

# CA: Internet Ad Prediction

**Module:** B8IT108 Data and Web Mining

**Module Code:** B8IT108

**Student Names:** Nicholas Godwin Udomboso

**Date:** 05/05/2021

**Contents**

[CA: Develop a product release pipeline 1](#_Toc71115807)

[1. Introduction 1](#_Toc71115808)

[**2. Data Balancing** 1](#_Toc71115809)

[**3. Missing Data** 3](#_Toc71115810)

[**4. Data Preparation** 4](#_Toc71115811)

[**5. Implementation** 5](#_Toc71115812)

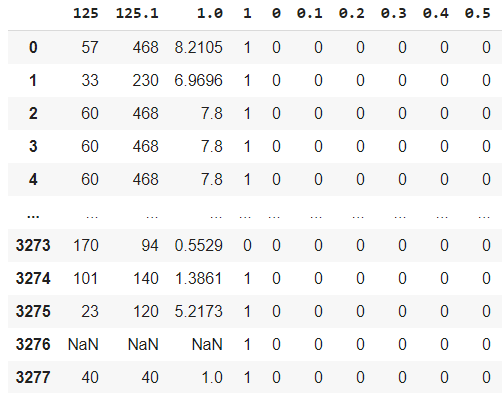
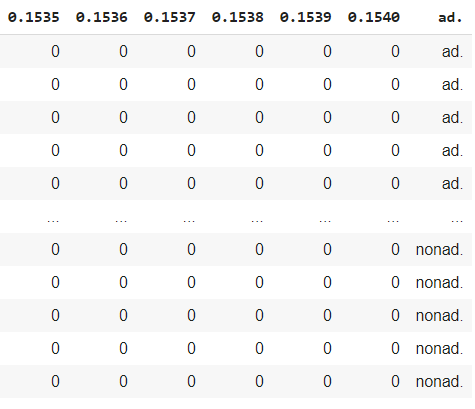
[**6. Evaluation** 6](#_Toc71115813)

[References 9](#_Toc71115814)

# 1. Introduction

The focus of this assessment is to build a model to predict the class of images provided in the dataset named ad.data. The images are to be classified into ad and nonad class.

To begin, it was necessary to read about the dataset in the documentation provided on moodle and to explore the dataset to understand the nature of the data to aid in modelling the data. The dataset is an image data with continuous features in the first three columns of the data. The data is in binary form. I loaded the data to python environment to analyse further. Below is a snapshot of the dataset.

… 

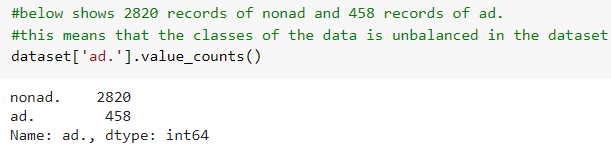
The data contains 3278 records with 1559 variables. 1558 variables are the independent variables, and the dependent variable is the last column of the data. I separated the independent variables and the dependent variables into numpy arrays X and y, respectively.

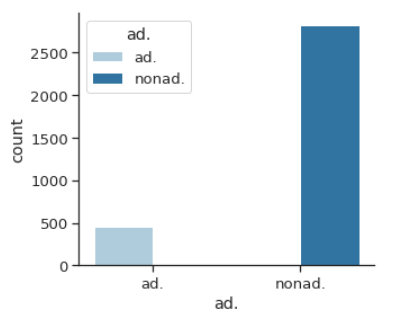
## **2. Data Balancing**

According to Analytics Vidhya (2020), Class imbalance occurs when the dataset does not have equal distribution of class of data in a dataset. In this situation the datapoints of one class is very large when compared to the datapoints of the other class. Class with the major datapoints becomes the majority class and the other the minority class. When this unequally distributed class of data is the class we are interested in to predict, then it is necessary to treat the imbalanced data as most machine learning techniques ignore the imbalanced data, resulting in poor performance of the classifier models ([Brownlee](https://machinelearningmastery.com/author/jasonb/), 2020).

To overcome the challenge, it is common to resample the data (Analytics Vidhya, 2020). Analytics Vidhya (2020) discusses two methods. One method is to duplicate the minority class records which is often referred to as oversampling, and it is commonly done using SMOTE (Synthetic Minority Oversampling Technique). The second method is called undersampling which is to reduce the records of the majority class to achieve balancing the majority with the minority class.

For this assessment the dataset is observed to be unbalanced. Below is the snapshot to show that the dataset has unbalanced categories of data.

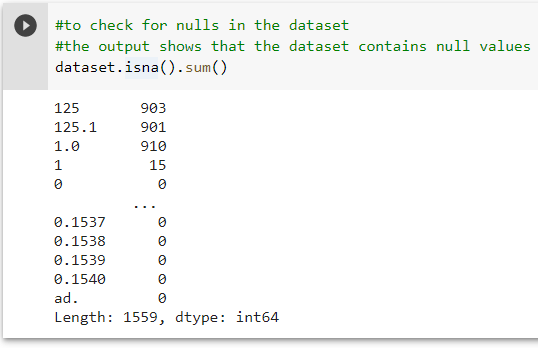




To balance the data, I opted to use SMOTE to balance the data. I chose SMOTE over undersampling because the latter tend to take off records from data that may have essential information. Secondly, I noticed that both methods (oversampling and undersampling) improved the performance of my models almost equally with no major difference in accuracy performance. So, I decided to go for oversampling with SMOTE to avoid losing instances of data through undersampling.

## **3. Missing Data**

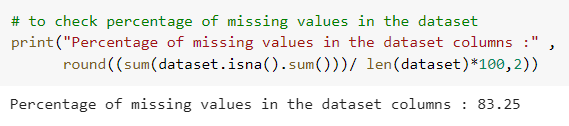
The dataset has missing data. The below snapshot helped to confirm that some variables in the dataset has missing values.



According to Brown *et al* (2003), missing data has effects on the knowledge discovery process of machine learning algorithms. In K-NN algorithm, missing data can lead to assignments of datapoints in wrong clusters. In the case of association rule, this can result in creation of invalid rules. It can generally cause inability of algorithms to discover correct patterns, variance, correlation in data or lead to distortion in data distribution.

In the case of this dataset, I observed high percentage of missing records in the dataset as shown in snapshot below. Kaushik (2020) explained that, ‘if the proportion of missing observations in data is small relative to the total number of observations, we can simply remove those observations. However, this is not the most often case. Deleting the rows containing missing values may lead to parting away with useful information or patterns. So, I chose not to drop them to avoid losing high percentage of information in the dataset. Keeping the records also means that I have to treat the unknown records to avoid having negative impacts on the accuracy performance of classifier models as earlier explained.

Below is the snapshot from my python code that estimates the percentage of missing values in the data.



Below are the steps I used to impute the unknown values in the dataset:

a). I replaced ‘?’ (question mark) character observed in the data with the below command. The command replaced the character with NaN (not a number). This is to indicate that the value is unknown or missing. Below is the command that replaced the ‘?’ (question mark) character.



b). I used sklearn.impute.KNNImputer library to impute the missing values. When comparing other methods to impute the values to the method I used, I found my choice reasonably a good choice in the situation. The columns contributing to the missing values are continuous variable columns as the columns are for height, weight and width. Using the mean to impute would not be appropriate in this scenario as the mean method would not consider any variance between the values. Imputing with the most frequent occurring values is not appropriate either as such imputation works best for categorical variables. According to Brown *et al* (2003), multiple imputation has wide acceptance as it combines several methods of imputation in a single procedure. The technique employs expectation and maximization likelihood to make the imputation. Brown *et al* (2003) explains that KNNImputer class in sklearn compares scenarios that are close to each other and the unique ones, that is, uses likeness from different cases. The method observes neighboring points by finding the distance between them, then estimate the missing values with available values of neighboring datapoints (Kaushik, 2020). It is more accurate than the mean. The drawback is that it is computationally expensive.

### **4. Data Preparation**

This is a process done to understand how the features are involved in the dataset. According to Brownlee (2016), features that are not required can have a negative impact on model performance because the features have a huge influence on what performance that can be achieved. Brownlee (2016) suggested some automatic feature selection techniques like univariate selection, principal component analysis etc. among others. These techniques help in picking the features that would provide the most support in predicting the prediction variable or in the interested outcome. This is because in some cases when feature is reduced, there will be benefits like reducing overfitting, improved accuracy, and reduced training time (Brownlee, 2016).

I used Univariate selection method to check the features that have the strongest relationship with the dependent variable. I chose the method because it is straightforward to use, and I am comfortable with it. I ran the below on python for a univariate test, using the default value k=10 to get the top 10 features having the strongest relationship with the output variable.

I returned the first five rows to display the features chosen by the univariate test. The selected features from the method had the highest scores in their relationship with the dependent variable. These 10 features were tested against my machine learning algorithms to monitor performance of each. I observed that the lesser the features used to implement the classifier models, the less the models performed on the dataset. The more the features the better the models performed. In other words, the models improved with introduction of more features. That eventually led to running the models on all the features of the dataset.

### **5. Implementation**

The following algorithms were the algorithms I implemented and modelled with the dataset. K-nearest neighbour (K-NN), Support Vector Machine (SVM), kernel SVM, Naïve Bayes, Decision Trees, Random Forest, and XGBoost. Performance outcomes of the models based on my observation on the dataset have been provided in a tabular form on this report. Below explains each of the models I implemented briefly.

K-NN:

As Schott (2019) puts it, K-NN is the one of the simplest techniques and preferred by many as it is easy to use and involves low calculation time. The model groups data points with the most similar ones together. It is good at not using assumptions to work on data. Its accuracy however depends on the data quality. It can be poor on where to group datapoints that can go to one group or another. It cannot on its own decide on the optimal value of k (number of nearest neighbour) to model.

SVM sees every data point as a point in a coordinate space of n-dimensions where n is the features of the data. In other words, the datapoints are seen as located in decision boundaries (hyper planes) that can assist to classify or differentiates the datapoints.

For Kernel SVM, the model does not use a linear boundary like SVM but curved. That generally makes Kernel SVM perform better sometimes than SVM.

Naïve Bayes is a probabilistic model and largely assumes that every input variable exists independently. It is also a good model for classification tasks (Chauhan, 2020).

Decision Trees as the name implies follows a tree-like structure to model its decision-making xprocess (Gupta, 2017). This involves choosing the features to make decisions on and then using conditions to split as the decision process grows in a tree-like fashion and knowing when to stop splitting. It is one of the most popular classification algorithms to model data.

Random Forest is a combination of many classification trees (Analytics Vidhya, 2020). It is fast and offers more stability and reliability than a decision tree.

XGBoost is a gradient boosted tree algorithm which helps to predict an output variable by combining estimates of prediction from other models (Amazon).

I implemented all the above methods on the dataset and found support vector machines to be the model best suited for this dataset as it returned the highest accuracy performance of 98.65%. The performance was achieved without any feature reduction. As earlier said, I discovered that the more the dataset features in the model, the better improved the accuracy performance of the model on the dataset. The experience was uniform across all the models mentioned in this report. So, I realized that the models require lots or all the features of the dataset to perform well. Below is a table to provide a summary on how each model performed on the dataset.

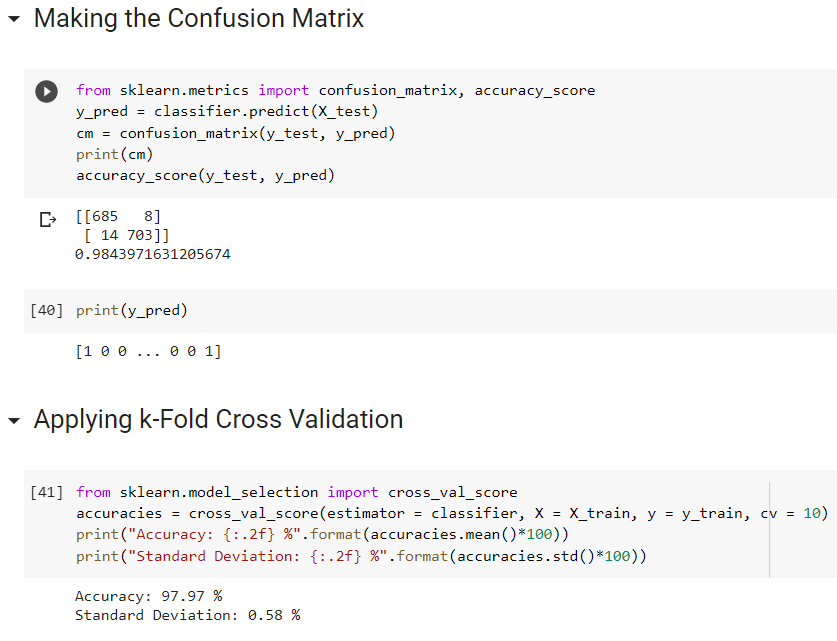
|  |  |  |
| --- | --- | --- |
| **Algorithms** | **Performance before features reduction (%)** | **Performance after features reduction (%)** |
| K-NN | 94.11 | 92.62 |
| SVM | 98.65 | 91.91 |
| Kernel SVM | 80.92 | 74.39 |
| Naïve Bayes | 87.02 | 84.11 |
| Decision Tree | 97.87 | 95.18 |
| Random Forest | 98.23 | 95.67 |
| XGBoost | 98.01 | 93.26 |

In my python code I have all the algorithms I implemented in the code. What I did was to remove comments from the algorithm I am interested in implementing per time to observe the outcome. I did this for each of the algorithms till I was able to record my observations for all the algorithms as each ran.

I ran the code multiple times for each algorithm. This was done to observe the performance of each algorithm when features of the dataset is reduced and when all the dataset features are implemented. They all had best performance when all the features were used. Please refer to the table above to see how the models performed.

### **6. Evaluation**

Below is a snapshot of the outcome of implementing Support Vector Machine (SVM) model on the dataset.



Below is a tabular representation of the confusion matrix obtained in the result above for the model.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Actual Class (Observation) | | |
|  |  | TRUE | FALSE |
| Predicted Class (Expectation) | Positive | 685 | 8 |
| Negative | 14 | 703 |

The table above shows that the SVM model predicts 685 correct images as ad and it is true, 703 images as nonad and it is true, 8 images as ad and it is false, then 14 images as nonad and it is false. Out of all the actual class (685+14), the SVM algorithm predicts 685 true (i.e., true positive) and 14 false (when it should have been true i.e., false negative). Also, out of all the non-actual class, the model predicts 703 false (i.e., true negative) and 8 true (when it should have been false i.e., false positive).

Evaluating the output of the model:

TP(true positive) =685

TN(true negative)=703

FP(false positive)=8

FN(false negative)=14

The model can be evaluated as follows (Narkhede, 2018):

Sensitivity = 685/699 = 0.98

Specificity = 703/711 = 0.99

Precision = 685/693 = 0.99

Recall = 685/699 = 0.98

Accuracy = 1388/1410 = 0.98

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Evaluation Measures** |  |  |  |  |
| **Term** | **Definition** | **Calculation** | **Result** | **Result(%)** |
| Sensitivity | Able to select what should be selected | TP/(TP+FN) | 0.979971388 | 98 |
| Specificity | Able to reject what should be rejected | TN/(TN+FP) | 0.988748242 | 98.87 |
| Precision | Proportion of relevant cases observed | TP/(TP+FP) | 0.988455988 | 98.85 |
| Recall | Proportion of all relevant cases that were discovered | TP/(TP+FN) | 0.979971388 | 98 |
| Accuracy | Aggregate performance value of model classifier | (TP+TN)/(TP+TN+FP+FN) | 0.984397163 | 98.44 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Actual Class (Observation) | | |  | |
|  |  | TRUE | FALSE | class precision | |
| Predicted Class (Expectation) | Positive | 685 | 8 | 0.988455988 | 98.85% |
| Negative | 14 | 703 | 0.980474198 | 98.05% |
|  | class recall | 0.979971388 | 0.988748242 |  |  |
| 98.00% | 98.88% |  |  |

From the above class recall shows that 98.00% of the observations are true, and 98.88% false, which is a good balance. Class precision provides information on percent of predictions that were correct per predicted label, positive or negative. Therefore 98.85% predicted as true were true, and 98.05% predicted as false were true.

In conclusion, the model is able to predict 98 false of a 100, when they are really false and 2 of a 100 as true when they are actually false.

# 

# References:

Waqas et al. (2016) Journal of Computer Science & Systems Biology., ‘Treatment of Missing Values in Data Mining’.

Brown et al. (2003) Industrial Management & Data Systems., ‘Data Mining and the impact of missing data’.

Analytics Vidya. ‘Overcoming Class Imbalance using SMOTE Techniques. Available at: <https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/> (Accessed 1 May 2021).

Brownlee, J. ‘SMOTE for Imbalanced Classification with Python’. Available at: <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification> (Accessed 1 May 2021).

Brownlee, J. (2019) ‘How to Perform Feature Selection with Categorical Data’. Available at: <https://machinelearningmastery.com/feature-selection-with-categorical-data/> (Accessed 3 May 2021).

Brownlee, J. (2016) ‘Feature Selection for Machine Learning in Python’. Available at: <https://machinelearningmastery.com/feature-selection-machine-learning-python/> (Accessed 3 May 2021).

Scikit-learn.org. API Reference. Available at: <https://scikit-learn.org/stable/modules/classes.html> (Accessed 29 April 2021).

Aliyev, V. (2016) ‘3 underrated strategies to deal with Missing Values’. Available at: <https://towardsdatascience.com/3-underrated-strategies-to-deal-with-missing-values-a539fb6c0690> (Accessed 1 May 2021).

Kaushik. (2020) ‘KNNImputer: A robust way to impute missing values (using Scikit-Learn). Available at: <https://www.analyticsvidhya.com/blog/2020/07/knnimputer-a-robust-way-to-impute-missing-values-using-scikit-learn/> (Accessed 3 May 2021).

Amazon ‘How XGBoost Works’. Available at: <https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html> (Accessed 3 May 2021).

Schott, M. (2019) ‘K-Nearest Neighbors (KNN) Algorithm for Machine Learning’. Available at: <https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26> (Accessed 3 May 2021).

Ray, S. (2017) Analytics Vidhya. ‘K-Nearest Neighbors (KNN) Algorithm for Machine Learning’. Available at:  [Understanding Support Vector Machine(SVM) algorithm from examples/](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/) (Accessed 3 May 2021).

Chauhan, N. (2020) KDNuggets. ‘Naïve Bayes Algorithm: Everything you need to know’. Available at: <https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html> (Accessed 3 May 2021).

Gupta, P. (2017) ‘Decision Trees in Machine Learning’. Available at: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052> (Accessed 3 May 2021).

Analytics Vidhya. (2020) ‘Lets Open the Black Box of Random Forests’. Available at: <https://www.analyticsvidhya.com/blog/2020/12/lets-open-the-black-box-of-random-forests/> (Accessed 3 May 2021).

Narkhede, S. (2018) ‘Understanding Confusion Matrix’. Available at: <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62> (Accessed 3 May 2021).

Narkhede, S. (2018) ‘Understanding AUC - ROC Curve. Available at: <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5> (Accessed 3 May 2021).