have results for all the questions. Figure 19 shows the classifiers along with their hyperparameters that were fed into Estimator Finder component to get the best classifier for each question using 10-fold cross validation.

Table 12: Unweighted Naive Binary Classification Scores Q01-Q05 Version 1.1

(*Samples with All Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	Decision Tree	88.23	82.76	74.02	72.87	92.65	0.73	1
	Logistic Regression	90.85	84.71	83.33	73.64	95.77	0.78	0.9
	K-Nearest Neighbors	88.23	79.72	79.05	64.34	95.1	0.71	0.98
	Random Forest	89.82	84.04	79.17	73.64	94.43	0.76	1
	Extra Trees	90.15	84.54	80	74.42	94.65	0.77	1
	SVM	90.67	85.98	80	77.52	94.43	0.79	0.98
	Naive Bayes	87.4	85.79	67.72	82.95	88.64	0.75	0.93
	Bagging	90.5	85.59	79.84	76.74	94.43	0.78	1
	ADABoost	90.5	86.97	77.61	80.62	93.32	0.79	0.98
	XGBoost	91.53	88.19	80.3	82.17	94.21	0.81	1
Q02	Decision Tree	97.4	97.13	97.81	95.71	98.55	0.97	1
	Logistic Regression	98.08	97.92	98.26	97	98.84	0.98	0.97
	K-Nearest Neighbors	96.01	95.34	98.17	91.85	98.84	0.95	1
	Random Forest	98.44	98.42	97.86	98.28	98.55	0.98	1
	Extra Trees	98.78	98.78	98.29	98.71	98.84	0.99	1
	SVM	98.25	98.13	98.27	97.42	98.84	0.98	1
	Naive Bayes	98.61	98.84	96.68	100	97.68	0.98	0.99
	Bagging	97.91	97.84	97.42	97.42	98.26	0.97	1
	ADABoost	98.61	98.63	97.87	98.71	98.55	0.98	1
	XGBoost	98.6	98.56	98.28	98.28	98.84	0.98	1
Q03	Decision Tree	87.04	84.08	80.47	76.4	91.75	0.78	1
	Logistic Regression	91.52	90.76	84.49	88.76	92.75	0.87	0.94
	K-Nearest Neighbors	89.46	86.92	84.62	80.34	93.5	0.82	0.98
	Random Forest	90.14	89.13	82.35	86.52	91.75	0.84	1
	Extra Trees	89.11	87.45	81.77	83.15	91.75	0.82	1
	SVM	89.79	89.66	79.9	89.33	90	0.84	0.99
	Naive Bayes	91.18	92.22	80.09	94.94	89.5	0.87	0.94
	Bagging	89.62	88.13	82.42	84.27	92	0.83	1
	ADABoost	91.35	90.79	83.68	89.33	92.25	0.86	0.99
	XGBoost	90.13	89.45	81.68	87.64	91.25	0.85	1
Q04	Decision Tree	86	60.36	28.07	28.57	92.15	0.28	1
	Logistic Regression	89.98	51.4	33.33	3.57	99.23	0.06	0.73
	K-Nearest Neighbors	89.45	51.12	22.22	3.57	98.66	0.06	0.92
	Random Forest	90.31	58.77	50	19.64	97.89	0.28	1
	Extra Trees	90.48	56.47	53.33	14.29	98.66	0.23	1
	SVM	90.14	49.9	0	0	99.81	0	0.99
	Naive Bayes	90.49	62.85	51.61	28.57	97.13	0.37	0.91
	Bagging	89.45	59.88	41.94	23.21	96.55	0.3	1
	ADABoost	89.97	61.76	46.88	26.79	96.74	0.34	0.98
	XGBoost	91.36	61.73	63.64	25	98.47	0.36	0.99
Q05	Decision Tree	95.15	94.17	94.12	91.19	97.14	0.93	1
	Logistic Regression	96.02	95.33	94.74	93.26	97.4	0.94	0.94
	K-Nearest Neighbors	94.1	92.23	95.43	86.53	97.92	0.91	1
	Random Forest	96.71	96.63	93.94	96.37	96.88	0.95	1
	Extra Trees	96.88	96.76	94.42	96.37	97.14	0.95	1
	SVM	96.71	96.37	94.85	95.34	97.4	0.95	1
	Naive Bayes	96.71	97.53	91.04	100	95.06	0.95	0.99
	Bagging	96.36	96.11	93.88	95.34	96.88	0.95	1
	ADABoost	95.83	95.59	92.89	94.82	96.36	0.94	1
	XGBoost	96.88	96.89	93.97	96.89	96.88	0.95	1

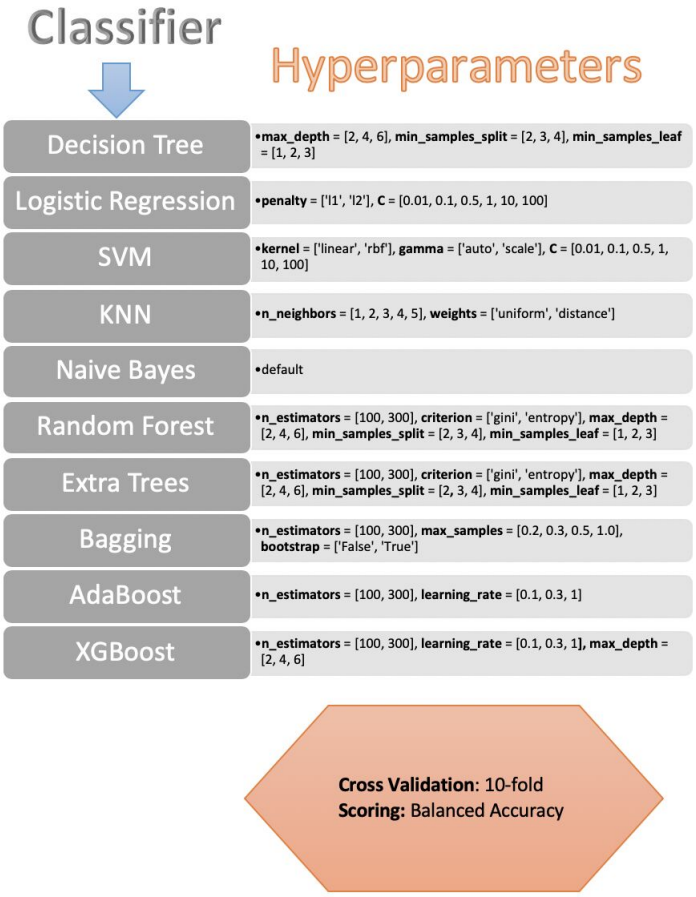

Figure 19: Phase 1 - Hyperparameter Tuning for Binary Classification

Table 13: Confusion Matrix for Binary Classification

		Actual Class
Predicted Class	True Positive (TP)	False Positive (FP)
	False Negative (FN)	True Negative (TN)

The scoring metric used initially during cross validation was *Accuracy*. Subsequently, the best classifier for each question was identified according to the best accuracy score. Then this classifier was trained and evaluated against the test set. The performance measures for prediction against the test data were: *Accuracy*, *Precision*, *Recall*, *Specificity*, *F1* and *ROC AUC*.

In order to understand these metrics, a general confusion matrix is shown in Table 13 which depicts the measure of successful predictions by any binary classification model. *True Positive* is an example for which the model correctly predicted the positive class. Similarly, *True Negative* is an example for which the model correctly predicted the negative class. *False Positive* will be the case in which the model incorrectly predicted the positive class for an example that actually belong to the negative class. Similarly, an example from actual positive class will fall into *False Negative* if the model incorrectly predicted the negative class for the same.

Since we are now aware of the TP, FP, TN and FN cases, it is easy to understand what all other performance metrics mean. *Accuracy* is defined as the correct number of predictions out of total number of examples. For binary classification, accuracy is defined as follows in equation 4.1.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

$$Recall\ (TPR) = \frac{TP}{TP + FN} \tag{4.3}$$

$$Specificity\ (TNR) = \frac{TN}{TN + FP} \tag{4.4}$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{FP + TN} \tag{4.5}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.6}$$

Precision determines how accurate the model predictions were for the positive class. For instance: If the model predicts 90 true positives and 10 false positives, then the precision will be 0.9 for the positive class. *Recall* is used to determine the proportion of

correctly identified positive examples out of all the possible positive labels. This measure is also called the *True Positive Rate*. In essence, precision determines the exactness and recall measures the completeness for the positive class. On the other hand, *Specificity*, which is also called the *True Negative Rate*, tells the proportion of correctly identified negative examples out of all the possible negative labels.

Furthermore, *F1* score is the harmonic mean of precision and recall. This is preferred when you want to look at one metric instead of precision and recall together to compare the performance of multiple classifiers. Finally, the *Receiver Operating Characteristic* (ROC) curve plots the true positive rate against the false positive rate. A good classifier in this case will be the one for which the plot is towards the top-left corner (with true positive rate on y-axis close to 1 and false positive rate on x-axis close to 0). The classifiers can be compared using *area under the curve* (AUC) metric. A purely random classifier will have a *RUC AUC* score equal to 0.5 whereas a perfect one will have this score equal to 1.

For our problem, there was no performance improvement for many questions after trying several combinations of hyperparameters and scoring metrics. Some of the scoring metrics did not work during cross-validation as the number of samples were not significant for classification.

We found two potential issues at this stage. First, the proportion of samples was not balanced for each class as in Table 7. Remember, Option(b) in this figure was our choice for this project. The existing study reveals that most of the classifiers are biased towards the majority class samples. Therefore, they prefer to predict more labels from the majority class. Subsequently, the desired performance was not achieved on the minority class samples. However, the overall accuracy was still high as most of the labels were predicted correctly from the majority class. Therefore the accuracy was not found to be a good performance measure for imbalanced data.

Second, we did not apply the same scaler/ transformer on test data that we used on training set during feature scaling. That means, the test data was not transformed in the same way as the training set.

The current phase is named as *Unweighted Learning* for our own purpose. We fixed the second issue within this phase along with changing the cross validation scoring metric to *Balanced Accuracy* instead of *Accuracy*. We will discuss this metric in the next section. Also, we worked out the class-imbalance problem using sampling techniques in second phase using a different set of classifiers for training and evaluation. Accordingly, Phase 2 will be about *Imbalanced Learning* through sampling techniques.

The key points from this phase are:

- The class-imbalance leads to poor performance for minority class samples.
- Accuracy is not the correct scoring metric for imbalanced data.
- Data cleaning and preparation tasks should be performed on training set first and then on test set before prediction. Feature transformation should be performed on the test data with the same scaler/ transformer object used on the training set.

4.1.2 Phase 2 - Imbalanced Learning

We call this phase to deal with class imbalance problem as *Imbalanced Learning*. A systematic study of class-imbalance has been carried out in [54]. Common examples with this problem are: Fraud detection, Spam filter, Medical diagnosis etc. There are several methods to deal with the imbalanced data- Sampling, Cost-Sensitive, Kernel-Based and Active Learning as in [55]. Below are the commonly used metrics to deal with imbalanced data.

$$Balanced\ Accuracy = \frac{Sensitivity + Specificity}{2} = \frac{\frac{TP}{TP+FN} + \frac{TN}{TN+FP}}{2} \tag{4.7}$$

$$G\text{-}Mean = \sqrt{Sensitivity \times Specificity} \tag{4.8}$$

The balanced accuracy metric (where sensitivity is the true positive rate and specificity is the true negative rate) was used for cross validation and compare the performance between different methods in this project. The above definition is valid for binary classification. This will be the average of recall obtained on each class for multiclass classification [43]. The additional metrics were also included in the output to investigate how they fluctuate with different methods.

4.1.2.1 Over-Sampling

The first method we utilised for solving the class imbalance problem was the Synthetic Minority Oversampling Technique (SMOTE) [56] which generates new synthetic samples by interpolation using K-Nearest Neighbour rule. That means the minority class was over-sampled by adding new samples to make it balanced with the existing majority class. We kept the same classifiers with their hyperparameters while applying this technique. The only change we made from the previous phase was that we used oversampling in this phase to balance the data in both classes before training and evaluation. However, we did not achieve significant performance improvement using this method when we compared the *balanced accuracy* scores with previous phase. We decided to move further and explore other methods.

However, there was one important observation made during investigation while going through the process again. That is not all features were useful for the classifier. If we look at our correlation heat map again in Figure 15, we missed identifying any variables which have had no linear relationship with the target variable. For example, many variables have correlation coefficient close to zero with*truthfulness* feature. We reflected back on this when we trained Random Forest Classifier on our training set and plotted the features according to their importance. Table 14 lists the *informative* features in decreasing order. One can also make use of the Weka software [57] to extract the discriminative features.

Only the first three features were found to be informative for majority of the ques-

Table 14: Informative Features in Decreasing Order (L-R)

Question	Feature 1	Feature 2	Feature 3
Q01	01-Effort	01-Uncomfortable	01-Relevance
Q02	02-Effort	02-Uncomfortable	02-Relevance
Q03	03-Effort	03-Uncomfortable	03-Relevance
Q04	04-Uncomfortable	Personal-Stability	04-Relevance
Q05	05-Effort	05-Uncomfortable	05-Relevance
Q06	06-Effort	06-Uncomfortable	06-Relevance
Q07	07-Effort	07-Uncomfortable	07-Relevance
Q08	08-Uncomfortable	08-Effort	Reciprocity
Q09	09-Effort	09-Uncomfortable	09-Relevance
Q10	10-Uncomfortable	10-Effort	10-Relevance
Q11	11-Uncomfortable	11-Effort	11-Relevance
Q12	12-Uncomfortable	Personal-Stability	12-Effort
Q13	13-Effort	13-Uncomfortable	13-Relevance
Q14	14-Effort	14-Uncomfortable	14-Relevance
Q15	15-Uncomfortable	15-Effort	15-Relevance
Q16	16-Effort	16-Relevance	16-Uncomfortable
Q17	17-Effort	17-Uncomfortable	17-Relevance
Q18	18-Effort	18-Uncomfortable	18-Relevance
Q19	19-Effort	19-Uncomfortable	19-Relevance
Q20	20-Effort	20-Uncomfortable	20-Relevance
Q21	21-Relevance	21-Uncomfortable	21-Effort
Q22	22-Effort	22-Uncomfortable	22-Relevance
Q23	23-Effort	23-Uncomfortable	23-Relevance
Q24	24-Effort	24-Relevance	24-Uncomfortable
Q25	25-Effort	25-Uncomfortable	25-Relevance
Q26	26-Effort	26-Uncomfortable	26-Relevance
Q27	27-Uncomfortable	27-Effort	27-Relevance
Q28	28-Effort	28-Relevance	28-Uncomfortable
Q29	29-Effort	29-Uncomfortable	29-Relevance
Q30	30-Effort	30-Uncomfortable	30-Relevance
Q31	31-Uncomfortable	31-Effort	31-Relevance
Q32	32-Uncomfortable	32-Effort	32-Relevance
Q33	33-Uncomfortable	33-Effort	33-Relevance
Q34	34-Uncomfortable	34-Effort	34-Relevance
Q35	35-Uncomfortable	35-Effort	35-Relevance
Q36	36-Uncomfortable	36-Effort	36-Relevance
Q37	37-Uncomfortable	37-Effort	37-Relevance
Q38	38-Effort	38-Uncomfortable	38-Relevance
Q39	39-Uncomfortable	39-Effort	39-Relevance
Q40	40-Uncomfortable	40-Effort	40-Relevance
Q41	41-Uncomfortable	41-Effort	41-Relevance
Q42	42-Uncomfortable	42-Effort	42-Relevance
Q43	43-Uncomfortable	43-Effort	43-Relevance
Q44	44-Uncomfortable	44-Effort	44-Relevance
Q45	45-Uncomfortable	45-Effort	45-Relevance
Q46	46-Uncomfortable	46-Effort	46-Relevance
Q47	47-Effort	47-Uncomfortable	47-Relevance
Q48	48-Effort	48-Uncomfortable	48-Relevance
Q49	49-Effort	49-Uncomfortable	49-Relevance
Q50	50-Uncomfortable	Age	Reciprocity

tions. Accordingly, we decided to use these features for our research. Since we were evaluating multiple models in each phase, we maintained two separate versions of the Estimator Finder and Predictor Components during our implementation.

- Version 1.1 - Training and Evaluation using all features
- Version 1.2 - Training and Evaluation using informative features

The reason to do so was to compare the performance using both versions rather than discarding the first version altogether. The first version could also be utilised as a fallback option in case of unexpected circumstances.

4.1.2.2 Under-Sampling

The second method that we used to deal with imbalanced data was random under-sampling [58]. There are many ways to do this using ensemble samplers with bagging, boosting and other methods as reviewed in [59]. We opted for Balanced Random Forest [60] and Easy Ensemble [59] classifiers for training and evaluation in this phase. Figure 20 shows the hyperparameters used to find the best classifier for each question using the Estimator Finder component. The only additional hyperparameter here is *sampling_strategy* which specifies the class to be used for re-sampling.

Balanced Random Forest is an ensemble of trees whereas Easy Ensemble is an ensemble of AdaBoost learners. The main idea is that the models are being trained on balanced bootstrap samples in both cases. The same set of models and hyperparameters were used for multiclass classification on the training data with sample proportion as outlined in Table 9. The micro-average was taken into account for precision, recall and f1 scores during multiclass classification. Micro-average is computed globally by counting the true positives, false negatives and false positives [43]. Further, we opted for *5-fold* instead of *10-fold* cross validation because ensemble classifiers have to go through several number of combinations during hyperparamater tuning and consequently there will be a performance hit.

4.1.3 Phase 3 - Cost-Sensitive Learning

Cost-Sensitive learning [61] [62] is about assigning weights to each class in order to define higher misclassification cost for the minority class. Basically there will be more penalty for learner to misclassify the minority class samples. For example: Class 0 has 100 samples and Class 1 has 10 samples. If we define the weight for Class 1 to be 10 and 1 for Class 0, then this will make classifier to be penalised 10 times more for misclassifying Class 1 samples. These weights should be adjusted as per our needs. Figure 21 shows the classifiers used for training and evaluation in this phase. The *class_weight* hyperparameter was included for tuning with each classifier. Not every classifier has this hyperparameter.

Lastly, we did not remove the outliers in data as we were not left with enough data after dropping them out and hence classification was not possible in such cases.

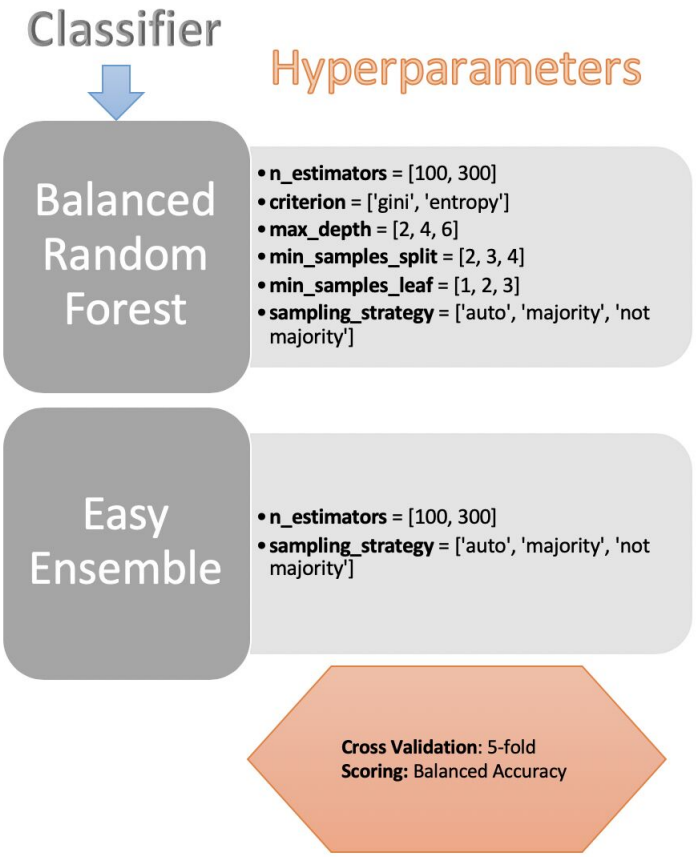

Figure 20: Phase 2 - Hyperparameter Tuning for Binary and Multiclass Classification with Under-Sampling

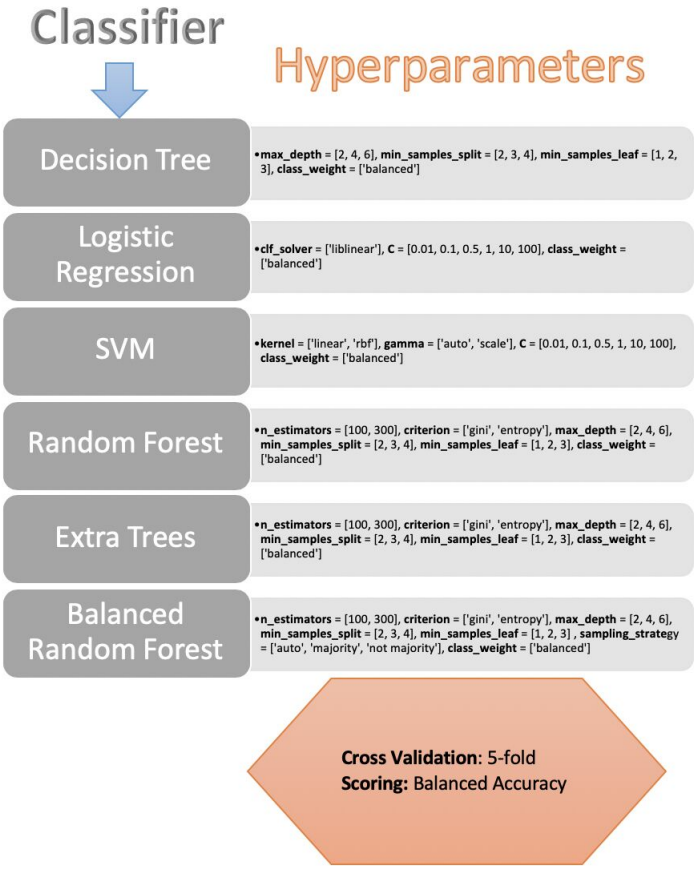

Figure 21: Phase 3 - Hyperparameter Tuning for Binary Classification with Cost-Sensitive Learning

Table 15: PCA Cumulative Explained Variance

Question	Dimension 1	Dimension 2	Dimension 3	Dimension 4	Dimension 5	Dimension 6	Dimension 7	Dimension 8	Dimension 9	Dimension 10	Dimension 11
Q01	0.3005	0.4649	0.5984	0.6961	0.7804	0.8315	0.8726	0.9102	0.9403	0.9662	0.987
Q02	0.2418	0.4327	0.5679	0.6647	0.7406	0.7989	0.8508	0.8967	0.9337	0.9589	0.9818
Q03	0.258	0.4198	0.5675	0.6676	0.7526	0.8116	0.855	0.8976	0.9305	0.9587	0.9841
Q04	0.2647	0.4555	0.5749	0.6763	0.7595	0.8194	0.8624	0.9005	0.9306	0.9602	0.9866
Q05	0.2617	0.4832	0.5963	0.6857	0.7481	0.7995	0.8423	0.8836	0.9219	0.9546	0.982
Q06	0.2214	0.4051	0.5759	0.6868	0.7499	0.8002	0.849	0.8882	0.9248	0.9567	0.9848
Q07	0.2801	0.4538	0.5956	0.6908	0.7813	0.8307	0.8718	0.9033	0.934	0.9615	0.9851
Q08	0.2721	0.4424	0.5689	0.6724	0.7669	0.8406	0.8814	0.9118	0.9407	0.9646	0.9846
Q09	0.2896	0.4456	0.5612	0.6671	0.734	0.7962	0.8458	0.8858	0.9224	0.9553	0.985
Q10	0.2876	0.4713	0.5718	0.6695	0.7559	0.828	0.8699	0.9025	0.9343	0.9624	0.9875
Q11	0.3027	0.4859	0.5974	0.6802	0.7612	0.823	0.8703	0.906	0.9352	0.9623	0.984
Q12	0.2673	0.4368	0.5952	0.7029	0.7908	0.8423	0.8793	0.9109	0.9394	0.9628	0.9837
Q13	0.2689	0.4399	0.5884	0.7007	0.7839	0.8469	0.8904	0.9214	0.9483	0.97	0.9874
Q14	0.3453	0.498	0.6242	0.7099	0.7845	0.8372	0.8793	0.9141	0.9416	0.9644	0.9847
Q15	0.2906	0.4493	0.5741	0.6686	0.7582	0.8336	0.8767	0.9086	0.9395	0.9649	0.9865
Q16	0.2982	0.4426	0.5668	0.6673	0.7367	0.7948	0.8501	0.8925	0.9308	0.9634	0.9857
Q17	0.2365	0.427	0.5581	0.6344	0.7054	0.7666	0.8229	0.8711	0.9154	0.9488	0.9791
Q18	0.2729	0.4386	0.5785	0.6774	0.7478	0.7998	0.8504	0.8876	0.9224	0.9544	0.9799
Q19	0.2462	0.4473	0.569	0.6531	0.7239	0.7793	0.8286	0.8711	0.9096	0.9458	0.9811
Q20	0.2843	0.4439	0.5927	0.6818	0.7442	0.7955	0.8415	0.8818	0.9188	0.9516	0.9804
Q21	0.2414	0.4575	0.5779	0.6925	0.7625	0.815	0.8578	0.8945	0.9265	0.957	0.9832
Q22	0.262	0.4567	0.5742	0.6727	0.7384	0.7943	0.8427	0.8856	0.9269	0.9558	0.9796
Q23	0.2384	0.4733	0.6076	0.6863	0.7555	0.808	0.8577	0.8978	0.9309	0.9626	0.9817
Q24	0.2375	0.4461	0.5578	0.6486	0.7279	0.7864	0.8343	0.8771	0.9141	0.9501	0.979
Q25	0.2827	0.4292	0.555	0.6508	0.7195	0.7793	0.8326	0.8787	0.9215	0.9554	0.9801
Q26	0.2367	0.4033	0.5462	0.6369	0.7197	0.7919	0.8474	0.8991	0.9321	0.9624	0.9865
Q27	0.292	0.4464	0.5645	0.6783	0.7353	0.7879	0.8344	0.8759	0.9153	0.948	0.9792
Q28	0.2785	0.4397	0.5725	0.652	0.7184	0.7786	0.8338	0.8831	0.9209	0.9537	0.9787
Q29	0.2581	0.4476	0.5666	0.6646	0.7315	0.7905	0.8331	0.8748	0.9114	0.9469	0.9807
Q30	0.33	0.5176	0.646	0.7365	0.8111	0.8573	0.8928	0.9209	0.9453	0.9679	0.984
Q31	0.2226	0.4117	0.5592	0.6609	0.7437	0.8158	0.8644	0.9003	0.9325	0.9604	0.9849
Q32	0.2789	0.4182	0.5514	0.6377	0.7091	0.7733	0.8275	0.8743	0.9138	0.9524	0.9799
Q33	0.2403	0.4568	0.5823	0.6967	0.7668	0.8165	0.8633	0.9018	0.9348	0.9588	0.9812
Q34	0.1999	0.3866	0.5573	0.6594	0.7339	0.7898	0.8421	0.8816	0.9195	0.9519	0.981
Q35	0.2062	0.3987	0.5576	0.6581	0.7286	0.7921	0.8444	0.8907	0.9282	0.9568	0.9851
Q36	0.2395	0.3979	0.5171	0.6223	0.7026	0.7741	0.8298	0.8814	0.9174	0.9516	0.9827
Q37	0.2337	0.4081	0.5667	0.6741	0.7484	0.805	0.8509	0.8904	0.9249	0.9555	0.9832
Q38	0.2492	0.4399	0.5755	0.6819	0.7483	0.7994	0.8455	0.8871	0.9272	0.9575	0.9809
Q39	0.2079	0.3832	0.5141	0.6199	0.7166	0.8062	0.8577	0.8985	0.9305	0.9594	0.9837
Q40	0.2213	0.4154	0.5754	0.6872	0.7566	0.8066	0.8543	0.8959	0.9314	0.9603	0.9861
Q41	0.2121	0.3973	0.565	0.6733	0.7396	0.7969	0.8434	0.887	0.9233	0.9547	0.9821
Q42	0.2382	0.4129	0.5565	0.6705	0.7391	0.798	0.8424	0.8831	0.9229	0.9552	0.9808
Q43	0.2142	0.4147	0.5306	0.6284	0.7179	0.7797	0.8338	0.878	0.9196	0.9542	0.9824
Q44	0.232	0.4015	0.5458	0.6481	0.7164	0.7754	0.8279	0.8728	0.9166	0.9507	0.9787
Q45	0.248	0.4242	0.5813	0.6821	0.7416	0.7984	0.8449	0.8861	0.9243	0.9552	0.9827
Q46	0.3051	0.5031	0.6139	0.7005	0.7806	0.8408	0.8809	0.9173	0.9461	0.9664	0.9847
Q47	0.2086	0.4105	0.5786	0.6925	0.7529	0.8103	0.8551	0.893	0.9294	0.9577	0.9829
Q48	0.264	0.4469	0.5747	0.6673	0.7378	0.794	0.8447	0.8883	0.9286	0.957	0.9795
Q49	0.2309	0.4508	0.58	0.6781	0.7433	0.7973	0.8484	0.8947	0.9297	0.9609	0.9844
Q50	0.232	0.423	0.5574	0.6669	0.7482	0.8256	0.8669	0.9065	0.9394	0.9662	0.9869

4.2 Unsupervised Learning

As per our design in Section 3.3.2, we applied K-Means algorithm on the overall data-set. However, the silhouette score was found to be zero which implied that the data was not structured at all. There was no success with Gaussian Mixture Models (GMM) as well. Therefore, we decided to run PCA algorithm to reduce the data to fewer dimensions. Table 15 shows the cumulative *explained variance* for each question excluding the target label.

Clearly, 80% of the data is preserved starting from seven dimensions. Less than 50% is preserved in two dimensions. Therefore, it was not meaningful to apply dimensionality reduction and find any patterns. After these findings, we decided to run K-Means algorithm for each question separately. However, the silhouette score was found to be between 0.1 and 0.3 which meant that the clusters would not be well formed out of this

data. There was no success after several attempts with different algorithms. Since our motivation to apply unsupervised learning methods was to identify any natural groupings of user responses, we decided to exploit the class labels. The idea was to first select informative features relevant to class labels in the same way as we discussed in Section 4.1.2.1 and then apply K-Means clustering algorithm. The difference is that the Random Forest classifier was applied to extract informative features on the overall data-set instead of training set and labels were not categorised to binary (0/ 1). Subsequently, we ran the K-Means algorithm on the data-set including labels for each question so that we could find any clusters/ segments along with the truthfulness measure (*1-7*). Classification algorithms perform much better than clustering algorithms for prediction if we were to predict the truthfulness measure. Below are the key points behind this methodology:

- Separate the feature data and labels.
- Run Random Forest Classifier to extract informative features relevant to class label (truthfulness feature) for each question.
- Concatenate the label back to feature data or use the original data-set for next steps.
- Scale the data so that each variable value is between 0 and 1.
- Apply K-Means algorithm and evaluate using silhouette score.
- Convert the cluster centres back to original form to interpret the results.

Interestingly, the score was found to be between 0.6 and 0.8 covering all the questions. The optimal number of clusters for each question was found to be 2 using elbow method [63]. The results of this approach will be discussed further in Section 5.2.

4.3 Probabilistic Reasoning

The main idea here was to understand any causal relationships in users' truthfulness across questions. We decided to meet this objective using Bayesian networks [64]. The open source library implementation in Python is hard to find in this area. We opted for pomegranate [65] library package which is the most sophisticated package currently available in Python. After learning the Bayesian network for each question, we moved on to generate network for the whole data-set. However, the search space of Directed Acyclic Graphs (DAG) is super-exponential to the order of number of variables. Therefore, we were not able to determine the causal relationships *optimally* for 50 questions altogether. Consequently, the Bayesian networks were learned for three question blocks (as outlined in Table 1, 2 and 3) separately. Moreover, Chow-Liu algorithm [66] [65] was found to be the fast approximation to learn the optimal tree-like structure for the Bayesian network. This algorithm does not support data with missing values. Therefore, we imputed the missing values using mean before learning the structure. We will discuss more about this method in Section 5.3.

Table 16: Python Library Reference

Task	**Library Package (w/ Version)**
Data Analysis	Matplotlib (3.1.0), NumPy (1.16.4), Pandas (0.24.2), Seaborn (0.9.0)
Data Preparation	NumPy (1.16.4), Pandas (1.16.4)
Supervised Learning	Imbalanced-learn (0.5.0), Scikit-learn (0.21.2), XGBoost (0.82)
Unsupervised Learning	Scikit-learn (0.21.2)
Probabilistic Reasoning	Pomegranate (0.10.0)

Table 17: System Information

Configuration	**Machine 1**	**Machine 2**
System	MacBook Pro 2018	CentOS Linux 7
Processor	Intel Core i5 @ 2.3 GHz	Intel Core i7-7700 CPU @ 3.60GHz x 8
Memory	8 GB 2133 MHz LPDDR3	15.5 GiB

4.4 Python Implementation

All of the work have been implemented in Python version 3.7.3. We have followed PEP-8 [2] coding style for our implementation. Table 16 lists the main library packages utilised during various phases of the project. NumPy [67] and Pandas [51] libraries have been used in every phase of the project. Matplotlib [68] library is used for data visualisation. Scikit-learn [43] and Imbalanced-learn [69] have been used for implementing machine learning models. Pomegranate [65] library package is for learning Bayesian networks for probabilistic reasoning.

Table 17 shows the list of machines used during implementation. Machine 2 has been used extensively for hyperparameter tuning of the machine learning models.

5 Results, Analysis and Evaluation

5.1 Supervised Learning

Table 18 reflects the versions maintained during implementation of the proposed design and methodologies in order to meet our objectives. The versions are followed throughout the code as well as this paper for consistency. The difference in binary classification for Phase 3 version 1.2 and 1.3 is that only Balanced Random Forest Classifier is used for training and evaluation in the former version whereas a combination of classifiers in the latter. There is no difference between Phase 3 version 1.3 and 1.4 for binary classification.

Further, the exception handling is also implemented in these two versions as there is a chance of run-time errors during cross validation if no sample is found for any of the

[2] https://www.python.org/dev/peps/pep-0008/

Table 18: Versioning for Supervised Learning Implementation

Phase	Type	Version	Feature Selection	Cross Val Scoring	Binary	Multi-class
1	Unweighted	1.1	No	10-fold balanced_accuracy	Yes	No
		1.2	Yes			
2	Over-Sampling	1.1	No	10-fold balanced_accuracy	Yes	No
		1.2	Yes			
2	Under-Sampling	1.1	No	5-fold balanced_accuracy	Yes	No
		1.2	Yes			Yes
3	Cost-Sensitive	1.1	No	5-fold balanced_accuracy	Yes	No
		1.2	Yes			
3	Cost-Sensitive	1.3	Yes	5-fold balanced_accuracy	Yes	Yes
		1.4	Yes	5-fold*		

*balanced_accuracy for binary; f1_micro for multiclass

classes which is quite possible in case of multiclass classification where we have very low proportion of samples for one of the classes in some cases. In particular, the record will be skipped and our Predictor component will continue classifying the samples for next question in the data-set. One such example from our implementation is illustrated in Figure 22. Please see the error handling for Q28 in this picture.

```
multi-class classification for Q25 completed.
multi-class classification for Q26 started.
multi-class classification for Q26 completed.
multi-class classification for Q27 started.
multi-class classification for Q27 completed.
multi-class classification for Q28 started.
Value Error: Class label 0 not present.
Record skipped.
multi-class classification for Q29 started.
multi-class classification for Q29 completed.
multi-class classification for Q30 started.
multi-class classification for Q30 completed.
multi-class classification for Q31 started.
multi-class classification for Q31 completed.
multi-class classification for Q32 started.
multi-class classification for Q32 completed.
multi-class classification for Q33 started.
multi-class classification for Q33 completed.
```

Figure 22: Exception Handling

Before analysing the results, it is best to point out the steps performed during implementation.

- Load the data-set into Pandas dataframe.
- Drop the *Prolific ID* column.

- Customise the column names by truncating additional space characters and sort the columns by demographics data and feedback questions in ascending order.
- Discretise *Age* and *Online Presence* attributes by dividing into bins.
- Categorise labels from 1-6 to 0/ False and 7 to 1/ True for binary classification. Categorise labels for multiclass classification as proposed in Section 3.3.1.2.
- Drop rows with missing class labels for the particular question as stratified split will not be possible otherwise.
- Split the data into training and test data with 7:3 ratio in stratified manner using class labels.
- Standardise and normalise the data for each variable excluding labels.
- Assign weights to each class using *compute_class_weight* function in scikit-learn library in case of Cost-Sensitive learning.
- Run classifiers with default parameters using Predictor component and verify the cross validation scores.
- Tune hyperparameters and identify the best classifier for each question using Estimator component.
- Copy the estimator output manually from the text file and use it in the Predictor component.
- Run the Predictor component to train the best classifier on the training data and make prediction on the test data.
- Write results for naive classification, cross validation and prediction scores to three different excel files.

5.1.1 Binary Classification

The results for binary classification using all the features are shown in Table 19 and Figure 23. Only the balanced accuracy score (Equation 4.7) is considered during the comparison for better clarity and also it is the correct metric to look for in case of imbalanced data. The detailed results for all the versions are included in Appendix **??** for cross validation scores and A.1.1 for prediction scores.

Clearly, the classifiers using Under-Sampling and Cost-Sensitive learning methods have outperformed the rest of the methods. The performance with cost-sensitive learning is almost equivalent to under-sampling in majority of the cases. It should be noted that the ensemble of samplers (classifiers) were used during under-sampling (as shown in Figure 20) in comparison to cost-sensitive learning in that a combination of basic and ensemble classifiers were used for prediction. So our Estimator component must have produced the basic model performing better than the ensemble of classifiers on 70%

Table 19: Performance Comparison of Binary Classification V1.1

(*Samples with All Features*)

Question	Unweighted	Over-Sampling	Under-Sampling	Cost-Sensitive
	Balanced Accuracy (%)			
Q01	92.71	84.91	90.55	94.1
Q02	98.31	98.31	98.15	98.15
Q03	91.72	86.95	93.69	94.35
Q04	62.36	63.42	79.97	82.22
Q05	97.29	50.6	97.29	97.89
Q06	93.3	93.92	92.98	93.3
Q07	88.58	87.68	90.84	90.84
Q08	68.72	62.87	82.51	79.28
Q09	87.66	80.13	94.12	93.58
Q10	82.7	82.16	89.05	88.54
Q11	86.34	76.23	85.36	89.22
Q12	59.82	59.23	82.61	82.97
Q13	73.72	74.43	86.97	84.16
Q14	81.14	75.05	81.3	80.59
Q15	70.41	71.36	83.78	86.34
Q16	97.83	96.99	97.67	97.67
Q17	90.64	92.02	92.8	94.54
Q18	96.45	50.72	97.24	97
Q19	97.55	51.69	97.45	96.91
Q20	95.8	96.4	95.79	96.98
Q21	95.01	51.94	94.8	94.43
Q22	90.18	80.77	87.83	89.29
Q23	96.08	97.29	97.59	97.59
Q24	88.22	81.64	89.04	89.41
Q25	96.56	91.88	96.31	96
Q26	85.3	75.65	91.76	91.36
Q27	94.03	88.2	94.12	93.48
Q28	97.86	49.78	97.86	97.54
Q29	93.98	95.38	94.8	94.8
Q30	84.64	79.71	89.01	88.28
Q31	84.31	79.64	89.85	90.14
Q32	91.22	88.06	92.33	91.67
Q33	95.47	94.64	96.12	96.41
Q34	85.64	80.98	88.68	88.41
Q35	90.49	86.02	91.22	90.72
Q36	94.49	85.35	95.39	94.83
Q37	50	51.13	92.6	91.97
Q38	95	51.88	95.62	95.62
Q39	80.51	72.32	87.25	85.98
Q40	90.66	80.89	91.94	91.4
Q41	90.72	91.09	91.17	92.49
Q42	95.39	91.28	95.1	95.51
Q43	95.18	83.89	94.91	94.64
Q44	94	91.39	95.68	95.41
Q45	90.72	86.05	94.39	93.78
Q46	83.17	82.36	87.65	89.11
Q47	94.29	93.07	93.95	94.27
Q48	95.06	89.18	95.15	95.99
Q49	89.63	86.8	88.3	91.06
Q50	61.51	64.25	81.49	84.63

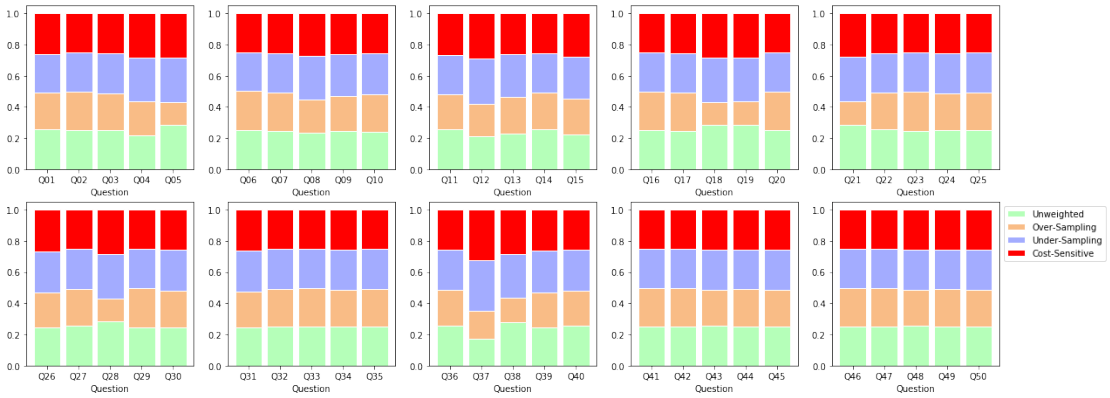

Figure 23: Performance Comparison of Binary Classification V1.1

training data for the particular case which eventually did not perform well on unseen test data. We can definitely achieve better performance using only ensemble classifiers with cost-sensitive learning after tuning the right hyperparameters.

There is a significant performance improvement using under-sampling and cost-sensitive learning in cases (Q04, Q08, Q12, Q13, Q15, Q37 and Q50) where the performance of classifiers was less than 75% using unweighted learning. The reason why classifiers with Over-Sampling technique did not perform well is because we changed the reality by generating synthetic samples in the training data while using the same set of classifiers as in Unweighted learning phase. Although such classifiers performed much better during cross-validation, they suffered immensely on the test data. The detailed results (Appendix A.1.1) also revealed that the classifiers detected more false positives at the expense of detecting as many samples from the positive minority class. However, this can be rectified by assigning weights to each class similar to cost-sensitive learning because we need to modify the objective function as we changed the reality by generating new synthetic samples for the training data. The performance boost can also be observed using ensemble of samplers as in Under-Sampling method.

The results of these methods using only informative features are in Table 20 and Figure 24. Ensemble of samplers using under-sampling technique are found to be the clear winner in this category.

Although we expected to achieve better performance with all these methods using only informative features, it is to be noted that we did not change any of the hyperparameters after feature selection. That means the classifiers did not generalise that well as we expected using same set of hyperparameters. One such example is Q37 under cost-sensitive category. Again, the performance can further be improved by fine tuning the hyperparameters for classifiers with under-sampling and cost-sensitive learning. This clearly shows the importance of these methods in case of imbalanced data.

Additionally, we measured the Precision, Recall, Specificity and other performance metrics for each category. The detailed results are shown in Appendix A.1.1. The

Table 20: Performance Comparison of Binary Classification V1.2

(*Samples with Informative Features*)

Question	Unweighted	Over-Sampling	Under-Sampling	Cost-Sensitive
	Balanced Accuracy (%)			
Q01	91.19	47.64	93.03	92.75
Q02	97.97	97.97	98.15	98.65
Q03	91.06	52.89	93.69	94.06
Q04	56.33	49.47	82.89	74.78
Q05	96.39	96.39	97.29	96.69
Q06	92.66	48.1	92.98	93.92
Q07	88.95	50.56	91.88	90.31
Q08	67.8	46.58	83.9	85.05
Q09	92.23	94.11	94.12	93.04
Q10	84.84	74.43	87.97	87.7
Q11	85.85	85.25	86.98	87.84
Q12	59.82	39.67	79.01	77.03
Q13	81.82	49.55	82.21	84.4
Q14	82.89	48.68	81.3	78.7
Q15	69.04	47.9	83.78	82.95
Q16	97.66	97.16	97.67	97.83
Q17	89.29	45.35	92.13	94.08
Q18	97	50.4	97.24	97.24
Q19	96.37	50.73	97.45	96.59
Q20	97.58	97.58	95.79	97.58
Q21	94.8	94.8	94.8	94.22
Q22	89.57	50.31	89.36	85.26
Q23	97.29	96.99	96.08	97.29
Q24	87.56	90.64	89.77	89.41
Q25	96.56	50.06	96.31	95.75
Q26	86.37	51.49	91.76	91.12
Q27	94.03	49.32	93.48	92.6
Q28	95.71	95.93	97.54	94.31
Q29	94.93	94.14	94.8	94.72
Q30	84.64	86.73	89.26	88.28
Q31	85.38	47.5	89.85	90.14
Q32	92.53	50.79	91.67	91.96
Q33	95.47	95.47	97.28	95.47
Q34	88.47	88.17	88.68	88.41
Q35	90.49	92.48	91.22	91.22
Q36	94.05	72.52	95.11	94.55
Q37	50	50.29	92.6	50
Q38	96.31	50.75	96.69	95.31
Q39	78.92	48.15	87.25	83.15
Q40	90.66	49.5	91.94	90.89
Q41	91.01	48.52	92.12	90.14
Q42	95.39	50.9	93.62	95.39
Q43	94.92	48.92	94.91	94.92
Q44	95.1	50.17	95.1	94
Q45	92.24	49.49	93.36	92.05
Q46	83.42	88.71	86.34	87.72
Q47	91.07	93.34	93.95	95.73
Q48	95.06	93.77	95.15	95.41
Q49	90.34	50.56	88.3	91.06
Q50	54.04	36.22	81.49	70.67

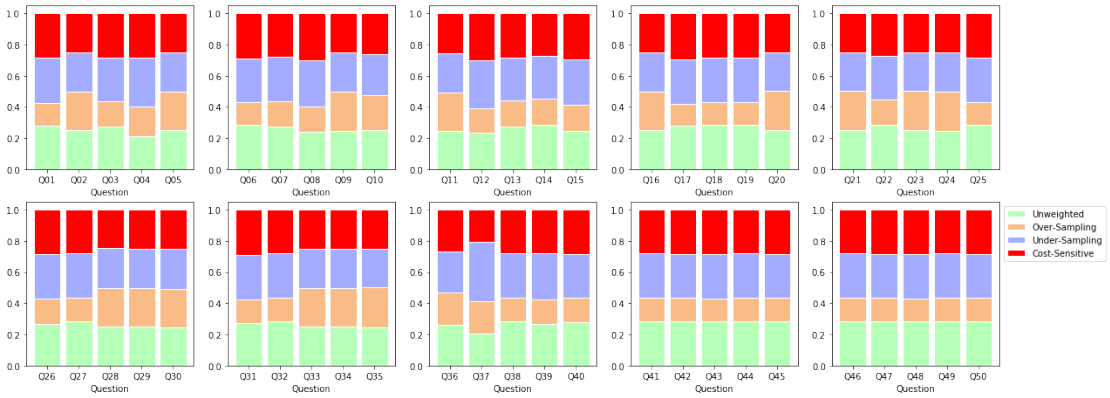

Figure 24: Performance Comparison of Binary Classification V1.2

numbers for these metrics excluding accuracy and balanced accuracy are with respect to positive class (1) in case of binary classification. The existing literature [70] explains that the precision-recall curve is more informative than the ROC curve while evaluating binary classifiers on imbalanced data. There is also a trade-off between precision and recall. When we tune our models to detect as many samples from the minority class, the precision suffers due to high recall. In other words, there will be more false positives when minority class is a positive class. Therefore, when we look at our detailed results, the precision can be observed very low due to high recall for minority class samples. F1 measure which is a harmonic mean of precision and recall also fluctuated accordingly.

Furthermore, the specificity score also went down when the classifier predicted low precision and high recall on positive minority class. This is because more false positives are coming out from the negative class. Hence, the true negative rate plummeted due to prediction of less number of correct negative samples.

Precision, f1 and specificity scores are found to be low with over-sampling, under-sampling and cost-sensitive learning in comparison to unweighted learning due to above reasons. We focused mainly on achieving high balanced accuracy for our problem. However, if we are concerned about getting high precision on positive class samples while maintaining the reasonable recall scores, we have to fine tune the models accordingly. This is usually expected in fraud detection and medical diagnosis scenarios. Basically, we have to determine the cost of FP, TP, FN and/ or TN according to the given requirements and then select the right metric to fine tune the classifiers.

5.1.2 Multiclass Classification

The results for Phase 3 version 1.4 (models fine-tuned for f1_micro) are shown in Table 21. We have observed that the models with same parameters used during binary classification performed poorly for multiclass classification. This was expected due to highly imbalanced sample proportion in three classes. With multiclass classification, it

is challenging to obtain high balanced accuracy score in such case. Therefore, we should focus on one particular class and fine tune our models. The micro average of precision, recall or f1 scores can also be taken into account to achieve desired results depending upon the requirement.

5.2 Unsupervised Learning

As mentioned in Section 4.2, we applied K-Means clustering on each question data after feature selection and engineering. The labels were also included in order to find groupings along with the truthfulness measure. The optimal number of clusters was found to be 2 for each question. The results for Q01-Q10 and Q11-Q20 are shown in Table 22 and 23 respectively.

The clusters are not perfect which is expected as we achieved the average silhouette score to be between 0.6 and 0.8. Younger people are found to be less uncomfortable and more truthful for questions Q03 (DOB), Q09 (Employer Name), Q13 (Personal Email) and Q14 (Professional Email) etc. whereas they are more uncomfortable and less truthful for Q06 (City of Residence), Q07 (Home Postcode), Q10 (Workplace Postcode), Q11 (Work Address), Q15 (Phone Number) etc.

These results imply that the younger generation are not so truthful regarding residential and phone number details in this context. Also, they require less effort to answer the questions for which they are comfortable. This is as we expected to find out.

Mostly all of the people are found to be truthful answering Q02 (Gender), Q16 (Ethnicity) and Q20 (Relationship Status) etc. regardless of the age. We can carry out similar analysis for any category of questions using our results. The remaining results can be found in Appendix A.2.

Apart from having a clear advantage over statistical analysis techniques, the results are found to be useful due to the fact that most of the sophisticated models like Neural Networks, Random Forest etc. which perform extremely well for prediction are black-box models and it is very hard to interpret the reasoning behind the results. The clustering results can provide a better picture in that case. Further, the cluster labels can be associated with the input data to run supervised learning algorithms for predictive modelling.

5.3 Probabilistic Reasoning

Here, we learnt Bayesian networks using data [71] [72] with the help of pomegranate [65] library in Python. As mentioned in Section 4.3, there is not an optimal way to determine structural relationships among too many variables in a realistic time-frame. Figure 25, 26 and 27 show the networks learnt for question blocks 1, 2 and 3 separately *(Please see questions in Table 1, 2 and 3).*

The results look interesting but complex enough to define any relationships quickly. It is amazing to see how dependencies exist among similar type of questions. One simple intuitive example to understand conditional independence can be inferred Figure 25 in

Table 21: Phase 3 - Cost-Sensitive Multiclass Classification Results V1.4

(*Samples with Informative Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision-Micro (%)	Recall-Micro (%)	F1-Micro
Q01	BalancedRandomForestClassifier	63.83	56.47	63.83	63.83	0.64
Q02	BalancedRandomForestClassifier	96.19	33.33	96.19	96.19	0.96
Q03	BalancedRandomForestClassifier	67.96	43.46	67.96	67.96	0.68
Q04	BalancedRandomForestClassifier	59.22	32.28	59.22	59.22	0.59
Q05	BalancedRandomForestClassifier	48.35	22.82	48.35	48.35	0.48
Q06	BalancedRandomForestClassifier	76	49.8	76	76	0.76
Q07	BalancedRandomForestClassifier	68.82	63.64	68.82	68.82	0.69
Q08	ExtraTreesClassifier	56.31	48.87	56.31	56.31	0.56
Q09	RandomForestClassifier	63.83	53.9	63.83	63.83	0.64
Q10	RandomForestClassifier	52.94	43.44	52.94	52.94	0.53
Q11	SVM	61.29	53.03	61.29	61.29	0.61
Q12	BalancedRandomForestClassifier	65.69	37.42	65.69	65.69	0.66
Q13	SVM	50	50.26	50	50	0.5
Q14	SVM	55.88	43.53	55.88	55.88	0.56
Q15	RandomForestClassifier	58.06	49.82	58.06	58.06	0.58
Q16	RandomForestClassifier	89.62	47.64	89.62	89.62	0.9
Q17	RandomForestClassifier	77.23	51.91	77.23	77.23	0.77
Q18	BalancedRandomForestClassifier	88.35	33.33	88.35	88.35	0.88
Q19	BalancedRandomForestClassifier	88.35	36.78	88.35	88.35	0.88
Q20	BalancedRandomForestClassifier	90.32	33.33	90.32	90.32	0.9
Q21	BalancedRandomForestClassifier	78.12	32.89	78.12	78.12	0.78
Q22	RandomForestClassifier	66.35	55.34	66.35	66.35	0.66
Q23	BalancedRandomForestClassifier	86.17	32.14	86.17	86.17	0.86
Q24	BalancedRandomForestClassifier	69.23	32.81	69.23	69.23	0.69
Q25	BalancedRandomForestClassifier	87.38	33.33	87.38	87.38	0.87
Q26	ExtraTreesClassifier	54.84	44.48	54.84	54.84	0.55
Q27	BalancedRandomForestClassifier	76.7	44.02	76.7	76.7	0.77
Q29	BalancedRandomForestClassifier	84.62	33.33	84.62	84.62	0.85
Q30	RandomForestClassifier	53.92	39.87	53.92	53.92	0.54
Q31	ExtraTreesClassifier	61.54	44.31	61.54	61.54	0.62
Q32	BalancedRandomForestClassifier	70.59	31.58	70.59	70.59	0.71
Q33	BalancedRandomForestClassifier	88.54	33.33	88.54	88.54	0.89
Q34	RandomForestClassifier	61.76	43.09	61.76	61.76	0.62
Q35	ExtraTreesClassifier	66.67	45.98	66.67	66.67	0.67
Q36	BalancedRandomForestClassifier	75.82	33.33	75.82	75.82	0.76
Q37	RandomForestClassifier	73.79	40.78	73.79	73.79	0.74
Q38	BalancedRandomForestClassifier	87.25	33.33	87.25	87.25	0.87
Q39	RandomForestClassifier	62.14	55.1	62.14	62.14	0.62
Q40	BalancedRandomForestClassifier	58.82	31.75	58.82	58.82	0.59
Q41	RandomForestClassifier	73.79	45.33	73.79	73.79	0.74
Q42	RandomForestClassifier	74.47	42.85	74.47	74.47	0.74
Q43	BalancedRandomForestClassifier	61.7	31.18	61.7	61.7	0.62
Q44	BalancedRandomForestClassifier	86.14	33.33	86.14	86.14	0.86
Q45	RandomForestClassifier	79.61	51.71	79.61	79.61	0.8
Q46	RandomForestClassifier	58.06	47.47	58.06	58.06	0.58
Q47	RandomForestClassifier	81.73	36.61	81.73	81.73	0.82
Q48	BalancedRandomForestClassifier	84.31	32.95	84.31	84.31	0.84
Q49	RandomForestClassifier	66.67	49.95	66.67	66.67	0.67
Q50	BalancedRandomForestClassifier	64.71	34	64.71	64.71	0.65

Table 22: K-Means Clustering Results for Q01-Q10

(*Samples with Informative Features*)

	01-Effort	**01-Uncomfortable**	**Age**	**01-Truthfulness**
Cluster 0	1	3	32	7
Cluster 1	2	6	31	2
	02-Effort	**02-Uncomfortable**	**Age**	**02-Truthfulness**
Cluster 0	1	1	32	7
Cluster 1	2	5	36	7
	03-Uncomfortable	**Age**	**Reciprocity**	**03-Truthfulness**
Cluster 0	2	31	4	7
Cluster 1	5	32	4	2
	Age	**Personal-Stability**	**Reciprocity**	**04-Truthfulness**
Cluster 0	31	4	4	6
Cluster 1	31	4	4	1
	05-Effort	**Age**	**Reciprocity**	**05-Truthfulness**
Cluster 0	1	30	5	7
Cluster 1	1	33	3	7
	06-Effort	**06-Uncomfortable**	**Age**	**06-Truthfulness**
Cluster 0	2	5	30	6
Cluster 1	1	1	32	7
	07-Uncomfortable	**Age**	**Reciprocity**	**07-Truthfulness**
Cluster 0	3	32	4	7
Cluster 1	5	30	4	2
	Age	**Personal-Stability**	**Reciprocity**	**08-Truthfulness**
Cluster 0	31	4	4	6
Cluster 1	32	4	4	1
	09-Effort	**09-Uncomfortable**	**Age**	**09-Truthfulness**
Cluster 0	2	6	31	5
Cluster 1	1	2	31	7
	10-Uncomfortable	**Age**	**Reciprocity**	**10-Truthfulness**
Cluster 0	3	32	4	7
Cluster 1	6	31	4	2

Table 23: K-Means Clustering Results for Q11-Q20

(*Samples with Informative Features*)

	11-Uncomfortable	**Age**	**Personal-Stability**	**11-Truthfulness**
Cluster 0	4	32	4	7
Cluster 1	6	30	4	2
	Age	**Personal-Stability**	**Reciprocity**	**12-Truthfulness**
Cluster 0	31	4	4	6
Cluster 1	32	4	4	1
	13-Uncomfortable	**Age**	**Reciprocity**	**13-Truthfulness**
Cluster 0	4	31	4	7
Cluster 1	5	32	4	1
	14-Uncomfortable	**Age**	**Reciprocity**	**14-Truthfulness**
Cluster 0	4	31	4	7
Cluster 1	6	32	4	1
	15-Uncomfortable	**Age**	**Reciprocity**	**15-Truthfulness**
Cluster 0	4	34	4	7
Cluster 1	6	31	4	1
	16-Effort	**Age**	**Personal-Stability**	**16-Truthfulness**
Cluster 0	1	27	4	7
Cluster 1	1	47	4	7
	17-Effort	**17-Uncomfortable**	**Reciprocity**	**17-Truthfulness**
Cluster 0	2	1	4	7
Cluster 1	2	5	4	6
	18-Effort	**18-Uncomfortable**	**Personal-Stability**	**18-Truthfulness**
Cluster 0	2	5	4	6
Cluster 1	1	1	4	7
	19-Effort	**Age**	**Personal-Stability**	**19-Truthfulness**
Cluster 0	4	33	4	5
Cluster 1	1	31	4	7
	Age	**Personal-Stability**	**Reciprocity**	**20-Truthfulness**
Cluster 0	27	4	4	7
Cluster 1	48	4	4	7

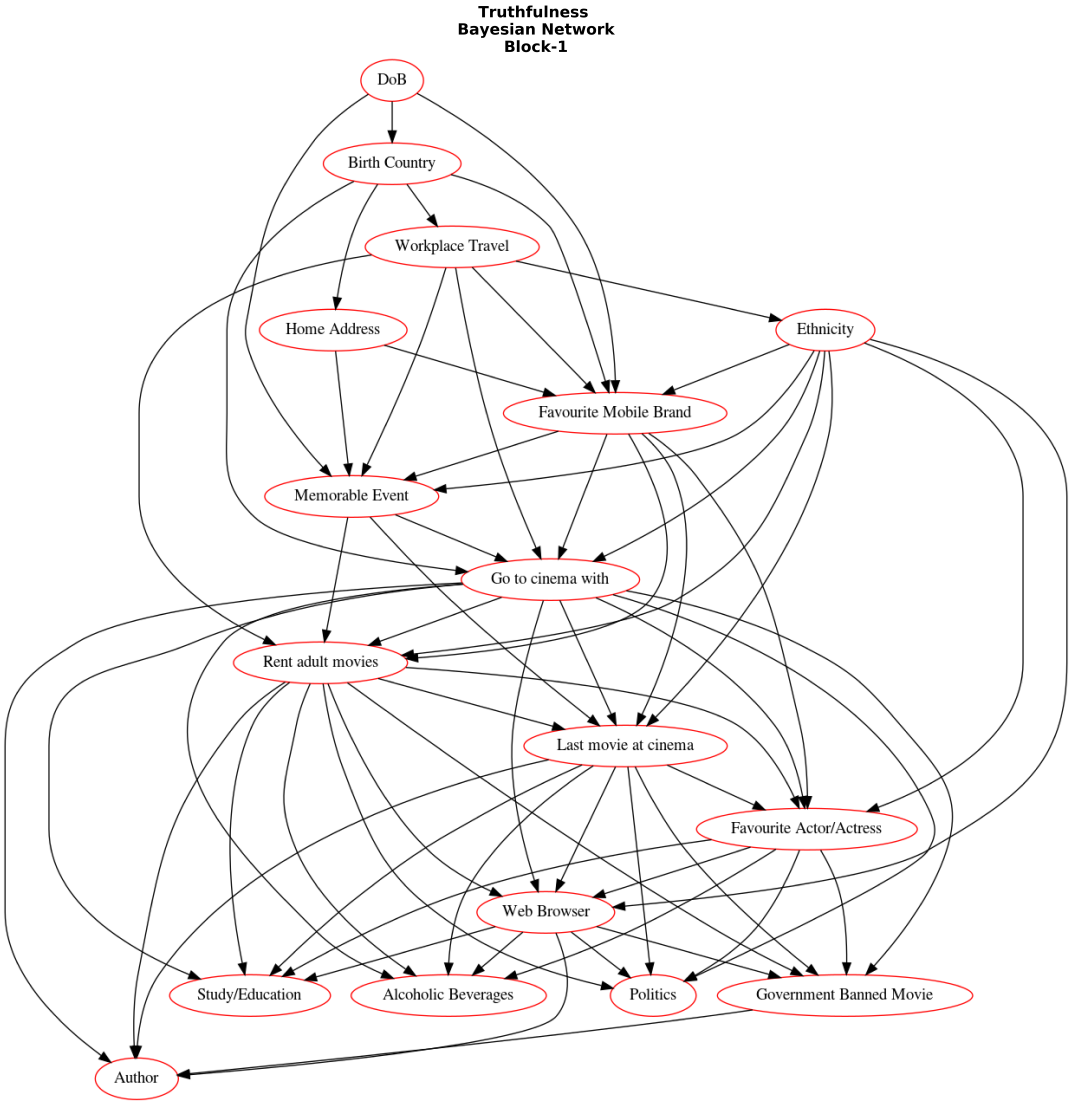

Figure 25: Exact Truthfulness Bayesian Network for Question Block 1

which *DoB* and *Workplace Travel* are found to be conditional independent given the evidence of *Birth Country* and *Home Address*. This follows from the equation 3.5.

We can make similar kind of inferences from any of these structures (Fig. 25, 26, 27) given some evidence. For a new user, a learnt model can be used to determine the measure of unknown factors given some evidence (i.e. a user sample with values available for some features/ variables). In other words, if we know how truthful (measured on 7-point Likert scale) someone is while answering *Gender*, *Personal Email*, *Holiday Destination*, *Sexual Orientation*, and *Hobby/Pastime*, we can use our model learnt for

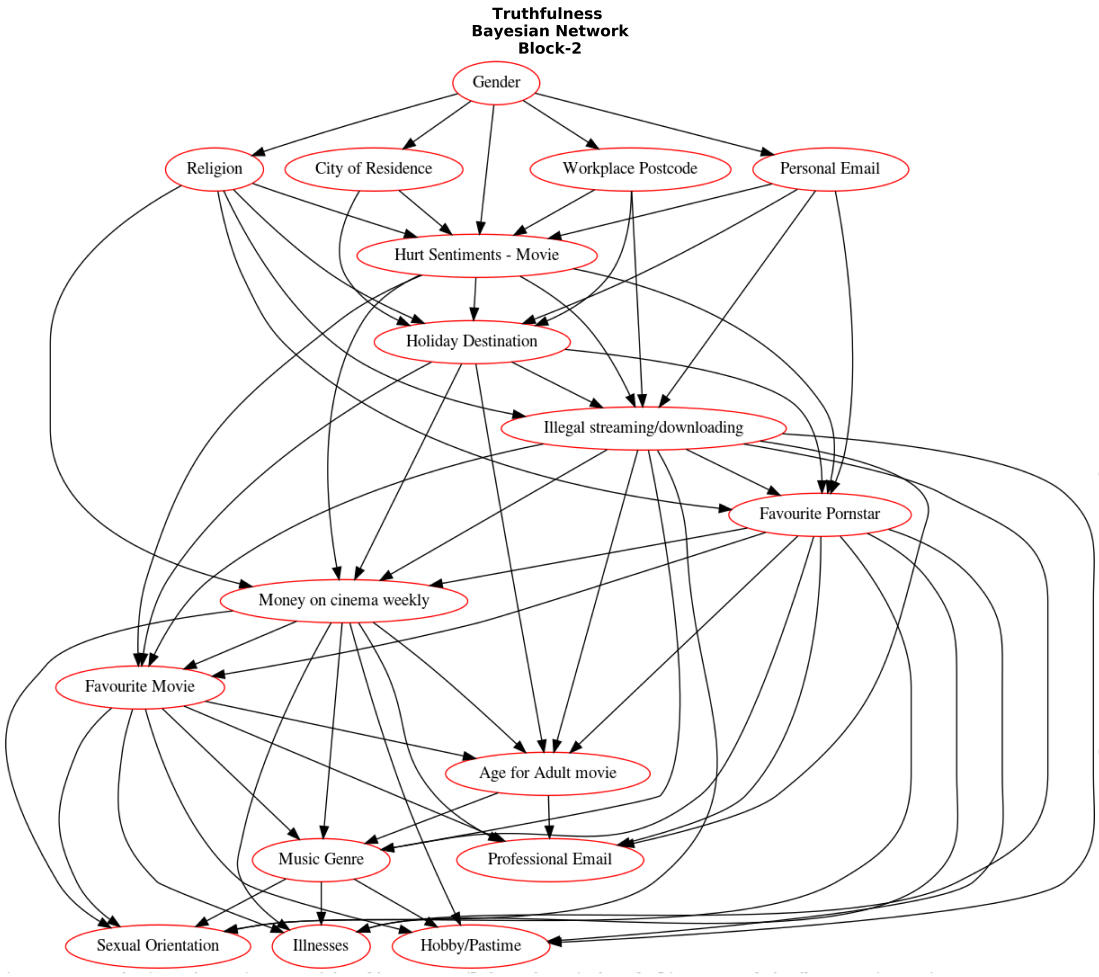

Figure 26: Exact Truthfulness Bayesian Network for Question Block 2

Question Block 1 to determine his/ her estimated truthfulness measures for all other questions in that block. This works according to the maximum likelihood estimation (MLE) principle [73] behind the scenes. However, the time complexity to predict values depend upon the number of known/ unknown variables and complexity of the model.

The only limitation with this method is that we do not have one single network for all 50 questions. Figure 28 shows the optimal tree like structure which is a fast approximation to the Bayesian network for all the questions.

This structure makes intuitive sense apart from nodes close to the root. In summary,

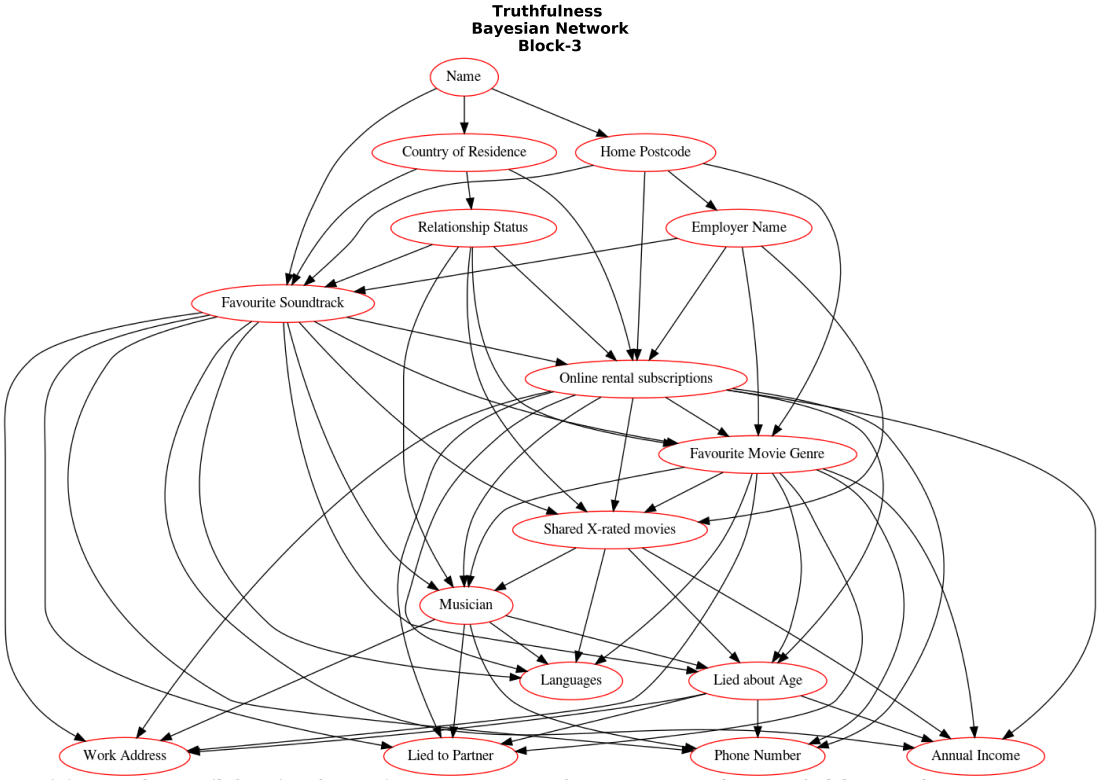

Figure 27: Exact Truthfulness Bayesian Network for Question Block 3

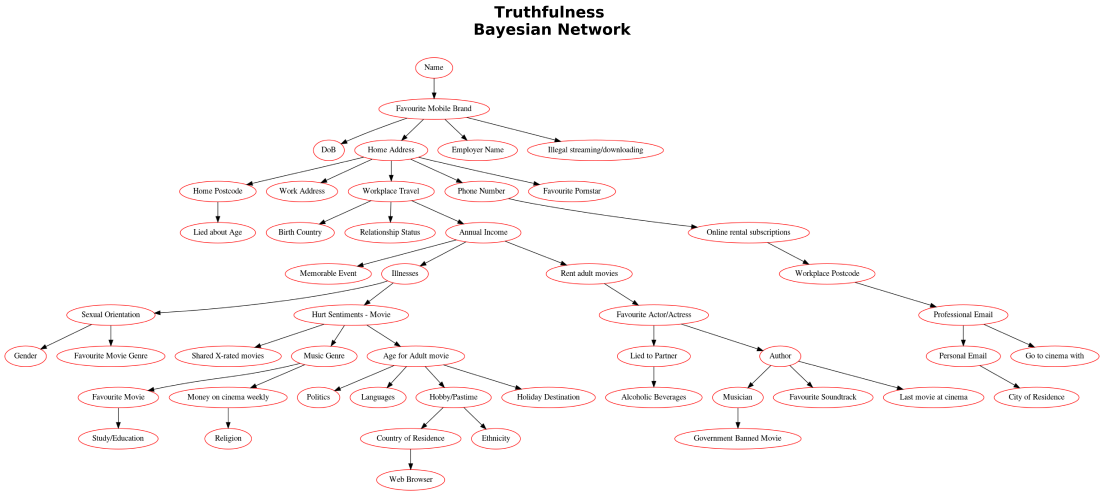

Figure 28: Chow-Liu Tree of Truthfulness Measure for Q01-Q50

the Chow-Liu tree provides a clear picture to learn inferences when we have too many factors. On the other hand, we can select the crucial factors using prior or expert knowledge and learn the optimal Bayesian network to determine accurate measures for only those factors.

5.4 Limitations and Future Work

A typical machine learning algorithm needs to be trained on thousands to millions of examples for *simple* and *complex* problems respectively. Our work was limited to less than 1000 samples in the context of purchasing discounted movie tickets. Therefore, we need more data to make our models perform effectively in a variety of contexts. Since it is usually very time-consuming and expensive to get true labels for any data-set, we look forward to use snorkel [74] library in Python which is based on *weak supervision*. This library allows to write *label functions* to generate *noisy* labels which are then passed through the generative model to provide a single, confidence-weighted training label for each data point after resolving conflicts. These labels can further be used to do predictive modelling.

Our work on multiclass classification is limited in a sense that we have not fine-tuned the models to the same extent as binary classification. Hence, our future work will also be to achieve high precision and recall (or f1_micro) scores for Class 3 samples as outlined in Table 9.

Moreover, our future work will concentrate on building deep learning neural networks [26] in order to evaluate their performance at each scale value of truthfulness measure (1-7) while incorporating class-imbalance techniques which we utilised in our current implementation. The approach is inspired from classic MNIST handwritten digit classification in [75]. This work will be implemented using Keras [76] and TensorFlow [77] frameworks. Although we used the anonymised version of the data-set, the existing study has shown that this heuristic is unable to preserve the privacy of individuals. Therefore, we shall also incorporate differential privacy mechanisms with our machine learning models based on the requirements.

6 Conclusion

In this research, we started by doing analysis on the given data-set. We looked at the type of data, underlying distribution and data correlations. We found that *effort*, *relevance* and *uncomfortable* were the key features determining the *truthfulness* of the answer. These findings along with the study of existing literature shaped our design choices. Subsequently, the problem under discussion was divided into tasks-*supervised learning*, *unsupervised learning* and *probabilistic reasoning*. The machine learning workflow was presented explaining step-by-step process to meet our objectives.

For supervised learning, the task was further divided into binary and multiclass classification. The data was split into training and test data-sets in a stratified manner which maintained the sample proportion in training and test data as it was in the original data-set. The data was cleaned and transformed to make it suitable for the classification. The models were trained and evaluated for each question in the training data using *cross-validation*. Further, the process of hyperparameter tuning was automated in the Estimator component, with the help of *GridSearchCV* class in scikit-learn library, allowing for several combinations of estimators and parameters for model evaluation. Subsequently, the best models were run on the test data to make predictions that were measured using several metrics including accuracy, balanced accuracy, precision, recall, specificity and f1 score.

Moreover, the informative features were extracted using Random Forest Classifier. The class imbalance problem was solved using over-sampling, under-sampling and cost-sensitive learning based methods. Afterwards, we trained and evaluated our models first using all the features and then with informative features. Since accuracy was not found to be the suitable metric for the imbalanced data, the models were fine-tuned with respect to the balanced accuracy. The classic as well as ensemble models employing under-sampling and cost-sensitive methods outperformed others in terms of performance. We recommend using cost-sensitive learning as most state-of-the-art models have the *class_weight* parameter to handle imbalanced data. The models using only informative features have achieved more or less the same performance with the benefit of less training time and reduced complexity to train on less number of features.

These models were also trained and evaluated for multiclass classification. The base models performed poorly due to highly imbalanced data. Subsequently, it was concluded that it would be challenging to achieve good balanced accuracy score for each class. Our work was limited in this case as we did not fine tune the hyperparameters further. Nevertheless, we achieved better results using f1_micro as the scoring metric. Our future work in this case is proposed in Section 5.4.

The data analysis and technical implementation was carried out using Python programming language and the open-source library packages available in Python. This has been outlined in Table 16.

Based on our research, it is evident that state-of-the-art machine learning models can learn from data effectively and predict outcomes with high balanced accuracy while employing suitable methods like cost-sensitive learning in case of imbalanced data. Unlike

statistical methods which are not suitable for large data-sets, we can train these models on thousands to millions of data to do predictive modelling.

Additionally, the interesting patterns in data were also discovered using unsupervised learning techniques. PCA was not found to be effective in our case as explained variance ratio was very low in the first few dimensions. However, K-Means clustering algorithm using only informative features produced clusters with reasonable silhouette scores. These clusters were then analysed to understand groupings within data as described in Section 5.2. We also generated the Bayesian networks to learn structural relationships in the truthfulness feature across questions. Since the search space of directed acyclic graphs is exponential in the number of variables, we provided the optimal tree like structure which is a fast approximation to the Bayesian network.

Lastly, we provided directions for our future work in Section 5.4 with focus on building deep neural networks to predict *truthfulness* measure at each scale value while dealing with highly imbalanced data using cost-sensitive learning. The work will also concentrate on weak supervision and implementing differential privacy algorithms to preserve the privacy of individuals.

References

[1] F. González, Y. Yu, A. Figueroa, C. López, and C. Aragon, "Global reactions to the cambridge analytica scandal: A cross-language social media study," in *Companion Proceedings of The 2019 World Wide Web Conference*, WWW '19, (New York, NY, USA), pp. 799–806, ACM, 2019.

[2] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, (Amsterdam, The Netherlands, The Netherlands), pp. 3–24, IOS Press, 2007.

[3] Z. Ghahramani, *Unsupervised Learning*, pp. 72–112. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," 2007.

[5] J. Fürnkranz and T. Kliegr, "A brief overview of rule learning," in *Rule Technologies: Foundations, Tools, and Applications* (N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, and D. Roman, eds.), (Cham), pp. 54–69, Springer International Publishing, 2015.

[6] L. van der Maaten, E. Postma, and H. Herik, "Dimensionality reduction: A comparative review," *Journal of Machine Learning Research - JMLR*, vol. 10, 01 2007.

[7] O. Chapelle, B. Scholkopf, and A. Zien, Eds., "Semi-supervised learning (chapelle, o. et al., eds.; 2006) [book reviews]," *IEEE Transactions on Neural Networks*, vol. 20, pp. 542–542, March 2009.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[9] C. M Bishop, "Model-based machine learning," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 371, p. 20120222, 02 2013.

[10] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, Mar 1986.

[11] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," vol. 2049, pp. 249–257, 01 2001.

[12] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[13] M. Sewell, "Ensemble learning," 2011.

[14] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, (Berlin, Heidelberg), pp. 1–15, Springer Berlin Heidelberg, 2000.

[15] R. Maclin and D. W. Opitz, "Popular ensemble methods: An empirical study," *CoRR*, vol. abs/1106.0257, 2011.

[16] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS '00, (London, UK, UK), pp. 1–15, Springer-Verlag, 2000.

[17] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, pp. 3–42, Apr 2006.

[18] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016.

[19] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[20] G. Guo, H. Wang, D. Bell, and Y. Bi, "Knn model-based approach in classification," 08 2004.

[21] A. Tharwat, T. Gaber, A. Ibrahim, and A. E. Hassanien, "Linear discriminant analysis: A detailed tutorial," *Ai Communications*, vol. 30, pp. 169–190,, 05 2017.

[22] J. Peng, K. Lida Lee, and G. M. Ingersoll, "An introduction to logistic regression analysis and reporting," *Journal of Educational Research - J EDUC RES*, vol. 96, pp. 3–14, 09 2002.

[23] I. Rish, "An empirical study of the naïve bayes classifier," *IJCAI 2001 Work Empir Methods Artif Intell*, vol. 3, 01 2001.

[24] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1, pp. 37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[25] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), pp. 161–168, ACM, 2006.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[27] H. K. Lam, "Class Lecture, Topic: "Multilayer Neural Networks", 7CCSMPNN, King's College London," 2019.

[28] K. Das and R. N. Behera, "A survey on machine learning: Concept,algorithms and applications," 2017.

[29] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, vol. 51, pp. 92:1–92:36, Sept. 2018.

[30] A. F. Westin, "Social and political dimensions of privacy," *Journal of Social Issues*, vol. 59, pp. 431 – 453, 07 2003.

[31] N. Malhotra, S. Kim, and J. Agarwal, "Internet users' information privacy concerns (iuipc): The construct, the scale, and a causal model," *Information Systems Research*, vol. 15, pp. 336–355, 12 2004.

[32] T. Donaldson and T. W. Dunfee, "Toward a unified conception of business ethics: Integrative social contracts theory," *Academy of Management Review*, vol. 19, no. 2, pp. 252–284, 1994.

[33] J. Grossklags and A. Acquisti, "When 25 cents is too much: An experiment on willingness-to-sell and willingness-to-protect personal information," in *Workshop on the Economics of Information Security Proceedings*, 2007.

[34] M. Lwin, J. Williams, and J. Wirtz, "Consumer online privacy concerns and responses: A power-responsibility equilibrium perspective," *Journal of the Academy of Marketing Science*, vol. 35, pp. 572–585, 02 2007.

[35] G. R. M. Mary J. Culnan, "The culnan-milne survey on consumers & online privacy notices: Summary of responses," *Interagency public workshop: Get noticed: Effective financial privacy notices*, pp. 47–54, 12 2001.

[36] M. Malheiros, S. Preibusch, and M. A. Sasse, ""fairly truthful": The impact of perceived effort, fairness, relevance, and sensitivity on personal data disclosure," in *Trust and Trustworthy Computing* (M. Huth, N. Asokan, S. Čapkun, I. Flechais, and L. Coles-Kemp, eds.), (Berlin, Heidelberg), pp. 250–266, Springer Berlin Heidelberg, 2013.

[37] N. J. D. NAGELKERKE, "A note on a general definition of the coefficient of determination," *Biometrika*, vol. 78, pp. 691–692, 09 1991.

[38] C. Dwork, *Differential Privacy*, pp. 338–340. Boston, MA: Springer US, 2011.

[39] C. Dwork, "Differential privacy: A survey of results," in *Theory and Applications of Models of Computation* (M. Agrawal, D. Du, Z. Duan, and A. Li, eds.), (Berlin, Heidelberg), pp. 1–19, Springer Berlin Heidelberg, 2008.

[40] M. Abadi, A. Chu, I. Goodfellow, B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *23rd ACM Conference on Computer and Communications Security (ACM CCS)*, pp. 308–318, 2016.

[41] "towardsdatascience.com", ""the 7 steps of machine learning"."

[42] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, no. 1, pp. 131 – 156, 1997.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Pearson Correlation Coefficient*, pp. 1–4. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[45] J. Rodgers and A. Nicewander, "Thirteen ways to look at the correlation coefficient," *American Statistician - AMER STATIST*, vol. 42, pp. 59–66, 02 1988.

[46] J. Hauke and K. Tomasz, "Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data," *Quaestiones Geographicae*, vol. 30, no. 2, pp. 87–93, 2011.

[47] J. Neyman, "On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection," *Journal of the Royal Statistical Society*, vol. 97, no. 4, pp. 558–625, 1934.

[48] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, (San Francisco, CA, USA), pp. 1137–1143, Morgan Kaufmann Publishers Inc., 1995.

[49] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[50] F. V. Jensen, *Introduction to Bayesian Networks*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 1996.

[51] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[52] W. wu, F. Jia, and C. Enders, "A comparison of imputation strategies for ordinal missing data on likert scale variables," *Multivariate Behavioral Research*, vol. 50, pp. 1–20, 07 2015.

[53] M. Saar-Tsechansky and F. Provost, "Handling missing values when applying classification models," *J. Mach. Learn. Res.*, vol. 8, pp. 1623–1657, Dec. 2007.

[54] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal.*, vol. 6, pp. 429–449, 2002.

[55] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 1263–1284, Sep. 2009.

[56] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, pp. 321–357, June 2002.

[57] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, Nov. 2009.

[58] X. Liu, J. Wu, and Z. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, pp. 539–550, April 2009.

[59] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 463–484, July 2012.

[60] C. Chen and L. Breiman, "Using random forest to learn imbalanced data," *University of California, Berkeley*, 01 2004.

[61] C. Elkan, "The foundations of cost-sensitive learning," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, (San Francisco, CA, USA), pp. 973–978, Morgan Kaufmann Publishers Inc., 2001.

[62] C. X. Ling, Q. Yang, J. Wang, and S. Zhang, "Decision trees with minimal costs," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, (New York, NY, USA), pp. 69–, ACM, 2004.

[63] T. Kodinariya and P. Dan Makwana, "Review on determining of cluster in k-means clustering," *International Journal of Advance Research in Computer Science and Management Studies*, vol. 1, pp. 90–95, 01 2013.

[64] E. Charniak, "Bayesian networks without tears.," *AI Magazine*, vol. 12, p. 50, Dec. 1991.

[65] J. Schreiber, "Pomegranate: fast and flexible probabilistic modeling in python," *Journal of Machine Learning Research*, vol. 18, no. 164, pp. 1–6, 2018.

[66] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, pp. 462–467, May 1968.

[67] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science Engineering*, vol. 13, pp. 22–30, March 2011.

[68] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science Engineering*, vol. 9, pp. 90–95, May 2007.

[69] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[70] T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLoS ONE*, vol. 10, p. e0118432, Mar. 2015.

[71] D. Margaritis, "Learning Bayesian Network Model Structure From Data, Ph.D dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2003, Accessed on: August 03, 2019., [Online].,Available: "https://www.cs.cmu.edu/"."

[72] J. Cheng, R. Greiner, J. Kelly, D. Bell, and W. Liu, "Learning bayesian networks from data: An information-theory based approach," *Artificial Intelligence*, vol. 137, no. 1, pp. 43 – 90, 2002.

[73] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of Mathematical Psychology*, vol. 47, no. 1, pp. 90 – 100, 2003.

[74] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," *Proc. VLDB Endow.*, vol. 11, pp. 269–282, Nov. 2017.

[75] Y. Lecun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Neural Information Processing Systems*, vol. 2, 11 1997.

[76] F. Chollet, *Deep Learning with Python*. Greenwich, CT, USA: Manning Publications Co., 1st ed., 2017.

[77] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

A Results

A.1 Supervised Learning

A.1.1 Prediction Scores

The prediction scores for phases outlined in Table 18 are covered in this section.

Table 24: Phase 1 - Unweighted Binary Classification Results V1.1

(*Samples with All Features*)

Question	**Model**	**Accuracy (%)**	**Balanced Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Specificity (%)**	**F1**	**ROC AUC**
Q01	DecisionTreeClassifier	95.58	92.71	92.45	87.5	97.93	0.9	0.96
Q02	ExtraTreesClassifier	97.99	98.31	95.28	100	96.62	0.98	0.99
Q03	RandomForestClassifier	91.57	91.72	82.35	92.11	91.33	0.87	0.98
Q04	AdaBoostClassifier	89.16	62.36	41.18	29.17	95.56	0.34	0.76
Q05	ExtraTreesClassifier	97.59	97.29	96.39	96.39	98.19	0.96	0.99
Q06	ExtraTreesClassifier	93.98	93.3	90.24	91.36	95.24	0.91	0.99
Q07	XGBClassifier	90.76	88.58	77.78	84.48	92.67	0.81	0.97
Q08	DecisionTreeClassifier	87.95	68.72	62.5	41.67	95.77	0.5	0.86
Q09	RandomForestClassifier	89.56	87.66	76.47	83.87	91.44	0.8	0.96
Q10	RandomForestClassifier	84.74	82.7	65.75	78.69	86.7	0.72	0.94
Q11	GaussianNB	87.55	86.34	61.29	84.44	88.24	0.71	0.93
Q12	XGBClassifier	90.36	59.82	41.67	22.73	96.92	0.29	0.82
Q13	DecisionTreeClassifier	83.13	73.72	50	59.52	87.92	0.54	0.88
Q14	DecisionTreeClassifier	87.55	81.14	58.33	71.79	90.48	0.64	0.88
Q15	BaggingClassifier	88.76	70.41	60	45.45	95.37	0.52	0.88
Q16	RandomForestClassifier	97.59	97.83	95.15	98.99	96.67	0.97	0.98
Q17	BaggingClassifier	93.98	90.64	96.83	82.43	98.86	0.89	0.99
Q18	RandomForestClassifier	96.39	96.45	93.62	96.7	96.2	0.95	1
Q19	RandomForestClassifier	97.19	97.55	93.81	98.91	96.18	0.96	0.99
Q20	RandomForestClassifier	95.98	95.8	93.02	95.24	96.36	0.94	0.99
Q21	RandomForestClassifier	93.57	95.01	83.33	98.68	91.33	0.9	0.98
Q22	ExtraTreesClassifier	91.97	90.18	83.82	86.36	93.99	0.85	0.96
Q23	RandomForestClassifier	96.39	96.08	94.05	95.18	96.99	0.95	0.99
Q24	RandomForestClassifier	88.76	88.22	78.57	86.84	89.6	0.82	0.92
Q25	DecisionTreeClassifier	95.58	96.56	89	100	93.12	0.94	0.98
Q26	GaussianNB	86.75	85.3	60.94	82.98	87.62	0.7	0.92
Q27	GaussianNB	92.77	94.03	82.8	97.47	90.59	0.9	0.96
Q28	RandomForestClassifier	97.59	97.86	94.85	98.92	96.79	0.97	0.99
Q29	RandomForestClassifier	93.17	93.98	83.91	96.05	91.91	0.9	0.99
Q30	GaussianNB	87.15	84.64	65.08	80.39	88.89	0.72	0.84
Q31	GaussianNB	86.35	84.31	69.01	80.33	88.3	0.74	0.94
Q32	RandomForestClassifier	90.36	91.22	78.89	93.42	89.02	0.86	0.96
Q33	GaussianNB	94.78	95.47	88.3	97.65	93.29	0.93	0.98
Q34	SVM	84.74	85.64	65.12	87.5	83.78	0.75	0.8
Q35	LogisticRegression	90.36	90.49	76.62	90.77	90.22	0.83	0.87
Q36	RandomForestClassifier	93.98	94.49	84.62	95.65	93.33	0.9	0.98
Q37	DecisionTreeClassifier	31.73	50	31.73	100	0	0.48	0.89
Q38	DecisionTreeClassifier	93.57	95	84.76	100	90	0.92	0.99
Q39	GaussianNB	83.53	80.51	56.06	75.51	85.5	0.64	0.91
Q40	DecisionTreeClassifier	89.96	90.66	74.36	92.06	89.25	0.82	0.96
Q41	RandomForestClassifier	89.16	90.72	75.79	94.74	86.71	0.84	0.97
Q42	RandomForestClassifier	95.18	95.39	88.61	95.89	94.89	0.92	0.98
Q43	GaussianNB	93.57	95.18	80.26	98.39	91.98	0.88	0.98
Q44	ExtraTreesClassifier	93.57	94	87.37	95.4	92.59	0.91	0.98
Q45	RandomForestClassifier	92.77	90.72	86.36	86.36	95.08	0.86	0.97
Q46	GaussianNB	85.94	83.17	59.68	78.72	87.62	0.68	0.94
Q47	GaussianNB	93.98	94.29	88.04	95.29	93.29	0.92	0.97
Q48	DecisionTreeClassifier	93.57	95.06	84.47	100	90.12	0.92	0.97
Q49	BaggingClassifier	91.97	89.63	86.76	84.29	94.97	0.86	0.97
Q50	XGBClassifier	90.36	61.51	31.25	27.78	95.24	0.29	0.87

Table 25: Phase 1 - Unweighted Binary Classification Results V1.2

(*Samples with Informative Features*)

Question	**Model**	**Accuracy (%)**	**Balanced Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Specificity (%)**	**F1**	**ROC AUC**
Q01	AdaBoostClassifier	95.18	91.19	94	83.93	98.45	0.89	0.99
Q02	SVM	97.59	97.97	94.39	100	95.95	0.97	1
Q03	RandomForestClassifier	91.16	91.06	82.14	90.79	91.33	0.86	0.97
Q04	XGBClassifier	88.35	56.33	30.77	16.67	96	0.22	0.77
Q05	RandomForestClassifier	96.79	96.39	95.18	95.18	97.59	0.95	0.99
Q06	RandomForestClassifier	93.98	92.66	92.31	88.89	96.43	0.91	0.99
Q07	XGBClassifier	93.17	88.95	88.68	81.03	96.86	0.85	0.98
Q08	DecisionTreeClassifier	88.35	67.8	66.67	38.89	96.71	0.49	0.85
Q09	RandomForestClassifier	91.57	92.23	77.33	93.55	90.91	0.85	0.96
Q10	RandomForestClassifier	87.15	84.84	71.01	80.33	89.36	0.75	0.95
Q11	GaussianNB	86.75	85.85	59.38	84.44	87.25	0.7	0.93
Q12	KNeighborsClassifier	90.36	59.82	41.67	22.73	96.92	0.29	0.73
Q13	GaussianNB	85.54	81.82	55.17	76.19	87.44	0.64	0.9
Q14	SVM	88.76	82.89	61.7	74.36	91.43	0.67	0.79
Q15	GaussianNB	81.93	69.04	36.96	51.52	86.57	0.43	0.86
Q16	RandomForestClassifier	97.59	97.66	96.04	97.98	97.33	0.97	0.98
Q17	ExtraTreesClassifier	93.17	89.29	96.72	79.73	98.86	0.87	0.99
Q18	RandomForestClassifier	96.79	97	93.68	97.8	96.2	0.96	0.99
Q19	XGBClassifier	95.98	96.37	91.84	97.83	94.9	0.95	0.98
Q20	SVM	96.79	97.58	91.3	100	95.15	0.95	0.98
Q21	SVM	92.77	94.8	80.85	100	89.6	0.89	0.97
Q22	RandomForestClassifier	90.36	89.57	78.38	87.88	91.26	0.83	0.96
Q23	RandomForestClassifier	97.19	97.29	94.19	97.59	96.99	0.96	0.98
Q24	SVM	88.35	87.56	78.31	85.53	89.6	0.82	0.9
Q25	DecisionTreeClassifier	95.58	96.56	89	100	93.12	0.94	0.98
Q26	GaussianNB	87.15	86.37	61.54	85.11	87.62	0.71	0.93
Q27	GaussianNB	92.77	94.03	82.8	97.47	90.59	0.9	0.96
Q28	DecisionTreeClassifier	95.98	95.71	94.62	94.62	96.79	0.95	0.97
Q29	RandomForestClassifier	93.98	94.93	85.06	97.37	92.49	0.91	0.98
Q30	GaussianNB	87.15	84.64	65.08	80.39	88.89	0.72	0.83
Q31	SVM	87.95	85.38	73.13	80.33	90.43	0.77	0.91
Q32	SVM	91.16	92.53	79.35	96.05	89.02	0.87	0.95
Q33	GaussianNB	94.78	95.47	88.3	97.65	93.29	0.93	0.98
Q34	SVM	85.14	88.47	64.21	95.31	81.62	0.77	0.93
Q35	LogisticRegression	90.36	90.49	76.62	90.77	90.22	0.83	0.87
Q36	DecisionTreeClassifier	93.98	94.05	85.53	94.2	93.89	0.9	0.95
Q37	DecisionTreeClassifier	31.73	50	31.73	100	0	0.48	0.89
Q38	RandomForestClassifier	95.58	96.31	89.8	98.88	93.75	0.94	1
Q39	BaggingClassifier	85.94	78.92	63.46	67.35	90.5	0.65	0.92
Q40	DecisionTreeClassifier	89.96	90.66	74.36	92.06	89.25	0.82	0.96
Q41	RandomForestClassifier	89.56	91.01	76.6	94.74	87.28	0.85	0.97
Q42	DecisionTreeClassifier	95.18	95.39	88.61	95.89	94.89	0.92	0.97
Q43	SVM	93.17	94.92	79.22	98.39	91.44	0.88	0.99
Q44	SVM	93.98	95.1	86	98.85	91.36	0.92	0.98
Q45	SVM	93.57	92.24	86.76	89.39	95.08	0.88	0.97
Q46	GaussianNB	86.35	83.42	60.66	78.72	88.12	0.69	0.94
Q47	ExtraTreesClassifier	91.97	91.07	88.24	88.24	93.9	0.88	0.97
Q48	DecisionTreeClassifier	93.57	95.06	84.47	100	90.12	0.92	0.97
Q49	RandomForestClassifier	92.37	90.34	86.96	85.71	94.97	0.86	0.97
Q50	AdaBoostClassifier	90.76	54.04	22.22	11.11	96.97	0.15	0.86

Table 26: Phase 2 - Over-Sampled Binary Classification Results V1.1

(*Samples with All Features*)

Question	**Model**	**Accuracy (%)**	**Balanced Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Specificity (%)**	**F1**	**ROC AUC**
Q01	SVM	90.36	84.91	80.77	75	94.82	0.78	0.93
Q02	RandomForestClassifier	97.99	98.31	95.28	100	96.62	0.98	1
Q03	KNeighborsClassifier	89.56	86.95	84.72	80.26	93.64	0.82	0.87
Q04	SVM	84.34	63.42	27.27	37.5	89.33	0.32	0.72
Q05	RandomForestClassifier	34.14	50.6	33.6	100	1.2	0.5	0.98
Q06	ExtraTreesClassifier	94.38	93.92	90.36	92.59	95.24	0.91	0.97
Q07	SVM	87.55	87.68	68	87.93	87.43	0.77	0.92
Q08	SVM	79.92	62.87	33.33	38.89	86.85	0.36	0.69
Q09	KNeighborsClassifier	85.54	80.13	71.67	69.35	90.91	0.7	0.8
Q10	SVM	83.94	82.16	64	78.69	85.64	0.71	0.87
Q11	SVM	82.33	76.23	50.85	66.67	85.78	0.58	0.81
Q12	SVM	85.54	59.23	23.08	27.27	91.19	0.25	0.62
Q13	KNeighborsClassifier	82.73	74.43	49.06	61.9	86.96	0.55	0.74
Q14	SVM	84.34	75.05	50	61.54	88.57	0.55	0.78
Q15	KNeighborsClassifier	85.94	71.36	47.22	51.52	91.2	0.49	0.82
Q16	XGBClassifier	96.79	96.99	94.17	97.98	96	0.96	0.98
Q17	KNeighborsClassifier	93.17	92.02	88	89.19	94.86	0.89	0.97
Q18	RandomForestClassifier	37.75	50.72	36.89	98.9	2.53	0.54	0.94
Q19	RandomForestClassifier	39.36	51.69	37.76	98.91	4.46	0.55	0.92
Q20	ExtraTreesClassifier	96.39	96.4	93.1	96.43	96.36	0.95	0.99
Q21	XGBClassifier	33.73	51.94	31.38	98.68	5.2	0.48	0.86
Q22	KNeighborsClassifier	83.13	80.77	65.79	75.76	85.79	0.7	0.88
Q23	ExtraTreesClassifier	97.19	97.29	94.19	97.59	96.99	0.96	0.98
Q24	SVM	84.74	81.64	75.68	73.68	89.6	0.75	0.9
Q25	KNeighborsClassifier	92.77	91.88	90.8	88.76	95	0.9	0.94
Q26	SVM	84.34	75.65	58	61.7	89.6	0.6	0.87
Q27	KNeighborsClassifier	90.36	88.2	86.67	82.28	94.12	0.84	0.88
Q28	RandomForestClassifier	37.35	49.78	37.25	98.92	0.64	0.54	0.94
Q29	ExtraTreesClassifier	93.57	95.38	82.61	100	90.75	0.9	0.99
Q30	SVM	83.94	79.71	58.73	72.55	86.87	0.65	0.87
Q31	KNeighborsClassifier	85.14	79.64	70	68.85	90.43	0.69	0.8
Q32	KNeighborsClassifier	89.56	88.06	82.05	84.21	91.91	0.83	0.88
Q33	SVM	95.18	94.64	92.94	92.94	96.34	0.93	0.97
Q34	KNeighborsClassifier	83.13	80.98	64.47	76.56	85.41	0.7	0.81
Q35	KNeighborsClassifier	90.36	86.02	84.75	76.92	95.11	0.81	0.92
Q36	KNeighborsClassifier	89.16	85.35	82.81	76.81	93.89	0.8	0.85
Q37	XGBClassifier	33.73	51.13	32.23	98.73	3.53	0.49	0.66
Q38	XGBClassifier	38.15	51.88	36.63	100	3.75	0.54	0.88
Q39	KNeighborsClassifier	81.53	72.32	52.83	57.14	87.5	0.55	0.8
Q40	KNeighborsClassifier	86.35	80.89	74.58	69.84	91.94	0.72	0.81
Q41	ExtraTreesClassifier	89.16	91.09	75.26	96.05	86.13	0.84	0.96
Q42	SVM	92.77	91.28	87.67	87.67	94.89	0.88	0.94
Q43	KNeighborsClassifier	88.76	83.89	79.31	74.19	93.58	0.77	0.84
Q44	SVM	91.57	91.39	85.87	90.8	91.98	0.88	0.96
Q45	KNeighborsClassifier	88.76	86.05	77.94	80.3	91.8	0.79	0.86
Q46	SVM	85.94	82.36	60	76.6	88.12	0.67	0.84
Q47	ExtraTreesClassifier	92.37	93.07	84.38	95.29	90.85	0.9	0.97
Q48	KNeighborsClassifier	90.76	89.18	89.02	83.91	94.44	0.86	0.93
Q49	KNeighborsClassifier	89.16	86.8	80.28	81.43	92.18	0.81	0.87
Q50	SVM	85.94	64.25	22.58	38.89	89.61	0.29	0.72

Table 27: Phase 2 - Over-Sampled Binary Classification Results V1.2

(*Samples with Informative Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	XGBClassifier	25.7	47.64	21.59	87.5	7.77	0.35	0.28
Q02	GaussianNB	97.59	97.97	94.39	100	95.95	0.97	0.98
Q03	RandomForestClassifier	34.54	52.89	31.8	100	5.78	0.48	0.98
Q04	XGBClassifier	12.05	49.47	9.54	95.83	3.11	0.17	0.82
Q05	SVM	96.79	96.39	95.18	95.18	97.59	0.95	0.98
Q06	RandomForestClassifier	32.13	48.1	31.67	93.83	2.38	0.47	0.82
Q07	RandomForestClassifier	26.91	50.56	23.5	94.83	6.28	0.38	0.42
Q08	BaggingClassifier	18.47	46.58	13.54	86.11	7.04	0.23	0.34
Q09	ExtraTreesClassifier	91.97	94.11	76.25	98.39	89.84	0.86	0.97
Q10	SVM	83.13	74.43	68.63	57.38	91.49	0.62	0.67
Q11	ExtraTreesClassifier	84.34	85.25	54.17	86.67	83.82	0.67	0.87
Q12	XGBClassifier	12.45	39.67	7.02	72.73	6.61	0.13	0.19
Q13	XGBClassifier	19.28	49.55	16.74	95.24	3.86	0.28	0.24
Q14	XGBClassifier	20.48	48.68	15.28	89.74	7.62	0.26	0.49
Q15	XGBClassifier	14.06	47.9	12.76	93.94	1.85	0.22	0.17
Q16	KNeighborsClassifier	96.79	97.16	93.33	98.99	95.33	0.96	0.98
Q17	RandomForestClassifier	28.11	45.35	27.66	87.84	2.86	0.42	0.78
Q18	RandomForestClassifier	37.35	50.4	36.73	98.9	1.9	0.54	0.95
Q19	RandomForestClassifier	38.15	50.73	37.3	98.91	2.55	0.54	0.93
Q20	SVM	96.79	97.58	91.3	100	95.15	0.95	0.98
Q21	ExtraTreesClassifier	92.77	94.8	80.85	100	89.6	0.89	0.98
Q22	RandomForestClassifier	30.52	50.31	26.64	92.42	8.2	0.41	0.72
Q23	ExtraTreesClassifier	96.79	96.99	93.1	97.59	96.39	0.95	0.97
Q24	KNeighborsClassifier	89.56	90.64	77.17	93.42	87.86	0.85	0.92
Q25	XGBClassifier	36.14	50.06	35.77	98.88	1.25	0.53	0.12
Q26	XGBClassifier	21.29	51.49	19.34	100	2.97	0.32	0.51
Q27	XGBClassifier	31.73	49.32	31.43	97.47	1.18	0.48	0.46
Q28	SVM	95.98	95.93	93.68	95.7	96.15	0.95	0.96
Q29	ExtraTreesClassifier	92.37	94.14	80.65	98.68	89.6	0.89	0.98
Q30	AdaBoostClassifier	83.53	86.73	55.95	92.16	81.31	0.7	0.84
Q31	XGBClassifier	24.9	47.5	23.53	91.8	3.19	0.37	0.83
Q32	RandomForestClassifier	32.13	50.79	30.86	98.68	2.89	0.47	0.2
Q33	SVM	94.78	95.47	88.3	97.65	93.29	0.93	0.97
Q34	SVM	83.94	88.17	62	96.88	79.46	0.76	0.88
Q35	ExtraTreesClassifier	90.36	92.48	74.12	96.92	88.04	0.84	0.94
Q36	SVM	83.53	72.52	86.84	47.83	97.22	0.62	0.86
Q37	XGBClassifier	32.13	50.29	31.85	100	0.59	0.48	0.74
Q38	AdaBoostClassifier	37.35	50.75	36.1	97.75	3.75	0.53	0.96
Q39	BaggingClassifier	22.89	48.15	19.05	89.8	6.5	0.31	0.75
Q40	AdaBoostClassifier	26.91	49.5	25.1	95.24	3.76	0.4	0.69
Q41	RandomForestClassifier	30.52	48.52	29.88	94.74	2.31	0.45	0.71
Q42	RandomForestClassifier	31.73	50.9	29.71	97.26	4.55	0.46	0.64
Q43	SVM	25.7	48.92	24.48	95.16	2.67	0.39	0.89
Q44	RandomForestClassifier	36.55	50.17	35.02	95.4	4.94	0.51	0.94
Q45	XGBClassifier	29.32	49.49	26.29	92.42	6.56	0.41	0.83
Q46	KNeighborsClassifier	84.34	88.71	54.88	95.74	81.68	0.7	0.88
Q47	ExtraTreesClassifier	91.97	93.34	82.18	97.65	89.02	0.89	0.97
Q48	SVM	93.98	93.77	90	93.1	94.44	0.92	0.91
Q49	AdaBoostClassifier	28.92	50.56	28.34	100	1.12	0.44	0.83
Q50	XGBClassifier	19.68	36.22	4.95	55.56	16.88	0.09	0.27

Table 28: Phase 2 - Under-Sampled Binary Classification Results V1.1

(*Samples with All Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	BalancedRandomForestClassifier	95.18	90.55	95.83	82.14	98.96	0.88	0.99
Q02	BalancedRandomForestClassifier	97.99	98.15	96.15	99.01	97.3	0.98	0.99
Q03	BalancedRandomForestClassifier	92.77	93.69	82.95	96.05	91.33	0.89	0.98
Q04	EasyEnsembleClassifier	73.9	79.97	25.3	87.5	72.44	0.39	0.86
Q05	BalancedRandomForestClassifier	97.59	97.29	96.39	96.39	98.19	0.96	1
Q06	BalancedRandomForestClassifier	93.98	92.98	91.25	90.12	95.83	0.91	0.99
Q07	BalancedRandomForestClassifier	85.94	90.84	62.37	100	81.68	0.77	0.98
Q08	EasyEnsembleClassifier	81.93	82.51	43.48	83.33	81.69	0.57	0.88
Q09	BalancedRandomForestClassifier	91.16	94.12	73.81	100	88.24	0.85	0.96
Q10	BalancedRandomForestClassifier	85.14	89.05	62.77	96.72	81.38	0.76	0.94
Q11	EasyEnsembleClassifier	85.94	85.36	57.58	84.44	86.27	0.68	0.93
Q12	EasyEnsembleClassifier	79.52	82.61	28.36	86.36	78.85	0.43	0.9
Q13	BalancedRandomForestClassifier	79.92	86.97	45.56	97.62	76.33	0.62	0.91
Q14	BalancedRandomForestClassifier	75.5	81.3	38.04	89.74	72.86	0.53	0.91
Q15	BalancedRandomForestClassifier	76.31	83.78	35.23	93.94	73.61	0.51	0.91
Q16	BalancedRandomForestClassifier	97.19	97.67	93.4	100	95.33	0.97	0.98
Q17	BalancedRandomForestClassifier	93.17	92.8	86.08	91.89	93.71	0.89	0.98
Q18	BalancedRandomForestClassifier	96.79	97.24	92.78	98.9	95.57	0.96	1
Q19	BalancedRandomForestClassifier	96.79	97.45	92	100	94.9	0.96	0.99
Q20	BalancedRandomForestClassifier	95.58	95.79	91.01	96.43	95.15	0.94	0.99
Q21	BalancedRandomForestClassifier	92.77	94.8	80.85	100	89.6	0.89	0.98
Q22	BalancedRandomForestClassifier	83.53	87.83	62.14	96.97	78.69	0.76	0.96
Q23	BalancedRandomForestClassifier	97.19	97.59	93.18	98.8	96.39	0.96	0.98
Q24	EasyEnsembleClassifier	88.35	89.04	75.82	90.79	87.28	0.83	0.96
Q25	BalancedRandomForestClassifier	95.58	96.31	89.8	98.88	93.75	0.94	0.98
Q26	BalancedRandomForestClassifier	87.95	91.76	61.33	97.87	85.64	0.75	0.95
Q27	BalancedRandomForestClassifier	91.97	94.12	79.8	100	88.24	0.89	0.97
Q28	BalancedRandomForestClassifier	97.59	97.86	94.85	98.92	96.79	0.97	0.99
Q29	BalancedRandomForestClassifier	92.77	94.8	80.85	100	89.6	0.89	0.99
Q30	BalancedRandomForestClassifier	87.15	89.01	62.67	92.16	85.86	0.75	0.95
Q31	BalancedRandomForestClassifier	86.35	89.85	64.84	96.72	82.98	0.78	0.94
Q32	BalancedRandomForestClassifier	90.36	92.33	77.08	97.37	87.28	0.86	0.96
Q33	BalancedRandomForestClassifier	96.39	96.12	94.19	95.29	96.95	0.95	0.99
Q34	BalancedRandomForestClassifier	83.94	88.68	61.76	98.44	78.92	0.76	0.92
Q35	BalancedRandomForestClassifier	89.96	91.22	74.39	93.85	88.59	0.83	0.97
Q36	BalancedRandomForestClassifier	93.98	95.39	82.93	98.55	92.22	0.9	0.98
Q37	BalancedRandomForestClassifier	90.36	92.6	77.23	98.73	86.47	0.87	0.96
Q38	BalancedRandomForestClassifier	94.38	95.62	86.41	100	91.25	0.93	0.99
Q39	BalancedRandomForestClassifier	79.52	87.25	49	100	74.5	0.66	0.9
Q40	BalancedRandomForestClassifier	87.95	91.94	67.74	100	83.87	0.81	0.95
Q41	BalancedRandomForestClassifier	88.76	91.17	74	97.37	84.97	0.84	0.96
Q42	BalancedRandomForestClassifier	94.78	95.1	87.5	95.89	94.32	0.92	0.98
Q43	BalancedRandomForestClassifier	93.98	94.91	82.19	96.77	93.05	0.89	0.98
Q44	BalancedRandomForestClassifier	94.38	95.68	86.14	100	91.36	0.93	0.98
Q45	BalancedRandomForestClassifier	93.17	94.39	81.01	96.97	91.8	0.88	0.97
Q46	BalancedRandomForestClassifier	83.94	87.65	54.32	93.62	81.68	0.69	0.94
Q47	BalancedRandomForestClassifier	92.77	93.95	83.84	97.65	90.24	0.9	0.97
Q48	EasyEnsembleClassifier	94.38	95.15	87.63	97.7	92.59	0.92	0.97
Q49	BalancedRandomForestClassifier	87.55	88.3	72.41	90	86.59	0.8	0.97
Q50	BalancedRandomForestClassifier	79.92	81.49	24.19	83.33	79.65	0.38	0.91

Table 29: Phase 2 - Under-Sampled Binary Classification Results V1.2

(*Samples with Informative Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	EasyEnsembleClassifier	91.16	93.03	72.97	96.43	89.64	0.83	0.98
Q02	BalancedRandomForestClassifier	97.99	98.15	96.15	99.01	97.3	0.98	0.99
Q03	BalancedRandomForestClassifier	92.77	93.69	82.95	96.05	91.33	0.89	0.98
Q04	BalancedRandomForestClassifier	69.08	82.89	23.76	100	65.78	0.38	0.85
Q05	BalancedRandomForestClassifier	97.59	97.29	96.39	96.39	98.19	0.96	1
Q06	BalancedRandomForestClassifier	93.98	92.98	91.25	90.12	95.83	0.91	0.99
Q07	EasyEnsembleClassifier	87.55	91.88	65.17	100	83.77	0.79	0.96
Q08	EasyEnsembleClassifier	82.33	83.9	44.29	86.11	81.69	0.58	0.88
Q09	BalancedRandomForestClassifier	91.16	94.12	73.81	100	88.24	0.85	0.96
Q10	BalancedRandomForestClassifier	84.34	87.97	61.7	95.08	80.85	0.75	0.94
Q11	BalancedRandomForestClassifier	84.34	86.98	53.95	91.11	82.84	0.68	0.94
Q12	BalancedRandomForestClassifier	76.71	79.01	25	81.82	76.21	0.38	0.87
Q13	EasyEnsembleClassifier	78.31	82.21	43.02	88.1	76.33	0.58	0.86
Q14	BalancedRandomForestClassifier	75.5	81.3	38.04	89.74	72.86	0.53	0.91
Q15	BalancedRandomForestClassifier	76.31	83.78	35.23	93.94	73.61	0.51	0.91
Q16	BalancedRandomForestClassifier	97.19	97.67	93.4	100	95.33	0.97	0.98
Q17	BalancedRandomForestClassifier	92.77	92.13	85.9	90.54	93.71	0.88	0.99
Q18	BalancedRandomForestClassifier	96.79	97.24	92.78	98.9	95.57	0.96	1
Q19	BalancedRandomForestClassifier	96.79	97.45	92	100	94.9	0.96	0.99
Q20	BalancedRandomForestClassifier	95.58	95.79	91.01	96.43	95.15	0.94	0.99
Q21	BalancedRandomForestClassifier	92.77	94.8	80.85	100	89.6	0.89	0.98
Q22	BalancedRandomForestClassifier	90.76	89.36	80.28	86.36	92.35	0.83	0.96
Q23	BalancedRandomForestClassifier	96.39	96.08	94.05	95.18	96.99	0.95	0.98
Q24	BalancedRandomForestClassifier	88.35	89.77	74.74	93.42	86.13	0.83	0.94
Q25	BalancedRandomForestClassifier	95.58	96.31	89.8	98.88	93.75	0.94	0.98
Q26	BalancedRandomForestClassifier	87.95	91.76	61.33	97.87	85.64	0.75	0.95
Q27	BalancedRandomForestClassifier	91.57	93.48	79.59	98.73	88.24	0.88	0.97
Q28	BalancedRandomForestClassifier	97.19	97.54	93.88	98.92	96.15	0.96	0.99
Q29	BalancedRandomForestClassifier	92.77	94.8	80.85	100	89.6	0.89	0.98
Q30	BalancedRandomForestClassifier	87.55	89.26	63.51	92.16	86.36	0.75	0.95
Q31	BalancedRandomForestClassifier	86.35	89.85	64.84	96.72	82.98	0.78	0.94
Q32	BalancedRandomForestClassifier	89.96	91.67	76.84	96.05	87.28	0.85	0.95
Q33	BalancedRandomForestClassifier	96.79	97.28	92.31	98.82	95.73	0.95	0.99
Q34	BalancedRandomForestClassifier	83.94	88.68	61.76	98.44	78.92	0.76	0.92
Q35	BalancedRandomForestClassifier	89.96	91.22	74.39	93.85	88.59	0.83	0.97
Q36	BalancedRandomForestClassifier	93.57	95.11	81.93	98.55	91.67	0.89	0.98
Q37	BalancedRandomForestClassifier	90.36	92.6	77.23	98.73	86.47	0.87	0.96
Q38	EasyEnsembleClassifier	96.39	96.69	92.55	97.75	95.62	0.95	0.99
Q39	BalancedRandomForestClassifier	79.52	87.25	49	100	74.5	0.66	0.9
Q40	BalancedRandomForestClassifier	87.95	91.94	67.74	100	83.87	0.81	0.95
Q41	BalancedRandomForestClassifier	89.56	92.12	75	98.68	85.55	0.85	0.96
Q42	BalancedRandomForestClassifier	94.38	93.62	89.33	91.78	95.45	0.91	0.98
Q43	BalancedRandomForestClassifier	93.98	94.91	82.19	96.77	93.05	0.89	0.98
Q44	BalancedRandomForestClassifier	93.98	95.1	86	98.85	91.36	0.92	0.98
Q45	BalancedRandomForestClassifier	92.37	93.36	79.75	95.45	91.26	0.87	0.97
Q46	EasyEnsembleClassifier	83.13	86.34	53.09	91.49	81.19	0.67	0.93
Q47	BalancedRandomForestClassifier	92.77	93.95	83.84	97.65	90.24	0.9	0.97
Q48	EasyEnsembleClassifier	94.38	95.15	87.63	97.7	92.59	0.92	0.97
Q49	BalancedRandomForestClassifier	87.55	88.3	72.41	90	86.59	0.8	0.97
Q50	BalancedRandomForestClassifier	79.92	81.49	24.19	83.33	79.65	0.38	0.91

Table 30: Phase 3 - Cost-Sensitive Binary Classification Results V1.1

(*Samples with All Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	BalancedRandomForestClassifier	94.78	94.1	85.25	92.86	95.34	0.89	0.99
Q02	BalancedRandomForestClassifier	97.99	98.15	96.15	99.01	97.3	0.98	0.99
Q03	BalancedRandomForestClassifier	93.17	94.35	83.15	97.37	91.33	0.9	0.97
Q04	BalancedRandomForestClassifier	67.87	82.22	23.08	100	64.44	0.38	0.82
Q05	BalancedRandomForestClassifier	97.99	97.89	96.43	97.59	98.19	0.97	0.99
Q06	BalancedRandomForestClassifier	93.98	93.3	90.24	91.36	95.24	0.91	0.99
Q07	BalancedRandomForestClassifier	85.94	90.84	62.37	100	81.68	0.77	0.97
Q08	BalancedRandomForestClassifier	82.33	79.28	43.55	75	83.57	0.55	0.91
Q09	BalancedRandomForestClassifier	91.16	93.58	74.39	98.39	88.77	0.85	0.96
Q10	BalancedRandomForestClassifier	83.53	88.54	60	98.36	78.72	0.75	0.94
Q11	BalancedRandomForestClassifier	82.33	89.22	50.56	100	78.43	0.67	0.94
Q12	BalancedRandomForestClassifier	72.69	82.97	23.86	95.45	70.48	0.38	0.85
Q13	BalancedRandomForestClassifier	83.13	84.16	50	85.71	82.61	0.63	0.92
Q14	BalancedRandomForestClassifier	74.3	80.59	36.84	89.74	71.43	0.52	0.9
Q15	BalancedRandomForestClassifier	76.31	86.34	35.87	100	72.69	0.53	0.89
Q16	BalancedRandomForestClassifier	97.19	97.67	93.4	100	95.33	0.97	0.98
Q17	BalancedRandomForestClassifier	93.98	94.54	85.54	95.95	93.14	0.9	0.98
Q18	BalancedRandomForestClassifier	96.79	97	93.68	97.8	96.2	0.96	1
Q19	BalancedRandomForestClassifier	96.39	96.91	91.92	98.91	94.9	0.95	0.99
Q20	BalancedRandomForestClassifier	96.39	96.98	91.21	98.81	95.15	0.95	0.99
Q21	BalancedRandomForestClassifier	92.77	94.43	81.52	98.68	90.17	0.89	0.98
Q22	BalancedRandomForestClassifier	89.96	89.29	77.33	87.88	90.71	0.82	0.96
Q23	BalancedRandomForestClassifier	97.19	97.59	93.18	98.8	96.39	0.96	0.98
Q24	BalancedRandomForestClassifier	88.35	89.41	75.27	92.11	86.71	0.83	0.93
Q25	BalancedRandomForestClassifier	95.18	96	88.89	98.88	93.12	0.94	0.98
Q26	BalancedRandomForestClassifier	89.96	91.36	66.67	93.62	89.11	0.78	0.96
Q27	BalancedRandomForestClassifier	91.57	93.48	79.59	98.73	88.24	0.88	0.97
Q28	BalancedRandomForestClassifier	97.19	97.54	93.88	98.92	96.15	0.96	0.99
Q29	BalancedRandomForestClassifier	92.77	94.8	80.85	100	89.6	0.89	0.99
Q30	BalancedRandomForestClassifier	87.15	88.28	63.01	90.2	86.36	0.74	0.95
Q31	BalancedRandomForestClassifier	85.94	90.14	63.83	98.36	81.91	0.77	0.94
Q32	BalancedRandomForestClassifier	89.96	91.67	76.84	96.05	87.28	0.85	0.95
Q33	BalancedRandomForestClassifier	96.39	96.41	93.18	96.47	96.34	0.95	0.99
Q34	BalancedRandomForestClassifier	83.53	88.41	61.17	98.44	78.38	0.75	0.92
Q35	BalancedRandomForestClassifier	89.96	90.72	75	92.31	89.13	0.83	0.96
Q36	BalancedRandomForestClassifier	93.17	94.83	80.95	98.55	91.11	0.89	0.97
Q37	BalancedRandomForestClassifier	89.96	91.97	77	97.47	86.47	0.86	0.97
Q38	BalancedRandomForestClassifier	94.38	95.62	86.41	100	91.25	0.93	0.99
Q39	BalancedRandomForestClassifier	78.71	85.98	48	97.96	74	0.64	0.9
Q40	BalancedRandomForestClassifier	87.15	91.4	66.32	100	82.8	0.8	0.94
Q41	BalancedRandomForestClassifier	89.56	92.49	74.51	100	84.97	0.85	0.97
Q42	BalancedRandomForestClassifier	94.78	95.51	86.59	97.26	93.75	0.92	0.98
Q43	BalancedRandomForestClassifier	93.57	94.64	81.08	96.77	92.51	0.88	0.98
Q44	BalancedRandomForestClassifier	94.38	95.41	86.87	98.85	91.98	0.92	0.98
Q45	BalancedRandomForestClassifier	91.57	93.78	76.47	98.48	89.07	0.86	0.97
Q46	BalancedRandomForestClassifier	82.33	89.11	51.65	100	78.22	0.68	0.93
Q47	BalancedRandomForestClassifier	93.57	94.27	86.32	96.47	92.07	0.91	0.97
Q48	BalancedRandomForestClassifier	94.78	95.99	87	100	91.98	0.93	0.98
Q49	BalancedRandomForestClassifier	87.15	91.06	68.63	100	82.12	0.81	0.96
Q50	BalancedRandomForestClassifier	71.49	84.63	20.22	100	69.26	0.34	0.91

Table 31: Phase 3 - Cost-Sensitive Binary Classification Results V1.2

(*Samples with Informative Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	BalancedRandomForestClassifier	93.57	95.22	78.57	98.21	92.23	0.87	0.99
Q02	BalancedRandomForestClassifier	97.99	98.15	96.15	99.01	97.3	0.98	1
Q03	BalancedRandomForestClassifier	92.37	93.4	82.02	96.05	90.75	0.88	0.97
Q04	BalancedRandomForestClassifier	68.27	76.86	21.65	87.5	66.22	0.35	0.81
Q05	BalancedRandomForestClassifier	97.19	96.69	96.34	95.18	98.19	0.96	0.99
Q06	BalancedRandomForestClassifier	93.98	94.26	87.5	95.06	93.45	0.91	0.99
Q07	BalancedRandomForestClassifier	85.14	90.31	61.05	100	80.63	0.76	0.97
Q08	BalancedRandomForestClassifier	82.33	85.05	44.44	88.89	81.22	0.59	0.92
Q09	BalancedRandomForestClassifier	91.57	93.85	75.31	98.39	89.3	0.85	0.96
Q10	BalancedRandomForestClassifier	84.34	87.97	61.7	95.08	80.85	0.75	0.95
Q11	BalancedRandomForestClassifier	83.94	88.46	53.09	95.56	81.37	0.68	0.94
Q12	BalancedRandomForestClassifier	74.7	77.91	23.38	81.82	74.01	0.36	0.82
Q13	BalancedRandomForestClassifier	82.33	86.53	48.75	92.86	80.19	0.64	0.93
Q14	BalancedRandomForestClassifier	79.92	78.7	42.25	76.92	80.48	0.55	0.91
Q15	BalancedRandomForestClassifier	13.65	50.23	13.31	100	0.46	0.23	0.9
Q16	BalancedRandomForestClassifier	97.59	97.66	96.04	97.98	97.33	0.97	0.98
Q17	BalancedRandomForestClassifier	92.77	94.08	81.82	97.3	90.86	0.89	0.98
Q18	BalancedRandomForestClassifier	96.79	97.24	92.78	98.9	95.57	0.96	0.99
Q19	BalancedRandomForestClassifier	94.78	95.86	87.62	100	91.72	0.93	0.98
Q20	BalancedRandomForestClassifier	96.79	97.58	91.3	100	95.15	0.95	0.98
Q21	BalancedRandomForestClassifier	91.97	94.22	79.17	100	88.44	0.88	0.98
Q22	BalancedRandomForestClassifier	81.12	87.16	58.41	100	74.32	0.74	0.95
Q23	BalancedRandomForestClassifier	96.39	96.39	93.02	96.39	96.39	0.95	0.98
Q24	BalancedRandomForestClassifier	88.35	89.77	74.74	93.42	86.13	0.83	0.93
Q25	BalancedRandomForestClassifier	93.98	93.82	90.22	93.26	94.38	0.92	0.98
Q26	BalancedRandomForestClassifier	89.16	90.87	64.71	93.62	88.12	0.77	0.95
Q27	BalancedRandomForestClassifier	90.36	92.94	76.7	100	85.88	0.87	0.97
Q28	BalancedRandomForestClassifier	97.19	97.54	93.88	98.92	96.15	0.96	0.99
Q29	BalancedRandomForestClassifier	92.77	94.8	80.85	100	89.6	0.89	0.98
Q30	BalancedRandomForestClassifier	87.15	88.28	63.01	90.2	86.36	0.74	0.95
Q31	BalancedRandomForestClassifier	85.94	90.14	63.83	98.36	81.91	0.77	0.94
Q32	BalancedRandomForestClassifier	89.56	91.01	76.6	94.74	87.28	0.85	0.96
Q33	BalancedRandomForestClassifier	95.18	95.49	90.11	96.47	94.51	0.93	0.98
Q34	BalancedRandomForestClassifier	83.53	88.41	61.17	98.44	78.38	0.75	0.92
Q35	BalancedRandomForestClassifier	90.36	91.49	75.31	93.85	89.13	0.84	0.96
Q36	BalancedRandomForestClassifier	92.77	94.55	80	98.55	90.56	0.88	0.97
Q37	BalancedRandomForestClassifier	88.35	91.47	73.15	100	82.94	0.84	0.97
Q38	BalancedRandomForestClassifier	93.57	95	84.76	100	90	0.92	1
Q39	BalancedRandomForestClassifier	79.12	83.15	48.35	89.8	76.5	0.63	0.91
Q40	BalancedRandomForestClassifier	87.95	91.41	68.13	98.41	84.41	0.81	0.95
Q41	BalancedRandomForestClassifier	89.56	92.49	74.51	100	84.97	0.85	0.97
Q42	BalancedRandomForestClassifier	95.58	96.47	87.8	98.63	94.32	0.93	0.98
Q43	BalancedRandomForestClassifier	91.57	93.85	75.31	98.39	89.3	0.85	0.99
Q44	BalancedRandomForestClassifier	93.98	94.31	88.3	95.4	93.21	0.92	0.98
Q45	BalancedRandomForestClassifier	91.16	92.05	77.5	93.94	90.16	0.85	0.97
Q46	BalancedRandomForestClassifier	82.73	87.72	52.33	95.74	79.7	0.68	0.93
Q47	BalancedRandomForestClassifier	93.17	93.68	86.17	95.29	92.07	0.91	0.97
Q48	BalancedRandomForestClassifier	93.98	95.37	85.29	100	90.74	0.92	0.98
Q49	BalancedRandomForestClassifier	87.15	91.06	68.63	100	82.12	0.81	0.97
Q50	BalancedRandomForestClassifier	75.5	81.67	21.33	88.89	74.46	0.34	0.86

Table 32: Phase 3 - Cost-Sensitive Binary Classification Results V1.3

(*Samples with Informative Features*)

Question	Model	Accuracy (%)	Balanced Accuracy (%)	Precision (%)	Recall (%)	Specificity (%)	F1	ROC AUC
Q01	SVM	88.76	92.75	66.67	100	85.49	0.8	0.99
Q02	ExtraTreesClassifier	98.39	98.65	96.19	100	97.3	0.98	0.99
Q03	DecisionTreeClassifier	92.77	94.06	82.22	97.37	90.75	0.89	0.97
Q04	DecisionTreeClassifier	67.87	74.78	20.83	83.33	66.22	0.33	0.76
Q05	RandomForestClassifier	97.19	96.69	96.34	95.18	98.19	0.96	0.99
Q06	ExtraTreesClassifier	94.38	93.92	90.36	92.59	95.24	0.91	0.99
Q07	DecisionTreeClassifier	85.14	90.31	61.05	100	80.63	0.76	0.97
Q08	BalancedRandomForestClassifier	82.33	85.05	44.44	88.89	81.22	0.59	0.92
Q09	BalancedRandomForestClassifier	91.16	93.04	75	96.77	89.3	0.85	0.96
Q10	BalancedRandomForestClassifier	83.94	87.7	61.05	95.08	80.32	0.74	0.95
Q11	BalancedRandomForestClassifier	84.34	87.84	53.85	93.33	82.35	0.68	0.94
Q12	BalancedRandomForestClassifier	73.09	77.03	22.22	81.82	72.25	0.35	0.8
Q13	SVM	83.53	84.4	50.7	85.71	83.09	0.64	0.82
Q14	BalancedRandomForestClassifier	79.92	78.7	42.25	76.92	80.48	0.55	0.9
Q15	ExtraTreesClassifier	77.11	82.95	35.71	90.91	75	0.51	0.89
Q16	ExtraTreesClassifier	97.59	97.83	95.15	98.99	96.67	0.97	0.98
Q17	BalancedRandomForestClassifier	92.77	94.08	81.82	97.3	90.86	0.89	0.98
Q18	RandomForestClassifier	96.79	97.24	92.78	98.9	95.57	0.96	1
Q19	RandomForestClassifier	95.98	96.59	91	98.91	94.27	0.95	0.98
Q20	DecisionTreeClassifier	96.79	97.58	91.3	100	95.15	0.95	0.98
Q21	RandomForestClassifier	91.97	94.22	79.17	100	88.44	0.88	0.97
Q22	DecisionTreeClassifier	84.74	85.26	66.28	86.36	84.15	0.75	0.89
Q23	ExtraTreesClassifier	97.19	97.29	94.19	97.59	96.99	0.96	0.98
Q24	ExtraTreesClassifier	88.35	89.41	75.27	92.11	86.71	0.83	0.96
Q25	RandomForestClassifier	95.18	95.75	89.69	97.75	93.75	0.94	0.97
Q26	RandomForestClassifier	89.56	91.12	65.67	93.62	88.61	0.77	0.96
Q27	SVM	90.36	92.6	77.23	98.73	86.47	0.87	0.97
Q28	DecisionTreeClassifier	94.78	94.31	93.48	92.47	96.15	0.93	0.95
Q29	ExtraTreesClassifier	93.17	94.72	82.42	98.68	90.75	0.9	0.98
Q30	BalancedRandomForestClassifier	87.15	88.28	63.01	90.2	86.36	0.74	0.95
Q31	BalancedRandomForestClassifier	85.94	90.14	63.83	98.36	81.91	0.77	0.94
Q32	ExtraTreesClassifier	90.36	91.96	77.66	96.05	87.86	0.86	0.97
Q33	ExtraTreesClassifier	94.78	95.47	88.3	97.65	93.29	0.93	0.98
Q34	DecisionTreeClassifier	83.53	88.41	61.17	98.44	78.38	0.75	0.92
Q35	ExtraTreesClassifier	89.96	91.22	74.39	93.85	88.59	0.83	0.97
Q36	BalancedRandomForestClassifier	92.77	94.55	80	98.55	90.56	0.88	0.97
Q37	DecisionTreeClassifier	31.73	50	31.73	100	0	0.48	0.89
Q38	ExtraTreesClassifier	93.98	95.31	85.58	100	90.62	0.92	1
Q39	BalancedRandomForestClassifier	79.12	83.15	48.35	89.8	76.5	0.63	0.91
Q40	DecisionTreeClassifier	87.95	90.89	68.54	96.83	84.95	0.8	0.95
Q41	ExtraTreesClassifier	88.35	90.14	74.23	94.74	85.55	0.83	0.96
Q42	ExtraTreesClassifier	95.18	95.39	88.61	95.89	94.89	0.92	0.98
Q43	SVM	92.37	94.92	76.54	100	89.84	0.87	0.95
Q44	ExtraTreesClassifier	93.57	94	87.37	95.4	92.59	0.91	0.98
Q45	BalancedRandomForestClassifier	91.16	92.05	77.5	93.94	90.16	0.85	0.97
Q46	BalancedRandomForestClassifier	82.73	87.72	52.33	95.74	79.7	0.68	0.93
Q47	SVM	94.38	95.73	85.86	100	91.46	0.92	0.98
Q48	DecisionTreeClassifier	94.38	95.41	86.87	98.85	91.98	0.92	0.97
Q49	BalancedRandomForestClassifier	87.15	91.06	68.63	100	82.12	0.81	0.97
Q50	BalancedRandomForestClassifier	88.35	70.67	31.03	50	91.34	0.38	0.81

Table 33: Phase 3 - Cost-Sensitive Multiclass Classification Results V1.3

(*Samples with Informative Features*)

Question	**Model**	**Accuracy (%)**	**Balanced Accuracy (%)**	**Precision-Micro (%)**	**Recall-Micro (%)**	**F1-Micro**
Q01	BalancedRandomForestClassifier	63.83	56.47	63.83	63.83	0.64
Q02	BalancedRandomForestClassifier	46.67	37.73	46.67	46.67	0.47
Q03	SVM	60.19	51.54	60.19	60.19	0.6
Q04	DecisionTreeClassifier	33.01	38.23	33.01	33.01	0.33
Q05	BalancedRandomForestClassifier	3.3	16.67	3.3	3.3	0.03
Q06	RandomForestClassifier	76	57.66	76	76	0.76
Q07	ExtraTreesClassifier	64.52	62.41	64.52	64.52	0.65
Q08	ExtraTreesClassifier	56.31	48.87	56.31	56.31	0.56
Q09	RandomForestClassifier	64.89	55.98	64.89	64.89	0.65
Q10	RandomForestClassifier	54.9	54.77	54.9	54.9	0.55
Q11	DecisionTreeClassifier	53.76	52.72	53.76	53.76	0.54
Q12	BalancedRandomForestClassifier	22.55	45.66	22.55	22.55	0.23
Q13	SVM	50	50.26	50	50	0.5
Q14	ExtraTreesClassifier	50.98	48.56	50.98	50.98	0.51
Q15	RandomForestClassifier	49.46	46.58	49.46	49.46	0.49
Q16	RandomForestClassifier	93.4	54.21	93.4	93.4	0.93
Q17	RandomForestClassifier	79.21	52.81	79.21	79.21	0.79
Q18	RandomForestClassifier	87.38	56.94	87.38	87.38	0.87
Q19	SVM	80.58	37.68	80.58	80.58	0.81
Q20	DecisionTreeClassifier	68.82	29.17	68.82	68.82	0.69
Q21	SVM	72.92	52.01	72.92	72.92	0.73
Q22	SVM	58.65	50.99	58.65	58.65	0.59
Q23	SVM	65.96	37.83	65.96	65.96	0.66
Q24	ExtraTreesClassifier	57.69	54.74	57.69	57.69	0.58
Q25	RandomForestClassifier	82.52	45.93	82.52	82.52	0.83
Q26	ExtraTreesClassifier	54.84	44.48	54.84	54.84	0.55
Q27	ExtraTreesClassifier	71.84	55.9	71.84	71.84	0.72
Q29	DecisionTreeClassifier	53.76	42.31	53.76	53.76	0.54
Q30	ExtraTreesClassifier	56.86	42.48	56.86	56.86	0.57
Q31	RandomForestClassifier	54.81	43.7	54.81	54.81	0.55
Q32	ExtraTreesClassifier	73.53	51.2	73.53	73.53	0.74
Q33	RandomForestClassifier	80.21	60	80.21	80.21	0.8
Q34	SVM	52.94	43.08	52.94	52.94	0.53
Q35	RandomForestClassifier	60.78	41.31	60.78	60.78	0.61
Q36	SVM	60.44	46.74	60.44	60.44	0.6
Q37	SVM	66.99	69.47	66.99	66.99	0.67
Q38	DecisionTreeClassifier	85.29	53.83	85.29	85.29	0.85
Q39	ExtraTreesClassifier	53.4	46.32	53.4	53.4	0.53
Q40	RandomForestClassifier	55.88	42.86	55.88	55.88	0.56
Q41	RandomForestClassifier	69.9	46.46	69.9	69.9	0.7
Q42	RandomForestClassifier	79.79	51.19	79.79	79.79	0.8
Q43	RandomForestClassifier	62.77	40.86	62.77	62.77	0.63
Q44	SVM	74.26	47.89	74.26	74.26	0.74
Q45	ExtraTreesClassifier	68.93	39.87	68.93	68.93	0.69
Q46	ExtraTreesClassifier	55.91	48.56	55.91	55.91	0.56
Q47	ExtraTreesClassifier	73.08	47.8	73.08	73.08	0.73
Q48	ExtraTreesClassifier	71.57	41.06	71.57	71.57	0.72
Q49	SVM	57.84	57.25	57.84	57.84	0.58
Q50	SVM	55.88	42.22	55.88	55.88	0.56

A.2 Unsupervised Learning

K-Means clustering results for Q21-Q30, Q31-Q40 and Q41-Q50 are summarised in tables 34, 35 and 36 respectively.

Table 34: K-Means Clustering Results for Q21-Q30

(*Samples with Informative Features*)

	21-Effort	**Age**	**Reciprocity**	**21-Truthfulness**
Cluster 0	1	30	4	7
Cluster 1	5	34	4	6
	22-Effort	**22-Uncomfortable**	**Age**	**22-Truthfulness**
Cluster 0	5	4	29	5
Cluster 1	2	1	32	7
	23-Effort	**Age**	**Reciprocity**	**23-Truthfulness**
Cluster 0	1	34	3	7
Cluster 1	1	29	5	7
	24-Uncomfortable	**Age**	**Reciprocity**	**24-Truthfulness**
Cluster 0	5	30	4	6
Cluster 1	1	33	4	7
	25-Effort	**Age**	**Reciprocity**	**25-Truthfulness**
Cluster 0	3	27	4	5
Cluster 1	1	31	4	7
	26-Effort	**26-Uncomfortable**	**Age**	**26-Truthfulness**
Cluster 0	3	6	30	5
Cluster 1	2	2	32	6
	27-Effort	**27-Uncomfortable**	**Age**	**27-Truthfulness**
Cluster 0	1	1	32	7
Cluster 1	3	3	33	5
	Age	**Online-Presence**	**Personal-Stability**	**28-Truthfulness**
Cluster 0	27	8	4	7
Cluster 1	38	5	5	7
	29-Effort	**29-Uncomfortable**	**Age**	**29-Truthfulness**
Cluster 0	2	5	29	6
Cluster 1	1	1	32	7
	30-Uncomfortable	**Age**	**Reciprocity**	**30-Truthfulness**
Cluster 0	4	31	4	7
Cluster 1	6	31	4	2

Table 35: K-Means Clustering Results for Q31-Q40

(*Samples with Informative Features*)

	31-Effort	**Age**	**Reciprocity**	**31-Truthfulness**
Cluster 0	6	31	4	6
Cluster 1	2	31	4	6
	32-Effort	**Age**	**Reciprocity**	**32-Truthfulness**
Cluster 0	1	32	4	7
Cluster 1	4	32	4	5
	33-Effort	**33-Uncomfortable**	**IUIPC-Awareness**	**33-Truthfulness**
Cluster 0	1	1	1	7
Cluster 1	3	3	4	6
	34-Effort	**Age**	**Personal-Stability**	**34-Truthfulness**
Cluster 0	5	32	4	6
Cluster 1	2	32	4	7
	35-Effort	**Age**	**Reciprocity**	**35-Truthfulness**
Cluster 0	2	31	4	7
Cluster 1	6	32	4	6
	36-Effort	**Age**	**Reciprocity**	**36-Truthfulness**
Cluster 0	5	32	4	6
Cluster 1	1	32	4	7
	37-Effort	**37-Uncomfortable**	**Reciprocity**	**37-Truthfulness**
Cluster 0	1	1	4	7
Cluster 1	5	3	4	6
	38-Effort	**38-Uncomfortable**	**Reciprocity**	**38-Truthfulness**
Cluster 0	1	1	4	7
Cluster 1	3	5	4	5
	Age	**Personal-Stability**	**Reciprocity**	**39-Truthfulness**
Cluster 0	35	4	4	3
Cluster 1	32	4	4	6
	40-Effort	**Age**	**Reciprocity**	**40-Truthfulness**
Cluster 0	5	30	4	6
Cluster 1	2	34	4	6

Table 36: K-Means Clustering Results for Q41-Q50

(*Samples with Informative Features*)

	41-Effort	**41-Uncomfortable**	**Age**	**41-Truthfulness**
Cluster 0	5	2	31	6
Cluster 1	1	1	31	7
	42-Effort	**42-Uncomfortable**	**Age**	**42-Truthfulness**
Cluster 0	1	1	32	7
Cluster 1	5	2	32	6
	43-Effort	**Age**	**Reciprocity**	**43-Truthfulness**
Cluster 0	2	32	4	7
Cluster 1	5	31	4	6
	44-Effort	**44-Uncomfortable**	**Age**	**44-Truthfulness**
Cluster 0	1	1	30	7
Cluster 1	5	2	34	6
	45-Effort	**45-Uncomfortable**	**Age**	**45-Truthfulness**
Cluster 0	4	2	31	6
Cluster 1	1	1	32	7
	46-Uncomfortable	**Age**	**Reciprocity**	**46-Truthfulness**
Cluster 0	3	32	4	7
Cluster 1	6	32	4	2
	47-Effort	**Age**	**Personal-Stability**	**47-Truthfulness**
Cluster 0	1	32	4	7
Cluster 1	4	31	4	5
	48-Effort	**Age**	**Personal-Stability**	**48-Truthfulness**
Cluster 0	1	47	4	7
Cluster 1	1	27	4	7
	49-Effort	**49-Uncomfortable**	**Age**	**49-Truthfulness**
Cluster 0	1	1	33	7
Cluster 1	3	5	31	6
	50-Uncomfortable	**Age**	**Reciprocity**	**50-Truthfulness**
Cluster 0	6	31	4	1
Cluster 1	5	32	4	6

B Source Code

Please note that most of the source code is repetitive in different versions. It was cumbersome to run the code and comment it repeatedly while working on different methodologies during the progress of the project. Hence, we maintained the separate copies to avoid cluttering the code and be clear with our results. The code for Estimator Component is not included in the Appendix for simplicity because we have 10 different files corresponding to each version. This code is provided in the supplement file.

B.1 Supervised Learning

B.1.1 Cost-Sensitive Classification V1.4

```
#!/usr/bin/env python3
# coding: utf-8

import os
import sys
import numpy as np
import pandas as pd
import re

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer

from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import RandomForestClassifier
from imblearn.ensemble import BalancedRandomForestClassifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict

from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score, balanced_accuracy_score

from sklearn.utils import class_weight

```

```
from imblearn.metrics import geometric_mean_score

import warnings

warnings.filterwarnings('ignore')
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

RESULTS_DIR = 'results'

# Predictor class for Cost-Sensitive Learning with informative features


class TruthfulnessCostSensitivePredictor:

    def __init__(self):
        try:
            print("Starting Truthfulness Predictor COST-SENSITIVE Version 1.4")
            self.user_responses = self.load_survey_data()

            self.user_responses.drop(columns=['Prolific ID'], inplace=True)

            self.reordered_user_responses = self.customise_data()

            self.transformed_user_responses = self.discrete_transform()
            print('Binary Classification Started.')
            self.classification('binary')
            print('Binary Classification Completed.')
            print('Multi-class Classification Started.')
            self.classification('multi-class')
            print('Multi-class Classification Completed.')
        except Exception as e:
            print('Error: ', str(e))
            sys.exit(1)

    def __final__(self):
        print("Ending Truthfulness Predictor COST-SENSITIVE Version 1.4")

    """[This function loads the data from csv file to pandas dataframe]

    Returns:
        [dataframe] -- [data]
    """

    def load_survey_data(self):
        try:
            df = pd.read_csv('All_Responses_Removed.csv')
        except IOError as e:
            print('Problem occured while loading the csv file. Please place the file at the same location along with this script.')
            raise
```

```
89          except Exception as e:
90              print('Problem occured while loading the csv file.')
91              print('Error: ', str(e))
92              raise
93
94          return df
95
96      """[This function customises the columns]
97
98      Returns:
99          [dataframe] -- [data with ordered columns]
100     """
101
102     def customise_data(self):
103
104         data = self.user_responses.copy()
105         data.columns = data.columns.str.replace(r'[\s\n\t ]+', '-')
106         data.columns = data.columns.str.replace(r'[a-d]-', '-')
107
108         demographics_data = data.iloc[:, :8]
109         demographics_data = demographics_data.reindex(
110             sorted(demographics_data.columns), axis=1)
111         question_data = data.reindex(sorted(data.columns[8:]), axis=1)
112         reordered_user_responses = pd.concat(
113             [demographics_data, question_data], axis=1)
114
115         return reordered_user_responses
116
117     """[This function discretises age and online-presence feature data]
118
119     Returns:
120         [dataframe] -- [discretised data]
121     """
122
123     def discrete_transform(self):
124
125         data = self.reordered_user_responses.copy()
126         data.loc[data['Age'] <= 17, 'Age'] = 0
127         data.loc[(data['Age'] > 17) & (data['Age'] <= 24), 'Age'] = 1
128         data.loc[(data['Age'] > 24) & (data['Age'] <= 34), 'Age'] = 2
129         data.loc[(data['Age'] > 34) & (data['Age'] <= 44), 'Age'] = 3
130         data.loc[(data['Age'] > 44) & (data['Age'] <= 54), 'Age'] = 4
131         data.loc[(data['Age'] > 54) & (data['Age'] <= 64), 'Age'] = 5
132         data.loc[data['Age'] > 64, 'Age'] = 6
133
134         data.loc[data['Online-Presence'] <= 5, 'Online-Presence'] = 0
135         data.loc[(data['Online-Presence'] > 5) &
136                  (data['Online-Presence'] <= 10), 'Online-Presence'] = 1
137         data.loc[(data['Online-Presence'] > 10) &
138                  (data['Online-Presence'] <= 15), 'Online-Presence'] = 2
139         data.loc[(data['Online-Presence'] > 15) &
140                  (data['Online-Presence'] <= 20), 'Online-Presence'] = 3
141         data.loc[(data['Online-Presence'] > 20) &
```

```
142                 (data['Online-Presence'] <= 25), 'Online-Presence'] = 4
143
144         return data
145
146     """[This function imputes missing data in any column with the mean
    value]
147
148     Returns:
149         [dataframe] -- [imputed data]
150     """
151
152     def impute_data(self, data):
153         imp = SimpleImputer(missing_values=np.nan, strategy='mean')
154         imputed_data = pd.DataFrame(imp.fit_transform(
155             data), columns=data.columns, index=data.index)
156
157         return imputed_data
158
159     """[This function transforms the data using Standard Scaler and Power
     Transformer object]
160
161     Returns:
162         [dataframe] -- [transformed training data]
163     """
164
165     def train_data_transformation(self, data):
166         scaler = StandardScaler()
167         standard_data = pd.DataFrame(scaler.fit_transform(
168             data), columns=data.columns, index=data.index)
169
170         transformer = PowerTransformer()
171         transformed_data = pd.DataFrame(transformer.fit_transform(
172             standard_data), columns=data.columns, index=data.index)
173
174         return scaler, transformer, transformed_data
175
176     """[This function applies the transformation on the test data]
177
178     Returns:
179         [dataframe] -- [transformed test data]
180     """
181
182     def test_data_transformation(self, train_scaler, train_transformer,
    data):
183         standard_data = pd.DataFrame(train_scaler.fit_transform(
184             data), columns=data.columns, index=data.index)
185
186         transformed_data = pd.DataFrame(train_transformer.fit_transform(
187             standard_data), columns=data.columns, index=data.index)
188
189         return transformed_data
190
191     """[This function transforms the data back to original form]
```

```
192
193     Returns:
194         [dataframe] -- [original data]
195     """
196
197     def data_inverse_transformation(self, scaler_object,
    transformer_object, data):
198         inverse_transformed_data = pd.DataFrame(transformer_object.
    inverse_transform(data),
199                                                 columns=data.columns,
    index=data.index)
200         inverse_scaled_data = pd.DataFrame(scaler_object.
    inverse_transform(inverse_transformed_data),
201                                            columns=data.columns, index=
    data.index)
202
203         return inverse_scaled_data
204
205     """[This function initialises the naive binary classifiers]
206
207     Returns:
208         [list] -- [instances of naive binary classifiers]
209     """
210
211     def get_naive_binary_estimators(self, class_weights):
212         decision_tree_clf = DecisionTreeClassifier(
213             class_weight=class_weights, random_state=42)
214         logistic_regression_clf = LogisticRegression(
215             solver='liblinear', class_weight=class_weights, random_state
    =42)
216         svm_clf = svm.SVC(gamma='auto', probability=True,
217                           class_weight=class_weights, random_state=42)
218         random_forest_clf = RandomForestClassifier(
219             n_estimators=10, class_weight=class_weights, random_state=42)
220         extra_trees_clf = ExtraTreesClassifier(
221             n_estimators=10, class_weight=class_weights, random_state=42)
222         balanced_rf_clf = BalancedRandomForestClassifier(
223             sampling_strategy='not majority', class_weight=class_weights,
     random_state=42)
224
225         naive_estimators = [decision_tree_clf, logistic_regression_clf,
226                             svm_clf, random_forest_clf, extra_trees_clf,
    balanced_rf_clf]
227
228         return naive_estimators
229
230     """[This function initialises the naive multiclass models]
231
232     Returns:
233         [list] -- [instances of naive multiclass models]
234     """
235
236     def get_naive_multi_class_estimators(self, class_weights):
```

```
237         decision_tree_clf = DecisionTreeClassifier(
238             class_weight=class_weights, random_state=42)
239         # Measure using cross-entropy loss
240         multinomial_lr_clf = LogisticRegression(
241             solver='lbfgs', multi_class='multinomial', class_weight=
    class_weights, random_state=42)
242         svm_clf = svm.SVC(gamma='auto', class_weight=class_weights,
243                           probability=True, random_state=42)
244         random_forest_clf = RandomForestClassifier(
245             n_estimators=10, class_weight=class_weights, random_state=42)
246         extra_trees_clf = ExtraTreesClassifier(
247             n_estimators=10, class_weight=class_weights, random_state=42)
248         lr_ovr_clf = LogisticRegression(
249             solver='lbfgs', multi_class='ovr',  class_weight=
    class_weights, random_state=42)
250         balanced_rf_clf = BalancedRandomForestClassifier(
251             sampling_strategy='not majority', class_weight=class_weights,
     random_state=42)
252         multi_class_estimators = [decision_tree_clf, multinomial_lr_clf,
    svm_clf,
253                                   random_forest_clf, extra_trees_clf,
    lr_ovr_clf, balanced_rf_clf]
254 
255         return multi_class_estimators
256 
257     """[This function calculates the cross-validation scores for a
    question]
258 
259     Returns:
260         [dictionary] -- [cross-val scores]
261     """
262 
263     def calc_cross_val_scores(self, clf, data, labels, clf_type, question
    ):
264         scores = {}
265 
266         try:
267             predicted_scores = cross_val_score(
268                 clf, data, labels, cv=10, scoring="accuracy")
269         except Exception as e:
270             raise ValueError(e)
271 
272         predicted_labels = cross_val_predict(clf, data, labels, cv=10)
273 
274         if clf_type == 'binary':
275             tn, fp, fn, tp = confusion_matrix(labels, predicted_labels).
    ravel()
276             specificity = round((tn / (tn + fp)) * 100, 2)
277 
278         predicted_prob = clf.predict_proba(data)
279         predicted_prob_true = [p[1] for p in predicted_prob]
280 
281         scores['Question'] = question
```

```
282         scores['Accuracy'] = round(predicted_scores.mean() * 100, 2)
283         scores['Balanced Accuracy'] = round(
284             balanced_accuracy_score(labels, predicted_labels) * 100, 2)
285         if clf_type == 'binary':
286             scores['Precision'] = round(precision_score(
287                 labels, predicted_labels) * 100, 2)
288             scores['Recall'] = round(recall_score(
289                 labels, predicted_labels) * 100, 2)
290             scores['Specificity'] = specificity
291             scores['F1'] = round(f1_score(labels, predicted_labels), 2)
292             scores['ROC AUC'] = round(
293                 roc_auc_score(labels, predicted_prob_true), 2)
294 
295         if clf_type == 'multi-class':
296             scores['Precision'] = round(precision_score(
297                 labels, predicted_labels, average='micro'), 2)
298             scores['Recall'] = round(recall_score(
299                 labels, predicted_labels, average='micro'), 2)
300             scores['Specificity'] = 'NA'
301             scores['F1'] = round(
302                 f1_score(labels, predicted_labels, average='micro'), 2)
303             scores['ROC AUC'] = 'NA'
304 
305         # print('Confusion Matrix for Q-%s is: ' % (str(question).zfill
    (2)))
306         # print(confusion_matrix(labels, predicted_labels))
307 
308         return scores
309 
310     """[This function trains the naive classifiers]
311 
312     Returns:
313         [list] -- [cross validation scores of all classifiers for a
    question]
314     """
315 
316     def run_naive_classifiers(self, train_x, train_y, class_weights,
    clf_type, question):
317         if clf_type == 'binary':
318             naive_estimators = self.get_naive_binary_estimators(
    class_weights)
319         if clf_type == 'multi-class':
320             naive_estimators = self.get_naive_multi_class_estimators(
321                 class_weights)
322 
323         clf_scores = []
324         for estimator in range(len(naive_estimators)):
325             clf = naive_estimators[estimator]
326             clf.fit(train_x, train_y)
327             try:
328                 scores = self.calc_cross_val_scores(
329                     clf, train_x, train_y, clf_type, question)
330             except ValueError as e:
```

```
331                 raise ValueError(e)
332             clf_scores.append(scores)
333 
334         # print('Perfect Confusion Matrix for Q-%s is: ' % (str(question)
    .zfill(2)))
335         # perfect_labels = train_y
336         # print(confusion_matrix(train_y, perfect_labels))
337 
338         return clf_scores
339 
340     """[This function creates a dataframe to save scores for naive
    classifiers]
341 
342     Returns:
343         [dataframe] -- [score dataframe for naive classifiers]
344     """
345 
346     def create_score_frame(self, scores):
347         dtc_scores, lrc_scores, svm_scores, rfc_scores, etc_scores,
    balanced_rf_scores = [
348             scores[i] for i in range(len(scores))]
349 
350         clf_score_frame = pd.DataFrame({
351             'Question': [dtc_scores['Question'], lrc_scores['Question'],
    svm_scores['Question'],
352                          rfc_scores['Question'], etc_scores['Question'],
    balanced_rf_scores['Question']],
353             'Model': ['Decision Tree', 'Logistic Regression', 'SVM', '
    Random Forest', 'Extra Trees',
354                       'Balanced Random Forest'],
355             'Accuracy': [dtc_scores['Accuracy'], lrc_scores['Accuracy'],
    svm_scores['Accuracy'],
356                          rfc_scores['Accuracy'], etc_scores['Accuracy'],
    balanced_rf_scores['Accuracy']],
357             'Balanced Accuracy': [dtc_scores['Balanced Accuracy'],
    lrc_scores['Balanced Accuracy'],
358                                   svm_scores['Balanced Accuracy'],
359                                   rfc_scores['Balanced Accuracy'],
    etc_scores['Balanced Accuracy'],
360                                   balanced_rf_scores['Balanced Accuracy'
    ]],
361             'Precision': [dtc_scores['Precision'], lrc_scores['Precision'
    ], svm_scores['Precision'],
362                           rfc_scores['Precision'], etc_scores['Precision'
    ], balanced_rf_scores['Precision']],
363             'Recall': [dtc_scores['Recall'], lrc_scores['Recall'],
    svm_scores['Recall'],
364                        rfc_scores['Recall'], etc_scores['Recall'],
    balanced_rf_scores['Recall']],
365             'Specificity': [dtc_scores['Specificity'], lrc_scores['
    Specificity'], svm_scores['Specificity'],
366                             rfc_scores['Specificity'], etc_scores['
    Specificity'], balanced_rf_scores['Specificity']],
```

```
367             'F1': [dtc_scores['F1'], lrc_scores['F1'], svm_scores['F1'],
368                    rfc_scores['F1'], etc_scores['F1'], balanced_rf_scores
    ['F1']],
369             'ROC AUC': [dtc_scores['ROC AUC'], lrc_scores['ROC AUC'],
    svm_scores['ROC AUC'],
370                         rfc_scores['ROC AUC'], etc_scores['ROC AUC'],
    balanced_rf_scores['ROC AUC']]
371         }, columns=['Question', 'Model', 'Accuracy', 'Balanced Accuracy',
     'Precision', 'Recall', 'Specificity', 'F1', 'ROC AUC'])
372 
373         return clf_score_frame
374 
375     """[This function creates a dataframe to save multiclassificaiton
    scores]
376 
377     Returns:
378         [dataframe] -- [score dataframe for naive classifiers]
379     """
380 
381     def create_multi_class_score_frame(self, scores):
382         decision_tree_clf, multinomial_lr_clf, svm_clf, random_forest_clf
    , extra_trees_clf, lr_ovr_clf, balanced_rf_clf = [
383             scores[i] for i in range(len(scores))]
384 
385         clf_score_frame = pd.DataFrame({
386             'Question': [decision_tree_clf['Question'],
    multinomial_lr_clf['Question'], svm_clf['Question'],
387                          random_forest_clf['Question'], extra_trees_clf['
    Question'],
388                          lr_ovr_clf['Question'], balanced_rf_clf['
    Question']],
389             'Model': ['Decision Tree', 'Multinomial Logistic Regression',
     'SVM', 'Random Forest', 'Extra Trees',
390                       'Logistic Regression OVR', 'Balanced Random Forest'
    ],
391             'Accuracy': [decision_tree_clf['Accuracy'],
    multinomial_lr_clf['Accuracy'], svm_clf['Accuracy'],
392                          random_forest_clf['Accuracy'], extra_trees_clf['
    Accuracy'],
393                          lr_ovr_clf['Accuracy'], balanced_rf_clf['
    Accuracy']],
394             'Balanced Accuracy': [decision_tree_clf['Balanced Accuracy'],
     multinomial_lr_clf['Balanced Accuracy'], svm_clf['Balanced Accuracy'
    ],
395                                   random_forest_clf['Balanced Accuracy'],
     extra_trees_clf['Balanced Accuracy'],
396                                   lr_ovr_clf['Balanced Accuracy'],
    balanced_rf_clf['Balanced Accuracy']],
397             'Precision': [decision_tree_clf['Precision'],
    multinomial_lr_clf['Precision'], svm_clf['Precision'],
398                           random_forest_clf['Precision'], extra_trees_clf
    ['Precision'],
```

```
399                         lr_ovr_clf['Precision'], balanced_rf_clf['
    Precision']],
400             'Recall': [decision_tree_clf['Recall'], multinomial_lr_clf['
    Recall'], svm_clf['Recall'],
401                        random_forest_clf['Recall'], extra_trees_clf['
    Recall'],
402                        lr_ovr_clf['Recall'], balanced_rf_clf['Recall']],
403             'Specificity': [decision_tree_clf['Specificity'],
    multinomial_lr_clf['Specificity'], svm_clf['Specificity'],
404                             random_forest_clf['Specificity'],
    extra_trees_clf['Specificity'],
405                             lr_ovr_clf['Specificity'], balanced_rf_clf['
    Specificity']],
406             'F1': [decision_tree_clf['F1'], multinomial_lr_clf['F1'],
    svm_clf['F1'],
407                    random_forest_clf['F1'], extra_trees_clf['F1'],
408                    lr_ovr_clf['F1'], balanced_rf_clf['F1']],
409             'ROC AUC': [decision_tree_clf['ROC AUC'], multinomial_lr_clf[
    'ROC AUC'], svm_clf['ROC AUC'],
410                         random_forest_clf['ROC AUC'], extra_trees_clf['
    ROC AUC'],
411                         lr_ovr_clf['ROC AUC'], balanced_rf_clf['ROC AUC'
    ]],
412         }, columns=['Question', 'Model', 'Accuracy', 'Balanced Accuracy',
     'Precision', 'Recall', 'Specificity', 'F1', 'ROC AUC'])
413 
414         return clf_score_frame
415 
416     """[This function creates a dataframe to save scores for the best
    classifier ]
417 
418     Returns:
419         [dataframe] -- [score dataframe for the best classifier]
420     """
421 
422     def create_score_frame_best_estimator(self, estimator, scores):
423         clf_score_frame = pd.DataFrame({
424             'Question': scores['Question'],
425             'Model': estimator,
426             'Accuracy': scores['Accuracy'],
427             'Balanced Accuracy': scores['Balanced Accuracy'],
428             'Precision': scores['Precision'],
429             'Recall': scores['Recall'],
430             'Specificity': scores['Specificity'],
431             'F1': scores['F1'],
432             'ROC AUC': scores['ROC AUC']
433         }, index=['Question'],
434             columns=['Question', 'Model', 'Accuracy', 'Balanced Accuracy'
    , 'Precision', 'Recall', 'Specificity', 'F1', 'ROC AUC'])
435 
436         return clf_score_frame
437 
438     """[This function makes a call to run naive binary classifiers]
```

```
439
440         Returns:
441             [dataframe] -- [score dataframe for naive binary classifiers]
442         """
443
444         def naive_classification(self, train_x, train_y, class_weights,
        clf_type, question):
445             try:
446                 scores = self.run_naive_classifiers(
447                     train_x, train_y, class_weights, clf_type, question)
448             except ValueError as e:
449                 raise ValueError(e)
450
451             if clf_type == 'binary':
452                 clf_score_frame = self.create_score_frame(scores)
453             if clf_type == 'multi-class':
454                 clf_score_frame = self.create_multi_class_score_frame(scores)
455
456             return clf_score_frame
457
458         """[This function provides the best binary classifiers manually
        retrieved from Estimator Finder component for all 50 questions]
459
460         Returns:
461             [dictionary] -- [instances of best binary classifiers]
462         """
463
464         def get_binary_estimators(self):
465
466             # cv=5 balanced_accuracy
467             binary_estimators = {'Q01': ['SVM', {'C': 0.01, 'class_weight':
        {0: 0.643652561247216, 1: 2.24031007751938}, 'gamma': 'auto', 'kernel'
        : 'rbf'}], 'Q02': ['ExtraTreesClassifier', {'class_weight': {0:
        0.8376811594202899, 1: 1.240343347639485}, 'criterion': 'entropy', '
        max_depth': 2, 'min_samples_leaf': 3, 'min_samples_split': 4, '
        n_estimators': 100}], 'Q03': ['DecisionTreeClassifier', {'class_weight
        ': {0: 0.7225, 1: 1.6235955056179776}, 'max_depth': 2, '
        min_samples_leaf': 1, 'min_samples_split': 2}], 'Q04': ['
        DecisionTreeClassifier', {'class_weight': {0: 0.553639846743295, 1:
        5.160714285714286}, 'max_depth': 2, 'min_samples_leaf': 1, '
        min_samples_split': 2}], 'Q05': ['RandomForestClassifier', {'
        class_weight': {0: 0.7506493506493507, 1: 1.4974093264248705}, '
        criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 2, '
        min_samples_split': 3, 'n_estimators': 300}], 'Q06': ['
        ExtraTreesClassifier', {'class_weight': {0: 0.7391304347826086, 1:
        1.5454545454545454}, 'criterion': 'gini', 'max_depth': 2, '
        min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 300}], '
        Q07': ['DecisionTreeClassifier', {'class_weight': {0:
        0.6509009009009009, 1: 2.156716417910448}, 'max_depth': 2, '
        min_samples_leaf': 1, 'min_samples_split': 2}], 'Q08': ['
        BalancedRandomForestClassifier', {'class_weight': {0:
        0.5838383838383838, 1: 3.4819277108433737}, 'criterion': 'gini', '
        max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 3, '
```

```
n_estimators': 100, 'sampling_strategy': 'majority'}], 'Q09': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.6658986175115207, 1: 2.0069444444444446}, 'criterion': 'entropy', '
max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, '
n_estimators': 300, 'sampling_strategy': 'not majority'}], 'Q10': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.664367816091954, 1: 2.020979020979021}, 'criterion': 'gini', '
max_depth': 4, 'min_samples_leaf': 3, 'min_samples_split': 2, '
n_estimators': 100, 'sampling_strategy': 'not majority'}], 'Q11': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.608421052631579, 1: 2.8058252427184467}, 'criterion': 'entropy', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 4, '
n_estimators': 300, 'sampling_strategy': 'majority'}], 'Q12': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.5494296577946768, 1: 5.5576923076923075}, 'criterion': 'entropy', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 4, '
n_estimators': 300, 'sampling_strategy': 'majority'}], 'Q13': ['SVM',
{'C': 0.01, 'class_weight': {0: 0.5995850622406639, 1:
3.0104166666666665}, 'gamma': 'scale', 'kernel': 'linear'}], 'Q14': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.5946502057613169, 1: 3.141304347826087}, 'criterion': 'gini', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 3, '
n_estimators': 100, 'sampling_strategy': 'not majority'}], 'Q15': ['
ExtraTreesClassifier', {'class_weight': {0: 0.578, 1:
3.7051282051282053}, 'criterion': 'gini', 'max_depth': 2, '
min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 100}], '
Q16': ['ExtraTreesClassifier', {'class_weight': {0:
0.8328530259365994, 1: 1.251082251082251}, 'criterion': 'entropy', '
max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 3, '
n_estimators': 100}], 'Q17': ['BalancedRandomForestClassifier', {'
class_weight': {0: 0.7118226600985221, 1: 1.680232558139535}, '
criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 2, '
min_samples_split': 4, 'n_estimators': 100, 'sampling_strategy': '
majority'}], 'Q18': ['RandomForestClassifier', {'class_weight': {0:
0.7853260869565217, 1: 1.376190476190476}, 'criterion': 'entropy', '
max_depth': 4, 'min_samples_leaf': 3, 'min_samples_split': 2, '
n_estimators': 300}], 'Q19': ['RandomForestClassifier', {'class_weight
': {0: 0.7896174863387978, 1: 1.3632075471698113}, 'criterion': '
entropy', 'max_depth': 6, 'min_samples_leaf': 2, 'min_samples_split':
2, 'n_estimators': 100}], 'Q20': ['DecisionTreeClassifier', {'
class_weight': {0: 0.7545691906005222, 1: 1.4820512820512821}, '
max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2}], 'Q21':
 ['RandomForestClassifier', {'class_weight': {0: 0.71712158808933, 1:
1.6514285714285715}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 100}], '
Q22': ['DecisionTreeClassifier', {'class_weight': {0:
0.6816037735849056, 1: 1.8766233766233766}, 'max_depth': 6, '
min_samples_leaf': 3, 'min_samples_split': 4}], 'Q23': ['
ExtraTreesClassifier', {'class_weight': {0: 0.7526041666666666, 1:
1.4896907216494846}, 'criterion': 'entropy', 'max_depth': 2, '
min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}], '
Q24': ['ExtraTreesClassifier', {'class_weight': {0: 0.71712158808933,
1: 1.6514285714285715}, 'criterion': 'entropy', 'max_depth': 6, '
```

```
min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 300}], '
Q25': ['RandomForestClassifier', {'class_weight': {0:
0.7810810810810811, 1: 1.3894230769230769}, 'criterion': 'entropy', '
max_depth': 4, 'min_samples_leaf': 3, 'min_samples_split': 4, '
n_estimators': 300}], 'Q26': ['RandomForestClassifier', {'class_weight
': {0: 0.6162046908315565, 1: 2.6513761467889907}, 'criterion': '
entropy', 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split':
2, 'n_estimators': 300}], 'Q27': ['SVM', {'C': 0.01, 'class_weight':
{0: 0.7316455696202532, 1: 1.5792349726775956}, 'gamma': 'auto', '
kernel': 'rbf'}], 'Q28': ['DecisionTreeClassifier', {'class_weight':
{0: 0.7983425414364641, 1: 1.337962962962963}, 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 3}], 'Q29': ['
ExtraTreesClassifier', {'class_weight': {0: 0.7206982543640897, 1:
1.6327683615819208}, 'criterion': 'gini', 'max_depth': 4, '
min_samples_leaf': 3, 'min_samples_split': 2, 'n_estimators': 100}], '
Q30': ['BalancedRandomForestClassifier', {'class_weight': {0:
0.6282608695652174, 1: 2.4491525423728815}, 'criterion': 'gini', '
max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 4, '
n_estimators': 300, 'sampling_strategy': 'not majority'}], 'Q31': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.6598173515981736, 1: 2.0642857142857145}, 'criterion': 'gini', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 2, '
n_estimators': 100, 'sampling_strategy': 'majority'}], 'Q32': ['
ExtraTreesClassifier', {'class_weight': {0: 0.7189054726368159, 1:
1.6420454545454546}, 'criterion': 'gini', 'max_depth': 2, '
min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}], '
Q33': ['ExtraTreesClassifier', {'class_weight': {0:
0.7585301837270341, 1: 1.467005076142132}, 'criterion': 'gini', '
max_depth': 4, 'min_samples_leaf': 2, 'min_samples_split': 4, '
n_estimators': 300}], 'Q34': ['DecisionTreeClassifier', {'class_weight
': {0: 0.6736596736596736, 1: 1.9395973154362416}, 'max_depth': 2, '
min_samples_leaf': 1, 'min_samples_split': 2}], 'Q35': ['
ExtraTreesClassifier', {'class_weight': {0: 0.6752336448598131, 1:
1.9266666666666667}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 100}], '
Q36': ['BalancedRandomForestClassifier', {'class_weight': {0:
0.6930455635491607, 1: 1.795031055900621}, 'criterion': 'entropy', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 3, '
n_estimators': 300, 'sampling_strategy': 'auto'}], 'Q37': ['
DecisionTreeClassifier', {'class_weight': {0: 0.7297979797979798, 1:
1.5879120879120878}, 'max_depth': 2, 'min_samples_leaf': 1, '
min_samples_split': 2}], 'Q38': ['ExtraTreesClassifier', {'
class_weight': {0: 0.7768817204301075, 1: 1.4029126213592233}, '
criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 3, '
min_samples_split': 2, 'n_estimators': 300}], 'Q39': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.6241900647948164, 1: 2.5130434782608697}, 'criterion': 'entropy', '
max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, '
n_estimators': 300, 'sampling_strategy': 'not majority'}], 'Q40': ['
DecisionTreeClassifier', {'class_weight': {0: 0.6705336426914154, 1:
1.965986394557823}, 'max_depth': 4, 'min_samples_leaf': 3, '
min_samples_split': 3}], 'Q41': ['ExtraTreesClassifier', {'
class_weight': {0: 0.7225, 1: 1.6235955056179776}, 'criterion': '
```

```
    entropy', 'max_depth': 4, 'min_samples_leaf': 2, 'min_samples_split':
    4, 'n_estimators': 100}], 'Q42': ['ExtraTreesClassifier', {'
    class_weight': {0: 0.706601466992665, 1: 1.7100591715976332}, '
    criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, '
    min_samples_split': 4, 'n_estimators': 100}], 'Q43': ['SVM', {'C':
    0.5, 'class_weight': {0: 0.6658986175115207, 1: 2.0069444444444446}, '
    gamma': 'auto', 'kernel': 'rbf'}], 'Q44': ['ExtraTreesClassifier', {'
    class_weight': {0: 0.76657824933687, 1: 1.4378109452736318}, '
    criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 3, '
    min_samples_split': 3, 'n_estimators': 100}], 'Q45': ['
    BalancedRandomForestClassifier', {'class_weight': {0:
    0.6816037735849056, 1: 1.8766233766233766}, 'criterion': 'entropy', '
    max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2, '
    n_estimators': 300, 'sampling_strategy': 'not majority'}], 'Q46': ['
    BalancedRandomForestClassifier', {'class_weight': {0:
    0.6162046908315565, 1: 2.6513761467889907}, 'criterion': 'entropy', '
    max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 4, '
    n_estimators': 100, 'sampling_strategy': 'auto'}], 'Q47': ['SVM', {'C'
    : 0.01, 'class_weight': {0: 0.7605263157894737, 1:
    1.4595959595959596}, 'gamma': 'scale', 'kernel': 'rbf'}], 'Q48': ['
    DecisionTreeClassifier', {'class_weight': {0: 0.7686170212765957, 1:
    1.4306930693069306}, 'max_depth': 4, 'min_samples_leaf': 3, '
    min_samples_split': 4}], 'Q49': ['BalancedRandomForestClassifier', {'
    class_weight': {0: 0.6930455635491607, 1: 1.795031055900621}, '
    criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'n_estimators': 100, 'sampling_strategy': 'auto
    '}], 'Q50': ['BalancedRandomForestClassifier', {'class_weight': {0:
    0.5381750465549349, 1: 7.048780487804878}, 'criterion': 'entropy', '
    max_depth': 2, 'min_samples_leaf': 2, 'min_samples_split': 3, '
    n_estimators': 300, 'sampling_strategy': 'not majority'}]}
468
469         return binary_estimators
470
471     """[This function provides the best multiclass models manually
    retrieved from Estimator Finder component for all 50 questions]
472
473     Returns:
474         [dictionary] -- [instances of best binary classifiers]
475     """
476
477     def get_multi_class_estimators(self):
478
479         # cv=5 f1_micro
480         multi_class_estimators = {'Q01': ['BalancedRandomForestClassifier
    ', {'class_weight': {0: 1.4183006535947713, 1: 1.954954954954955, 2:
    0.5607235142118863}, 'criterion': 'entropy', 'max_depth': 4, '
    min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 300, '
    sampling_strategy': 'not majority'}], 'Q02': ['
    BalancedRandomForestClassifier', {'class_weight': {0: 27.0, 1:
    11.571428571428571, 2: 0.34763948497854075}, 'criterion': 'entropy', '
    max_depth': 6, 'min_samples_leaf': 3, 'min_samples_split': 4, '
    n_estimators': 300, 'sampling_strategy': 'not majority'}], 'Q03': ['
    BalancedRandomForestClassifier', {'class_weight': {0:
```

```
2.212962962962963, 1: 3.0641025641025643, 2: 0.4500941619585687}, '
criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 3, 'n_estimators': 300, 'sampling_strategy': 'not
majority'}], 'Q04': ['BalancedRandomForestClassifier', {'class_weight'
: {0: 0.4927536231884058, 1: 3.7777777777777777, 2:
1.4166666666666667}, 'criterion': 'gini', 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300, '
sampling_strategy': 'not majority'}], 'Q05': ['
BalancedRandomForestClassifier', {'class_weight': {0:
14.066666666666666, 1: 5.410256410256411, 2: 0.3644214162348877}, '
criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 1, '
min_samples_split': 4, 'n_estimators': 100, 'sampling_strategy': 'not
majority'}], 'Q06': ['BalancedRandomForestClassifier', {'class_weight'
: {0: 5.5476190476190474, 1: 2.5053763440860215, 2:
0.41312056737588654}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 300, '
sampling_strategy': 'not majority'}], 'Q07': ['
BalancedRandomForestClassifier', {'class_weight': {0: 1.5, 1:
2.1176470588235294, 2: 0.5373134328358209}, 'criterion': 'gini', '
max_depth': 4, 'min_samples_leaf': 2, 'min_samples_split': 3, '
n_estimators': 100, 'sampling_strategy': 'not majority'}], 'Q08': ['
ExtraTreesClassifier', {'class_weight': {0: 0.7376543209876543, 1:
1.6597222222222223, 2: 0.9598393574297188}, 'criterion': 'gini', '
max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 4, '
n_estimators': 300}], 'Q09': ['RandomForestClassifier', {'class_weight
': {0: 2.0277777777777777, 1: 1.8717948717948718, 2:
0.5069444444444444}, 'criterion': 'gini', 'max_depth': 2, '
min_samples_leaf': 2, 'min_samples_split': 3, 'n_estimators': 100}], '
Q10': ['RandomForestClassifier', {'class_weight': {0:
1.4303030303030304, 1: 2.017094017094017, 2: 0.5539906103286385}, '
criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 3, 'n_estimators': 100}], 'Q11': ['SVM', {'C':
0.1, 'class_weight': {0: 0.9907407407407407, 1: 1.829059829059829, 2:
0.6925566343042071}, 'gamma': 'scale', 'kernel': 'linear'}], 'Q12': ['
BalancedRandomForestClassifier', {'class_weight': {0:
0.5042735042735043, 1: 2.8095238095238093, 2: 1.5128205128205128}, '
criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 300, 'sampling_strategy': 'not
majority'}], 'Q13': ['SVM', {'C': 1, 'class_weight': {0:
0.8090277777777778, 1: 1.8943089430894309, 2: 0.8090277777777778}, '
gamma': 'scale', 'kernel': 'rbf'}], 'Q14': ['SVM', {'C': 10, '
class_weight': {0: 0.7181818181818181, 1: 2.1944444444444446, 2:
0.8681318681318682}, 'gamma': 'auto', 'kernel': 'linear'}], 'Q15': ['
RandomForestClassifier', {'class_weight': {0: 0.6125356125356125, 1:
3.4126984126984126, 2: 0.9307359307359307}, 'criterion': 'gini', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 3, '
n_estimators': 300}], 'Q16': ['RandomForestClassifier', {'class_weight
': {0: 27.444444444444443, 1: 6.333333333333333, 2:
0.3564213564213564}, 'criterion': 'gini', 'max_depth': 4, '
min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 100}], '
Q17': ['RandomForestClassifier', {'class_weight': {0:
19.416666666666668, 1: 1.3625730994152048, 2: 0.45155038759689925}, '
criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 2, '
```

```
min_samples_split': 2, 'n_estimators': 100}], 'Q18': ['
BalancedRandomForestClassifier', {'class_weight': {0:
19.916666666666668, 1: 3.1866666666666665, 2: 0.37936507936507935}, '
criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 1, '
min_samples_split': 4, 'n_estimators': 100, 'sampling_strategy': 'not
majority'}], 'Q19': ['BalancedRandomForestClassifier', {'class_weight'
: {0: 9.958333333333334, 1: 4.192982456140351, 2: 0.3757861635220126},
 'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, '
min_samples_split': 4, 'n_estimators': 100, 'sampling_strategy': 'not
majority'}], 'Q20': ['BalancedRandomForestClassifier', {'class_weight'
: {0: 18.083333333333332, 1: 4.018518518518518, 2:
0.37094017094017095}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 300, '
sampling_strategy': 'not majority'}], 'Q21': ['
BalancedRandomForestClassifier', {'class_weight': {0: 9.25, 1:
1.8974358974358974, 2: 0.4228571428571429}, 'criterion': 'entropy', '
max_depth': 6, 'min_samples_leaf': 3, 'min_samples_split': 4, '
n_estimators': 300, 'sampling_strategy': 'not majority'}], 'Q22': ['
RandomForestClassifier', {'class_weight': {0: 5.041666666666667, 1:
1.1203703703703705, 2: 0.5238095238095238}, 'criterion': 'entropy', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 2, '
n_estimators': 100}], 'Q23': ['BalancedRandomForestClassifier', {'
class_weight': {0: 24.22222222222222, 1: 3.303030303030303, 2:
0.3765112262521589}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 300, '
sampling_strategy': 'not majority'}], 'Q24': ['
BalancedRandomForestClassifier', {'class_weight': {0:
4.2105263157894735, 1: 1.7391304347826086, 2: 0.45714285714285713}, '
criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 300, 'sampling_strategy': 'not
majority'}], 'Q25': ['BalancedRandomForestClassifier', {'class_weight'
: {0: 19.833333333333332, 1: 2.9382716049382718, 2:
0.3832528180354267}, 'criterion': 'entropy', 'max_depth': 4, '
min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 100, '
sampling_strategy': 'not majority'}], 'Q26': ['ExtraTreesClassifier',
{'class_weight': {0: 2.3225806451612905, 1: 0.9473684210526315, 2:
0.6605504587155964}, 'criterion': 'gini', 'max_depth': 2, '
min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 100}], '
Q27': ['BalancedRandomForestClassifier', {'class_weight': {0:
11.380952380952381, 1: 1.6258503401360545, 2: 0.4353369763205829}, '
criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 300, 'sampling_strategy': 'not
majority'}], 'Q29': ['BalancedRandomForestClassifier', {'class_weight'
: {0: 15.166666666666666, 1: 2.757575757575758, 2:
0.3888888888888889}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 300, '
sampling_strategy': 'not majority'}], 'Q30': ['RandomForestClassifier'
, {'class_weight': {0: 1.0085470085470085, 1: 1.9666666666666666, 2:
0.6666666666666666}, 'criterion': 'gini', 'max_depth': 4, '
min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 100}], '
Q31': ['ExtraTreesClassifier', {'class_weight': {0: 4.016666666666667,
 1: 0.9917695473251029, 2: 0.5738095238095238}, 'criterion': 'gini', '
max_depth': 2, 'min_samples_leaf': 3, 'min_samples_split': 2, '
```

```
n_estimators': 100}], 'Q32': ['BalancedRandomForestClassifier', {'
class_weight': {0: 7.212121212121212, 1: 1.5555555555555556, 2:
0.45075757575757575}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 100, '
sampling_strategy': 'not majority'}], 'Q33': ['
BalancedRandomForestClassifier', {'class_weight': {0:
24.666666666666668, 1: 3.3636363636363638, 2: 0.3756345177664975}, '
criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 100, 'sampling_strategy': 'not
majority'}], 'Q34': ['RandomForestClassifier', {'class_weight': {0:
7.833333333333333, 1: 1.030701754385965, 2: 0.5257270693512305}, '
criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 4, 'n_estimators': 100}], 'Q35': ['
ExtraTreesClassifier', {'class_weight': {0: 7.151515151515151, 1:
1.048888888888889, 2: 0.5244444444444445}, 'criterion': 'gini', '
max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 4, '
n_estimators': 100}], 'Q36': ['BalancedRandomForestClassifier', {'
class_weight': {0: 7.066666666666666, 1: 1.7235772357723578, 2:
0.4389233954451346}, 'criterion': 'gini', 'max_depth': 4, '
min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300, '
sampling_strategy': 'not majority'}], 'Q37': ['RandomForestClassifier'
, {'class_weight': {0: 9.916666666666666, 1: 1.6527777777777777, 2:
0.4358974358974359}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}], '
Q38': ['BalancedRandomForestClassifier', {'class_weight': {0:
19.583333333333332, 1: 3.1333333333333333, 2: 0.3802588996763754}, '
criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 100, 'sampling_strategy': 'not
majority'}], 'Q39': ['RandomForestClassifier', {'class_weight': {0:
3.0641025641025643, 1: 0.8129251700680272, 2: 0.6927536231884058}, '
criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 3, 'n_estimators': 100}], 'Q40': ['
BalancedRandomForestClassifier', {'class_weight': {0:
7.933333333333334, 1: 0.9794238683127572, 2: 0.5396825396825397}, '
criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1, '
min_samples_split': 4, 'n_estimators': 300, 'sampling_strategy': 'not
majority'}], 'Q41': ['RandomForestClassifier', {'class_weight': {0:
19.833333333333332, 1: 1.391812865497076, 2: 0.448210922787194}, '
criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 100}], 'Q42': ['
RandomForestClassifier', {'class_weight': {0: 36.166666666666664, 1:
1.5724637681159421, 2: 0.4280078895463511}, 'criterion': 'gini', '
max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 3, '
n_estimators': 100}], 'Q43': ['BalancedRandomForestClassifier', {'
class_weight': {0: 6.636363636363637, 1: 1.140625, 2:
0.5069444444444444}, 'criterion': 'entropy', 'max_depth': 6, '
min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 100, '
sampling_strategy': 'not majority'}], 'Q44': ['
BalancedRandomForestClassifier', {'class_weight': {0:
12.944444444444445, 1: 2.9871794871794872, 2: 0.3864013266998342}, '
criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
min_samples_split': 2, 'n_estimators': 100, 'sampling_strategy': 'not
majority'}], 'Q45': ['RandomForestClassifier', {'class_weight': {0:
```

```
    26.444444444444443, 1: 0.967479674796748, 2: 0.5185185185185185}, '
    criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 3, '
    min_samples_split': 4, 'n_estimators': 100}], 'Q46': ['
    RandomForestClassifier', {'class_weight': {0: 1.0909090909090908, 1:
    1.7560975609756098, 2: 0.6605504587155964}, 'criterion': 'entropy', '
    max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2, '
    n_estimators': 100}], 'Q47': ['RandomForestClassifier', {'class_weight
    ': {0: 10.0, 1: 2.2857142857142856, 2: 0.40609137055837563}, '
    criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 1, '
    min_samples_split': 3, 'n_estimators': 100}], 'Q48': ['
    BalancedRandomForestClassifier', {'class_weight': {0:
    26.22222222222222, 1: 2.5376344086021505, 2: 0.38943894389438943}, '
    criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'n_estimators': 300, 'sampling_strategy': 'not
    majority'}], 'Q49': ['RandomForestClassifier', {'class_weight': {0:
    6.611111111111111, 1: 1.2395833333333333, 2: 0.4897119341563786}, '
    criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
    min_samples_split': 4, 'n_estimators': 300}], 'Q50': ['
    BalancedRandomForestClassifier', {'class_weight': {0:
    0.4527938342967245, 1: 3.7301587301587302, 2: 1.910569105691057}, '
    criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 1, '
    min_samples_split': 2, 'n_estimators': 100, 'sampling_strategy': 'not
    majority'}]}
481
482         return multi_class_estimators
483
484     """[This function trains and evaluates the best classifiers for all
    questions]
485
486     Returns:
487         [dictionaries] -- [cross-validation and prediction scores]
488     """
489
490     def run_best_estimator(self, train_x, train_y, test_x, test_y,
    estimator, params, clf_type, question):
491         estimator_scores = {}
492
493         if estimator == 'DecisionTreeClassifier':
494             clf = DecisionTreeClassifier(max_depth=params['max_depth'],
    min_samples_split=params['min_samples_split'],
495                                          min_samples_leaf=params['
    min_samples_leaf'], class_weight=params['class_weight'], random_state
    =42)
496         elif estimator == 'LogisticRegression':
497             clf = LogisticRegression(
498                 C=params['C'], solver=params['solver'], class_weight=
    params['class_weight'], random_state=42)
499         elif estimator == 'SVM':
500             clf = svm.SVC(C=params['C'], kernel=params['kernel'],
501                           gamma=params['gamma'], probability=True,
    class_weight=params['class_weight'], random_state=42)
502         elif estimator == 'RandomForestClassifier':
```

```
503             clf = RandomForestClassifier(criterion=params['criterion'],
    n_estimators=params['n_estimators'],
504                                          max_depth=params['max_depth'],
    min_samples_split=params['min_samples_split'],
505                                          min_samples_leaf=params['
    min_samples_leaf'], class_weight=params['class_weight'], random_state
    =42)
506         elif estimator == 'ExtraTreesClassifier':
507             clf = ExtraTreesClassifier(criterion=params['criterion'],
    n_estimators=params['n_estimators'],
508                                        max_depth=params['max_depth'],
    min_samples_split=params['min_samples_split'],
509                                        min_samples_leaf=params['
    min_samples_leaf'], class_weight=params['class_weight'], random_state
    =42)
510         elif estimator == 'BalancedRandomForestClassifier':
511             clf = BalancedRandomForestClassifier(n_estimators=params['
    n_estimators'],
512                                                  max_depth=params['
    max_depth'], sampling_strategy=params['sampling_strategy'],
513                                                  min_samples_leaf=params[
    'min_samples_leaf'], min_samples_split=params['min_samples_split'],
514                                                  class_weight=params['
    class_weight'], random_state=42)
515
516         clf.fit(train_x, train_y)
517         cross_val_scores = self.calc_cross_val_scores(
518             clf, train_x, train_y, clf_type, question)
519
520         predicted_labels = clf.predict(test_x)
521
522         if clf_type == 'binary':
523             tn, fp, fn, tp = confusion_matrix(test_y, predicted_labels).
    ravel()
524             specificity = round((tn / (tn + fp)) * 100, 2)
525
526         predicted_prob = clf.predict_proba(test_x)
527         predicted_prob_true = [p[1] for p in predicted_prob]
528         estimator_scores['Question'] = question
529         estimator_scores['Accuracy'] = round(
530             accuracy_score(test_y, predicted_labels) * 100, 2)
531         estimator_scores['Balanced Accuracy'] = round(
532             balanced_accuracy_score(test_y, predicted_labels) * 100, 2)
533         if clf_type == 'binary':
534             estimator_scores['Precision'] = round(
535                 precision_score(test_y, predicted_labels) * 100, 2)
536             estimator_scores['Recall'] = round(
537                 recall_score(test_y, predicted_labels) * 100, 2)
538             estimator_scores['Specificity'] = specificity
539             estimator_scores['F1'] = round(
540                 f1_score(test_y, predicted_labels), 2)
541             estimator_scores['ROC AUC'] = round(
542                 roc_auc_score(test_y, predicted_prob_true), 2)
```

```
543
544         if clf_type == 'multi-class':
545             estimator_scores['Precision'] = round(
546                 precision_score(test_y, predicted_labels, average='micro'
    ) * 100, 2)
547             estimator_scores['Recall'] = round(
548                 recall_score(test_y, predicted_labels, average='micro') *
     100, 2)
549             estimator_scores['Specificity'] = 'NA'
550             estimator_scores['F1'] = round(
551                 f1_score(test_y, predicted_labels, average='micro'), 2)
552             estimator_scores['ROC AUC'] = 'NA'
553
554         # print('Perfect Confusion Matrix for Q-%s is: ' % (str(question)
    .zfill(2)))
555         # perfect_labels = train_y
556         # print(confusion_matrix(train_y, perfect_labels))
557
558         return cross_val_scores, estimator_scores
559
560     """[This function performs classification]
561     """
562
563     def classification(self, clf_type):
564
565         class_weights = {}
566         important_features = {}
567         relevant_indexes = []
568         computed_weights = []
569         try:
570             demographics_column_indexes = ['Age', 'Gender', 'IUIPC-
    Awareness', 'IUIPC-Collection', 'IUIPC-Control',
571                                            'Online-Presence', 'Personal-
    Stability', 'Reciprocity']
572             relevant_indexes.extend(demographics_column_indexes)
573
574             naive_clf_scores = pd.DataFrame(columns=[
575                                             'Question', 'Model', '
    Accuracy', 'Balanced Accuracy', 'Precision', 'Recall', 'Specificity',
    'F1', 'ROC AUC'])
576             cross_val_scores = pd.DataFrame(columns=[
577                                             'Question', 'Model', '
    Accuracy', 'Balanced Accuracy', 'Precision', 'Recall', 'Specificity',
    'F1', 'ROC AUC'])
578             best_clf_scores = pd.DataFrame(
579                 columns=['Question', 'Model', 'Accuracy', 'Balanced
    Accuracy', 'Precision', 'Recall', 'Specificity', 'F1', 'ROC AUC'])
580
581             data = self.transformed_user_responses.copy()
582             if clf_type == 'binary':
583                 data.iloc[:, 10:207:4] = (data.iloc[:, 10:207:4] == 7.0)
584             if clf_type == 'multi-class':
585                 data.iloc[:, 10:207:4] = data.iloc[:,
```

```
                                                    10:207:4].replace
    ([1.0, 2.0, 3.0], 0)
                data.iloc[:, 10:207:4] = data.iloc[:,
                                                    10:207:4].replace
    ([4.0, 5.0, 6.0], 1)
                data.iloc[:, 10:207:4] = data.iloc[:,
                                                    10:207:4].replace
    ([7.0], 2)

            for question_number in range(1, 51):
                question = 'Q' + str(question_number).zfill(2)
                question_indexes = []
                question_indexes.extend([str(question_number).zfill(2) +
    '-Effort',
                                         str(question_number).zfill(
                    2) + '-Relevance',
                    str(question_number).zfill(
                    2) + '-Uncomfortable',
                    str(question_number).zfill(2) + '-Truthfulness'])
                relevant_indexes.extend(question_indexes)

                question_label = str(question_number).zfill(
                    2) + '-Truthfulness'

                try:
                    if clf_type == 'binary':
                        train_data_question, test_data_question =
    train_test_split(
                            data[relevant_indexes], stratify=data[
    question_label], test_size=0.3,
                            random_state=42)

                    if clf_type == 'multi-class':
                        cleaned_user_responses = data.loc[:,
    relevant_indexes].dropna(
                        )
                        train_data_question, test_data_question =
    train_test_split(cleaned_user_responses,

              stratify=cleaned_user_responses[

                  question_label],

              test_size=0.3, random_state=42)
                except Exception as e:
                    print('Error: ' + str(e))
                    print('Record skipped.')
                    del relevant_indexes[8:]
                    continue

                train_x_question = train_data_question.copy()
                test_x_question = test_data_question.copy()

```

```
628                 if clf_type == 'binary':
629                     train_x_question = self.impute_data(train_x_question)
630                     test_x_question = self.impute_data(test_x_question)
631 
632                     computed_weights = class_weight.compute_class_weight(
    'balanced', np.unique(
633                         train_x_question[question_label]),
    train_x_question[question_label]).tolist()
634                     class_weights[question] = {
635                         0: computed_weights[0], 1: computed_weights[1]}
636 
637                 try:
638                     if clf_type == 'multi-class':
639                         computed_weights = class_weight.
    compute_class_weight('balanced', np.unique(
640                             train_x_question[question_label]),
    train_x_question[question_label]).tolist()
641                         class_weights[question] = {
642                             0: computed_weights[0], 1: computed_weights
    [1], 2: computed_weights[2]}
643                 except Exception as e:
644                     print("Error: " + str(e))
645                     print('Record skipped.')
646                     del relevant_indexes[8:]
647                     continue
648 
649                 train_y_question = train_x_question.loc[:, question_label
    ]
650                 test_y_question = test_x_question.loc[:, question_label]
651                 train_x_question.drop(columns=question_label, inplace=
    True)
652                 test_x_question.drop(columns=question_label, inplace=True
    )
653 
654                 train_scaler_question, train_transformer_question,
    transformed_train_x_question = \
655                     self.train_data_transformation(train_x_question)
656 
657                 transformed_test_x_question = \
658                     self.test_data_transformation(
659                         train_scaler_question, train_transformer_question
    , test_x_question)
660                 # Feature Selection
661                 rf = RandomForestClassifier(n_estimators=300,
    random_state=42)
662                 rf.fit(transformed_train_x_question, train_y_question)
663 
664                 importances = rf.feature_importances_
665                 indices = np.argsort(importances)[::-1]
666 
667                 important_features[question] = indices[0:3].tolist()
668 
```

```
669                 featured_train_data = transformed_train_x_question.iloc
    [:,
670
    important_features[question]]
671                 featured_test_data = transformed_test_x_question.iloc[:,
672
    important_features[question]]
673
674                 print(clf_type + ' Classification for ' + question + '
    Started.')
675                 try:
676                     clf_score_frame = self.naive_classification(
    featured_train_data, train_y_question,
677
    class_weights[question], clf_type, question)
678                 except ValueError as e:
679                     print("Value Error: " + str(e))
680                     print('Record skipped.')
681                     del relevant_indexes[8:]
682                     continue
683                 naive_clf_scores = naive_clf_scores.append(
    clf_score_frame)
684
685                 if clf_type == 'binary':
686                     binary_estimators = self.get_binary_estimators()
687                     validated_estimator = binary_estimators[question]
688                 if clf_type == 'multi-class':
689                     multi_class_estimators = self.
    get_multi_class_estimators()
690                     validated_estimator = multi_class_estimators[question
    ]
691
692                 best_estimator = validated_estimator[0]
693                 best_estimator_params = validated_estimator[1]
694                 try:
695                     best_clf_cross_val_scores, best_estimator_scores =
    self.run_best_estimator(featured_train_data, train_y_question,
696
                             featured_test_data, test_y_question,
697
                             best_estimator, best_estimator_params, clf_type,
     question)
698                 except ValueError as e:
699                     print("Value Error: " + str(e))
700                     print('Record skipped.')
701                     del relevant_indexes[8:]
702                     continue
703                 print(clf_type + ' Classification for ' +
704                       question + ' Completed.')
705
706                 cross_val_frame = self.create_score_frame_best_estimator(
707                     best_estimator, best_clf_cross_val_scores)
708
```

```
709                 cross_val_scores = cross_val_scores.append(
    cross_val_frame)
710
711                 best_clf_frame = self.create_score_frame_best_estimator(
712                     best_estimator, best_estimator_scores)
713
714                 best_clf_scores = best_clf_scores.append(best_clf_frame)
715
716                 del relevant_indexes[8:]
717
718         except Exception as e:
719             print('Problem occured during classification.')
720             raise
721
722         naive_clf_scores = naive_clf_scores.set_index(['Question', 'Model
    '])
723         cross_val_scores = cross_val_scores.set_index(['Question'])
724         best_clf_scores = best_clf_scores.set_index(['Question'])
725
726         if clf_type == 'multi-class':
727             naive_clf_scores = naive_clf_scores.drop(
728                 columns=['Specificity', 'ROC AUC'])
729             cross_val_scores = cross_val_scores.drop(
730                 columns=['Specificity', 'ROC AUC'])
731             best_clf_scores = best_clf_scores.drop(
732                 columns=['Specificity', 'ROC AUC'])
733
734         if not os.path.isdir(RESULTS_DIR):
735             os.makedirs(RESULTS_DIR)
736
737         naive_clf_file_path = os.path.join(
738             RESULTS_DIR, clf_type + '
    _naive_cost_sensitive_clf_cross_val_scores_1.4.xlsx')
739         best_cross_val_file_path = os.path.join(
740             RESULTS_DIR, clf_type + '
    _best_clf_cost_sensitive_cross_val_scores_1.4.xlsx')
741         best_clf_file_path = os.path.join(
742             RESULTS_DIR, clf_type + '
    _best_clf_cost_sensitive_accuracy_scores_1.4.xlsx')
743
744         # Write results to the EXCEL files
745         try:
746             if ~naive_clf_scores.empty:
747                 naive_clf_scores.to_excel(naive_clf_file_path)
748             if ~cross_val_scores.empty:
749                 cross_val_scores.to_excel(best_cross_val_file_path)
750             if ~best_clf_scores.empty:
751                 best_clf_scores.to_excel(best_clf_file_path)
752         except Exception as e:
753             print('Error: ', str(e))
754             print('Problem occured while writing to the EXCEL file.')
755             raise
756
```

```
757
758 predictor = TruthfulnessCostSensitivePredictor()
```

Listing 1: Cost-Sensitive Classification V1.4

B.2 Unsupervised Learning

B.2.1 K-Means Clustering

```
1  #!/usr/bin/env python
2  # coding: utf-8
3
4  import os
5  import numpy as np
6  import pandas as pd
7  import matplotlib.pyplot as plt
8  import re
9  import seaborn as sns
10
11 from sklearn.model_selection import train_test_split
12 from sklearn.impute import SimpleImputer
13
14 from sklearn.preprocessing import MinMaxScaler
15 from sklearn.preprocessing import StandardScaler
16 from sklearn.preprocessing import PowerTransformer
17
18 from sklearn.ensemble import RandomForestClassifier
19
20 from IPython.display import display
21
22 from sklearn.cluster import KMeans
23 from sklearn.metrics import silhouette_score
24
25 from sklearn import metrics
26
27 from imblearn.ensemble import BalancedRandomForestClassifier
28 import xlsxwriter
29
30
31 """[This function loads the data from csv file to pandas dataframe]
32
33     Returns:
34         [dataframe] -- [data]
35 """
36
37
38 def load_survey_data():
39     df = pd.read_csv('All_Responses_Removed.csv')
40     return df
41
42
43 user_responses = load_survey_data()
44
45 user_responses.drop(columns=['Prolific ID'], inplace=True)
```

```
46
47 user_responses.columns = user_responses.columns.str.replace(r'[\s\n\t ]+'
       , '-')
48 user_responses.columns = user_responses.columns.str.replace(r'[a-d]-', '-
       ')
49 demographics_data = user_responses.iloc[:, :8]
50 demographics_user_responses = demographics_data.reindex(
51     sorted(demographics_data.columns), axis=1)
52 question_subset = user_responses.reindex(
53     sorted(user_responses.columns[8:]), axis=1)
54 reordered_user_responses = pd.concat(
55     [demographics_user_responses, question_subset], axis=1)
56
57 relevant_indexes = []
58 demographics_column_indexes = ['Age', 'Gender', 'IUIPC-Awareness', 'IUIPC
       -Collection', 'IUIPC-Control',
59                                'Online-Presence', 'Personal-Stability', '
       Reciprocity']
60
61 relevant_indexes.extend(demographics_column_indexes)
62
63 """[This function imputes missing data in any column with the mean value]
64
65     Returns:
66         [dataframe] -- [imputed data]
67 """
68
69
70 def impute_data(data):
71     imp = SimpleImputer(missing_values=np.nan, strategy='mean')
72     imputed_data = pd.DataFrame(imp.fit_transform(
73         data), columns=data.columns, index=data.index)
74
75     return imputed_data
76
77
78 """[This function transforms the data using MinMaxScaler]
79
80     Returns:
81         [dataframe] -- [transformed training data]
82 """
83
84
85 def data_scale(data):
86     scaler = MinMaxScaler()
87     standard_data = pd.DataFrame(scaler.fit_transform(
88         data), columns=data.columns, index=data.index)
89
90     return scaler, standard_data
91
92
93 """[This function transforms the data back to original form]
94
```

```
95      Returns:
96          [dataframe] -- [original data]
97  """
98
99
100 def data_inverse_scale(scaler_object, data):
101     inverse_scaled_data = scaler_object.inverse_transform(data)
102
103     return inverse_scaled_data
104
105
106 range_n_clusters = list(range(2, 11))
107 results = {}
108 clusters = {}
109 important_features = {}
110 cluster_segments = {}
111
112 i = 0
113
114 writer = pd.ExcelWriter('K-Means_Clustering_Results.xlsx', engine='
        xlsxwriter')
115 workbook = writer.book
116 print('Running K-Means Clustering Algorithm.')
117 for question_number in range(1, 51):
118
119     question = 'Q' + str(question_number).zfill(2)
120
121     columns = []
122     columns.extend([str(question_number).zfill(2) + '-Effort',
123                     str(question_number).zfill(2) + '-Relevance',
124                     str(question_number).zfill(2) + '-Uncomfortable',
125                     str(question_number).zfill(2) + '-Truthfulness'])
126     relevant_indexes.extend(columns)
127     question_label = str(question_number).zfill(2) + '-Truthfulness'
128     cleaned_reordered_user_responses = pd.DataFrame(
129         reordered_user_responses[relevant_indexes].dropna()).reset_index(
        drop=True)
130
131     train_x_question = cleaned_reordered_user_responses.copy()
132     train_y_question = train_x_question.loc[:, question_label]
133     train_x_question.drop(columns=question_label, inplace=True)
134
135     scaler, good_data = data_scale(train_x_question)
136     # Feature Selection
137     rf = RandomForestClassifier(n_estimators=300, random_state=42)
138     rf.fit(good_data, train_y_question)
139     importances = rf.feature_importances_
140     indices = np.argsort(importances)[::-1]
141
142     important_features[question] = indices[0:3].tolist()
143     good_data = pd.DataFrame(data_inverse_scale(
144         scaler, good_data), index=good_data.index, columns=good_data.
        columns)
```

```
145
146     featured_train_data = good_data.iloc[:, important_features[question]]
147
148     features = [relevant_indexes[i] for i in indices[0:3].tolist()]
149     features = sorted(features)
150     features.append(str(question_number).zfill(2) + '-Truthfulness')
151
152     featured_train_data = featured_train_data.sort_index(axis=1)
153
154     good_data = pd.concat([featured_train_data, train_y_question], axis
    =1)
155
156     scaler, good_data = data_scale(good_data)
157     # print('Running K-Means for ' + question + '.')
158     clusterer = KMeans(n_clusters=2).fit(good_data)
159     preds = clusterer.predict(good_data)
160
161     centers = clusterer.cluster_centers_
162     labels = clusterer.labels_
163
164     true_centers = data_inverse_scale(scaler, centers)
165
166     good_data['labels'] = list(map(int, labels))
167
168     score = metrics.silhouette_score(good_data, preds)
169     # print("For n_clusters = {}. The average silhouette_score with
    Kmeans is : {}".format(2, score))
170
171     segments = ['Cluster {}'.format(i) for i in range(0, len(centers))]
172     true_centers = pd.DataFrame(np.round(true_centers), columns=features)
173     true_centers.index = segments
174     # display(true_centers)
175
176     cluster_segments[question] = true_centers
177
178     true_centers.to_excel(writer, sheet_name='Sheet1', startrow=i,
    startcol=0)
179
180     i = i+4
181     del relevant_indexes[8:]
182 writer.save()
183 print('Saving Results to the current directory.')
```

Listing 2: K-Means Clustering

B.3 Probabilistic Reasoning

B.3.1 Optimal Bayesian Network for Question Blocks

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 import os
5 import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import re
from pomegranate import BayesianNetwork
import pygraphviz
from joblib import dump, load

IMAGES_DIR = 'images'

BAYESIAN_DIR = 'images/bayesian'


"""[This function saves the images to the hard disk]

Returns:
    [png] -- [png file]
"""


def save_fig(folder, fig_id, tight_layout=True):
    if not os.path.isdir(folder):
        os.makedirs(folder)
    file_path = os.path.join(folder, fig_id + '.png')
    if tight_layout:
        plt.tight_layout()
    plt.savefig(file_path, format='png', dpi=300)


"""[This function loads the data from csv file to pandas dataframe]

    Returns:
        [dataframe] -- [data]
"""


def load_survey_data():
    df = pd.read_csv('All_Responses_Removed.csv')
    return df


user_responses = load_survey_data()

user_responses.drop(columns=['Prolific ID'], inplace=True)
user_responses.columns = user_responses.columns.str.replace(r'[\s\n\t ]+'
    , '-')
user_responses.columns = user_responses.columns.str.replace(r'[a-d]-', '-
    ')
demographics_data = user_responses.iloc[:, :8]
demographics_user_responses = demographics_data.reindex(
    sorted(demographics_data.columns), axis=1)
question_subset = user_responses.reindex(
    sorted(user_responses.columns[8:]), axis=1)
reordered_user_responses = pd.concat(
```

```
57      [demographics_user_responses, question_subset], axis=1)
58
59  blocks = {}
60  block_1 = [3, 4, 8, 12, 16, 17, 22, 25, 28, 30, 34, 37, 38, 40, 44, 48,
        49]
61  block_2 = [2, 6, 10, 13, 14, 18, 19, 24, 27, 31, 32, 35, 39, 41, 45, 47,
        50]
62  block_3 = [1, 5, 7, 9, 11, 15, 20, 21, 23, 26, 29, 33, 36, 42, 43, 46]
63  block_1_labels = []
64  for i in range(len(block_1)):
65      block_1_labels.append(str(block_1[i]).zfill(2) + '-Truthfulness')
66  block_2_labels = []
67  for i in range(len(block_2)):
68      block_2_labels.append(str(block_2[i]).zfill(2) + '-Truthfulness')
69  block_3_labels = []
70  for i in range(len(block_3)):
71      block_3_labels.append(str(block_3[i]).zfill(2) + '-Truthfulness')
72  blocks['1'] = block_1_labels
73  blocks['2'] = block_2_labels
74  blocks['3'] = block_3_labels
75
76  fig = plt.figure(figsize=(30, 30), dpi=300)
77  bayes_algorithm = 'exact'
78  bayesian_net_models = {}
79  for block, value in blocks.items():
80
81      subset = reordered_user_responses.loc[:, value]
82
83      if block == '1':
84          column_names = ['DoB', 'Birth Country', 'Home Address', '
        Workplace Travel', 'Ethnicity', 'Politics',
85                          'Memorable Event', 'Study/Education', 'Alcoholic
        Beverages', 'Favourite Mobile Brand',
86                          'Author', 'Government Banned Movie', 'Web Browser
        ', 'Favourite Actor/Actress',
87                          'Last movie at cinema', 'Go to cinema with', '
        Rent adult movies']
88      if block == '2':
89          column_names = ['Gender', 'City of Residence', 'Workplace
        Postcode', 'Personal Email',
90                          'Professional Email', 'Religion', 'Sexual
        Orientation', 'Illnesses',
91                          'Hobby/Pastime', 'Hurt Sentiments - Movie', '
        Holiday Destination', 'Music Genre',
92                          'Age for Adult movie', 'Favourite Movie', 'Money
        on cinema weekly',
93                          'Illegal streaming/downloading', 'Favourite
        Pornstar']
94      if block == '3':
95          column_names = ['Name', 'Country of Residence', 'Home Postcode',
        'Employer Name', 'Work Address',
96                          'Phone Number', 'Relationship Status', 'Lied to
        Partner', 'Languages',
```

```
                        'Annual Income', 'Shared X-rated movies', 'Lied about Age', 'Musician',
                        'Favourite Movie Genre', 'Favourite Soundtrack', 'Online rental subscriptions']
    print('Generating Bayesian Network for Question Block ' + block + '.')
    model = BayesianNetwork.from_samples(
        subset, state_names=column_names, algorithm=bayes_algorithm)
    if block not in bayesian_net_models:
        bayesian_net_models[block] = model

    plt.title('Truthfulness \n Bayesian Network \n' +
              'Block-' + block, fontsize=30, fontweight='bold')
    model.plot(with_labels=True)
    save_fig(BAYESIAN_DIR, bayes_algorithm +
             '_bayesian_net_likert_' + 'block_' + block)
    print('Saving Bayesian Network for Question Block ' +
          block + ' in ' + BAYESIAN_DIR + ' directory.')
```

Listing 3: Optimal Bayesian Network for Question Blocks

B.3.2 Chow-Liu Tree for all Questions

```
#!/usr/bin/env python
# coding: utf-8

import os
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
from pomegranate import BayesianNetwork
import pygraphviz

BAYESIAN_DIR = 'images/bayesian'

"""[This function saves the images to the hard disk]

Returns:
    [png] -- [png file]
"""


def save_fig(folder, fig_id, tight_layout=True):
    if not os.path.isdir(folder):
        os.makedirs(folder)

    file_path = os.path.join(folder, fig_id + '.png')
    if tight_layout:
        plt.tight_layout()
    plt.savefig(file_path, format='png', dpi=300)


"""[This function loads the data from csv file to pandas dataframe]
```

```
32
33      Returns:
34          [dataframe] -- [data]
35  """
36
37
38  def load_survey_data():
39      df = pd.read_csv('All_Responses_Removed.csv')
40      return df
41
42
43  user_responses = load_survey_data()
44  user_responses.drop(columns=['Prolific ID'], inplace=True)
45  user_responses.columns = user_responses.columns.str.replace(r'[\s\n\t ]+'
        , '-')
46  user_responses.columns = user_responses.columns.str.replace(r'[a-d]-', '-
        ')
47  demographics_data = user_responses.iloc[:, :8]
48  demographics_user_responses = demographics_data.reindex(
49      sorted(demographics_data.columns), axis=1)
50  question_subset = user_responses.reindex(
51      sorted(user_responses.columns[8:]), axis=1)
52  reordered_user_responses = pd.concat(
53      [demographics_user_responses, question_subset], axis=1)
54
55  labels = reordered_user_responses.iloc[:, 10:207:4]
56  labels.fillna(labels.mean(), inplace=True)
57  labels = labels.round(0)
58
59  blocks = {}
60  block_1 = [3, 4, 8, 12, 16, 17, 22, 25, 28, 30, 34, 37, 38, 40, 44, 48,
        49]
61  block_2 = [2, 6, 10, 13, 14, 18, 19, 24, 27, 31, 32, 35, 39, 41, 45, 47,
        50]
62  block_3 = [1, 5, 7, 9, 11, 15, 20, 21, 23, 26, 29, 33, 36, 42, 43, 46]
63  column_names_1 = ['DoB', 'Birth Country', 'Home Address', 'Workplace
        Travel', 'Ethnicity', 'Politics',
64                    'Memorable Event', 'Study/Education', 'Alcoholic
        Beverages', 'Favourite Mobile Brand',
65                    'Author', 'Government Banned Movie', 'Web Browser', '
        Favourite Actor/Actress',
66                    'Last movie at cinema', 'Go to cinema with', 'Rent
        adult movies']
67  column_names_2 = ['Gender', 'City of Residence', 'Workplace Postcode', '
        Personal Email',
68                    'Professional Email', 'Religion', 'Sexual Orientation',
         'Illnesses',
69                    'Hobby/Pastime', 'Hurt Sentiments - Movie', 'Holiday
        Destination', 'Music Genre',
70                    'Age for Adult movie', 'Favourite Movie', 'Money on
        cinema weekly',
71                    'Illegal streaming/downloading', 'Favourite Pornstar']
72  column_names_3 = ['Name', 'Country of Residence', 'Home Postcode', '
```

```
    Employer Name', 'Work Address',
                  'Phone Number', 'Relationship Status', 'Lied to Partner
    ', 'Languages',
                  'Annual Income', 'Shared X-rated movies', 'Lied about
    Age', 'Musician',
                  'Favourite Movie Genre', 'Favourite Soundtrack', '
    Online rental subscriptions']

blocks = dict(zip(block_1, column_names_1))
blocks.update(dict(zip(block_2, column_names_2)))
blocks.update(dict(zip(block_3, column_names_3)))
column_names = []
for k, v in sorted(blocks.items()):
    column_names.append(v)

fig = plt.figure(figsize=(30, 30), dpi=300)
bayes_algorithm = 'chow-liu'
bayesian_net_models = {}
print('Generating Chow-Liu Tree for Truthfulness across questions.')
model = BayesianNetwork.from_samples(
    labels, state_names=column_names, algorithm=bayes_algorithm, n_jobs
    =-1)
plt.title('Truthfulness \n Bayesian Network \n',
          fontsize=30, fontweight='bold')
model.plot(with_labels=True)

save_fig(BAYESIAN_DIR, bayes_algorithm + '_full_bayesian_net_likert')
print('Saving Chow-Liu Tree for Truthfulness across questions in ' +
      BAYESIAN_DIR + ' directory.')
```

Listing 4: Chow-Liu Tree for Truthfulness across Questions