Introduction

In this report we discuss about the master dataset of the NSW government school locations and student enrolment numbers. The master dataset contains comprehensive information for all government schools in NSW. Data items include school locations, latitude and longitude coordinates, school type, student enrolment numbers, electorate information, contact details and more (1).

For our analysis we have used neural networks, support vector machine and naive bayes methods to predict target variables such as the *Opportunity_class* and *Late_opening_school*.

Data Preprocessing

As the dataset provided contains raw data, it is the best if we process the data accordingly before analyzing it. The dataset is imported in Jupyter Notebook as **data**. Using **data.info()** we can create a table (as shown in Table 1) where we can observe that the dataset has 2216 rows and 44 columns.

```
data = data[~data.isin(["np"]).any(axis=1)]
new_data = data.dropna()
```

Here we can observe that the column **Support_classes** does not have any values in it. Thus we remove this column first using:

```
data = data.drop('Support_classes', axis=1)
```

There are also a lot of rows with **np**, thus we remove any such rows followed by dropping all the rows with null values.

Now we are left with 1385 rows and 43 columns. After having a careful look at the dataset **new_data**, we selected to use *Opportunity_class* and *Late_opening_school* as two independent target variables.

In order to verify which target variable is a better fit to our models, we proprocess the dataset separately twice for each target variable.

For the target variable as *Opportunity_class*, we are using the following as predictors:

- latest_year_enrolment_FTE
- Indigenous_pct
- LBOTE_pct
- ICSEA_value
- Latitude
- Longitude
- Level_of_schooling
- Selective_school
- School_specialty_type
- School_subtype
- Preschool_ind
- Distance_education
- Intensive_english_centre
- School_gender
- Late_opening_school
- ASGS_remoteness

Table 1: data.info()

Columns	Non-Null Count	Dtype
School_code	2216	int64
AgeID	2214	float64
School_name	2216	object
Street	2216	object
Town_suburb	2209	object
Postcode	2215	float64
Phone	2216	object
School_Email	2215	object
Website	2216	object
Fax	2113	object
latest_year_enrolment_FTE	2166	float64
Indigenous_pct	2166	object
LBOTE_pct	2154	object
ICSEA_value	2157	float64
Level_of_schooling	2216	object
Selective_school	2216	object
Opportunity_class	2216	object
School_specialty_type	2216	object
School_subtype	2216	object
Support_classes	0	float64
Preschool_ind	2216	object
Distance_education	2216	object
Intensive_english_centre	2216	object
School_gender	2216	object
Late_opening_school	2216	object
Date_1st_teacher	2215	object
LGA	2216	object
electorate_from_2023	2216	object
electorate_2015_2022	2208	object
Fed_electorate	2208	object
Operational_directorate	2216	object
Principal_network	2216	object
Operational_directorate_office	2215	object
Operational_directorate_office_phone	2215	object
Operational_directorate_office_address	2215	object
FACS_district	2206	object
Local_health_district	2206	object
AECG_region	2192	object
ASGS_remoteness	2216	object
Latitude	2216	float64
Longitude	2216	float64
Assets unit	2178	object
SA4	2215	object
Date_extracted	2216	object

As the columns contains a mixture of numerical and categorical values, we first identify them and then separate them to convert the convert the categorical values into dummies. The columns with the categorical values are:

- Level_of_schooling
- Selective_school
- School_specialty_type
- School_subtype
- Preschool_ind
- Distance_education
- Intensive_english_centre
- School_gender
- Late_opening_school
- ASGS_remoteness

Finally we set X and y as follows:

```
X = pd.concat([data_drop_for_dummies, dummies], axis=1)
y = data_for_preprocessing['Opportunity_class']
```

And for y since **Oppurtunity_class** is categorical, we convert Y and N to 1 and 0 as follows:

```
y = y.replace({'Y': 1, 'N': 0})
```

From the Table 1 we can observe that **Indigenous_pct** and **LBOTE_pct** are of object data type so we convert them into float.

```
X['Indigenous_pct'] = X['Indigenous_pct'].astype(float)
X['LBOTE_pct'] = X['LBOTE_pct'].astype(float)
```

For the purpose of our analysis, we have split our data into 80:20 ratio.

Also, as our dataset has values from different ranges we have scaled and normalised our data using **StandardScaler** as shown below:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Neural Networks

Model 1 : Target Variable - Oppurtunity_class

For the implementation of the neural network we have used the Keras library. The first layer is a dense layer with 64 neurons, the ReLu activation function and the input. The second layer is another dense layer of 32 neurons and a ReLu activation funtion and the final layer is with 1 neuron and a sigmoid function for the binary classification. The code for the following has been shown below.

```
model1 = keras.Sequential([
    keras.layers.Dense(64, activation='relu',
    input_shape=(X_train.shape[1],)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Next we compiled the model using **adam** optimizer for gradient descent and **binary_crossentropy** as the loss funtion. The code for this has been shown below.

```
model1.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

As shown in the code below, we used **X_train** and **y_train** for training and the process is run for 10 times where model's parameters are updated after processing each batch of 32 training samples.

```
model1.fit(X_train, y_train, epochs=10, batch_size=32)
```

Using the code below, we got the following results:
Test Loss: 19.870243966579437 Test Accuracy: 94.22382712364197

```
loss, accuracy = model1.evaluate(X_test, y_test)
print(f"Test Loss: {loss*100}")
print(f"Test Accuracy: {accuracy*100}")
```

Next we computed the confusion matrix as follows:

	Predicted Negative	**Predicted Positive**
Actual Negative	261	0
Actual Positive	16	0

We have also calculated the accuracy, precision, recall and f1-score as follows:

Accuracy: 94.22382671480143
Precision: 0.0
Recall: 0.0
F1-score: 0.0

From the results as mentioned above we can conclude that although the model has an impressive test accuracy of 94.22%, the model is struggling to classify positive results as there are no true positives, precision is zero and recall is zero as well. This could be due to the fact that our model is not a good fit for the chosen target variable.

Thus we decided to change the target variable from Oppurtunity_class to Late_opening_school.

Model 2 : Target Variable - Late_opening_school

For this we preprocessed the data the same way we did previously where we only changed the target variable.

Now we got impressive results as shown below:

Test Loss: 5.506386235356331
Test Accuracy: 97.47292399406433

	Predicted Negative	**Predicted Positive**
Actual Negative	260	2
Actual Positive	3	12

Accuracy: 98.19
Precision: 85.71
Recall: 80.00
F1-score: 82.76

From the results above we can conclude that the model can now predict with target variable 97.47% times accurately. The precision of 85.71% means that when the model predicts a positive outcome, it is correct in the vast majority of cases. The F1-score of 82.76 shows a balance between precision and recall. The recall of 80.00% shows the model's ability to identify most of the actual positive cases.

For a better understanding of our training and testing loss we have plotted a graph as shown in Figure 2. From the graph we can observe that the training loss decreases steadily over the number of epochs, while the testing loss also decreases, but at a slower rate. This suggests that the model is learning from the training data, but it is still overfitting to some extent. The graph also shows that the testing loss starts to plateau after about 8 epochs.

Thus we can conclude that model 2 is a better model as it can predict the target variable better.

We have referred to the following links for building this model:

https://www.youtube.com/watch?v=qFJeN9V1ZsI&ab_channel=freeCodeCamp.org
https://www.youtube.com/watch?v=brB4vwAuqCY&ab_channel=learndataa
https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/
https://www.kaggle.com/code/kumarabhishekone/keras-sequential
https://www.kaggle.com/code/samfiddis/simple-keras-model

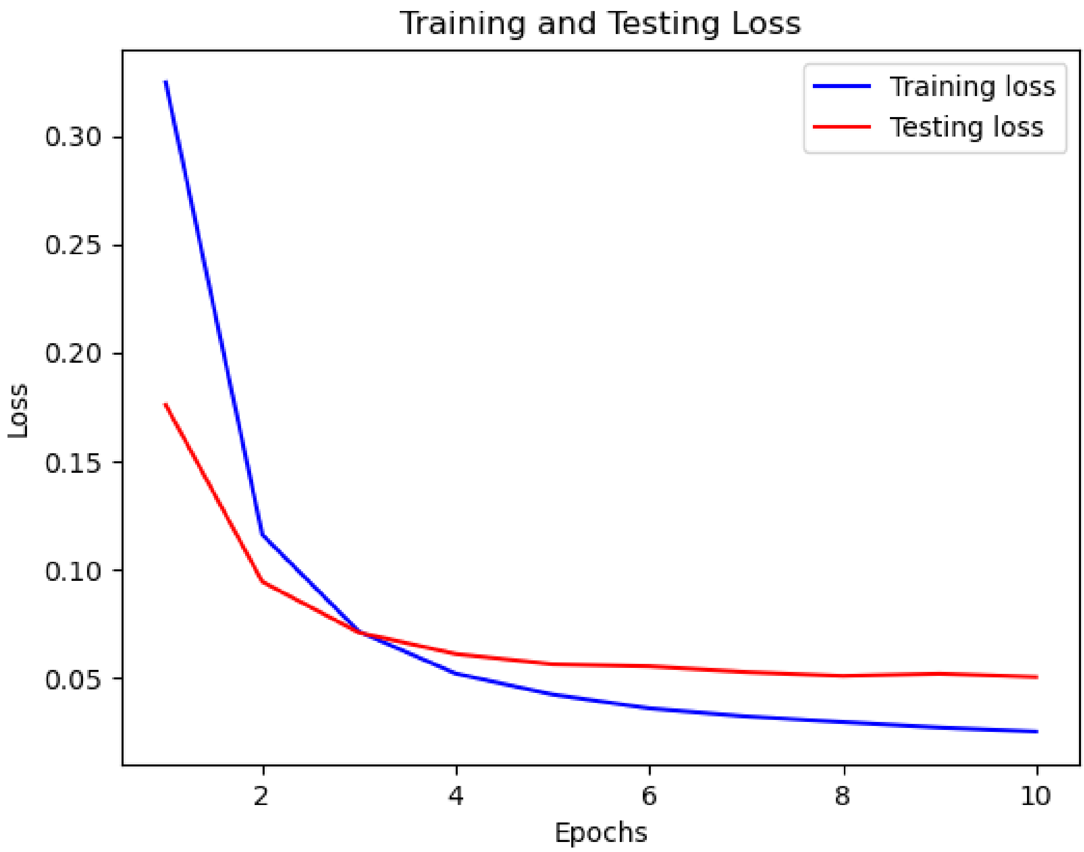


Figure 1: Training and Testing Loss

https://www.kaggle.com/code/pablocastilla/classification-with-keras
https://keras.io/guides/sequential_model/

Support Vector Machines

SVM is a machine learning method used for accurate data classification and predictions. It creates a decision boundary with the objective of maximizing the separation margins between data points belonging to different classes. Before model creation, label encoding was the chosen method for pre-processing to transform categorical variables. This approach was particularly well-suited for predictor variables like "opportunity class," "pre_school," and "intensive English centre" that exhibit binary characteristics, making label encoding a more appropriate choice. Moreover, label encoding also serves the purpose of preserving the ordinal relationship within the data (3). For example, columns like "level of school" and "selective school" exhibit such ordinal characteristics, making label encoding the more fitting choice. As the school dataset used is not linearly separable, various kernels were explored, including the polynomial and radial kernels in addition to the standard linear kernel.

```
label_encoder = LabelEncoder()

label_columns = new_data[['Level_of_schooling', 'Selective_school',
'School_specialty_type', 'School_subtype', 'Preschool_ind',
'Distance_education', 'Intensive_english_centre',
'School_gender', 'ASGS_remoteness', 'Opportunity_class']]
for col in label_columns:
    new_data[col] = label_encoder.fit_transform(new_data[col])
```

To optimize the model performance, hyperparameter tuning was applied to the creation of the linear kernel. This involved experimenting with different cost function values, as well as sampling values for gamma and degree parameters for the other kernels, to achieve the highest accuracy. A range of different cost function values were used and these values were plotted against the accuracy score as shown below. This helps to determine which value provides the optimum model performance. For the linear kernel, using a value of 10 provides the highest accuracy. This is one of the methods of using cross validation to evaluate model performance.

```
svm_clf1 = LinearSVC(C=1, loss="hinge")

svm_clf1.fit(X_train, y_train)

Margins = [0.01, 0.1, 1, 10, 100]
resAccuracy = []
for m in Margins:
    svm_clf_cv = LinearSVC(C=m, loss="hinge")
    svm_clf_cv.fit(X_train, y_train)
    y_pred_cv = svm_clf_cv.predict(X_test)
    accCurr = accuracy_score(y_test, y_pred_cv)*100
    print(f'C = {m}, accuracy = {accCurr}')
```

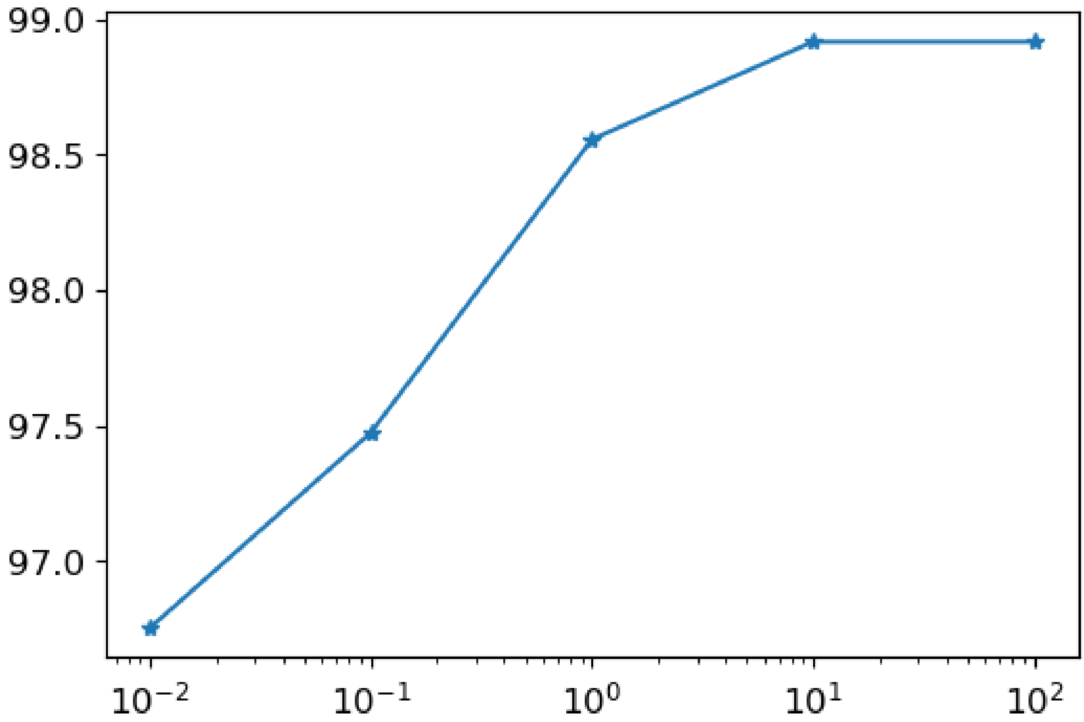


Figure 2: Cross-validation plot of C against accuracy score

```
    resAccuracy.append(accCurr)
plt.figure(figsize=(6,4))
plt.semilogx(Margins, resAccuracy, '*-')
plt.show()
```

C = 0.01, accuracy = 96.75090252707581
C = 0.1, accuracy = 97.47292418772562
C = 1, accuracy = 98.55595667870037
C = 10, accuracy = 98.91696750902527
C = 100, accuracy = 98.91696750902527

By employing the linear kernel, an accuracy score of 98% was achieved. In the case of the polynomial kernel, a pipeline was also utilized to fine-tune the model.

The outcomes were tabularised into a confusion matrix to showcase the model's accurate classification of classes. When employing the pipeline with the polynomial kernel, we attained an accuracy score of 97%. Both the models were very accurate. Besides using the accuracy score, the other metric used was precision score and cross fold mean score. For this dataset, the use of rbf kernel failed to correctly distinguish between correct classes, therefore rbf is not a suitable model for this dataset.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred)*100)
precision_linear = precision_score(y_test, y_pred, average='weighted')
print("Precision Score:", precision_linear*100)
```

98.55595667870037
Precision Score: 98.50862134255635

```
y_pred3 = svm_polynomial.predict(X_test)
confusion_matrix(y_pred3, y_test)
```

98.91696750902527

It's essential to highlight that various train-test split ratios were experimented with, including 70-30, but as the 80-20 split yielded the best results, it was selected for the model. It's worth noting that while label encoding was employed, one-hot encoding was also considered. The accuracy score improved significantly when using label encoding for SVM, therefore this was chosen as the preferred method.

Naive Bayes

Naïve Bayes is a supervised machine learning algorithm based on the probability theory and is called the eager learner (Wang, 2023). It is usually used for classification problems. This method was chosen as the dataset used for this report contained more categorical variables than numerical variables.

As seen in the image below, the dataset was divided into two: 80% for training and 20% for testing. For model building, the GaussianNB classifier was used as it can handle both uint and float values. To train the model with the training data, both the X_train and y_train were passed into the Naïve Bayes classifier model using the model_oc.fit. The accuracy of the training data is 67.60% while the accuracy of the test data is 70.04% which means that the model correctly classified 67.60% of the training data during training and 70.04% of the test data during testing. The predictions made by the model can also be seen below.

```
#data splitting: 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train Dimension: {}".format(X_train.shape))
print("y_train Dimension: {}".format(y_train.shape))
print("X_test Dimension: {}".format(X_test.shape))
print("y_test Dimension: {}".format(y_test.shape))
```

```
X_train Dimension: (1108, 55)
y_train Dimension: (1108,)
X_test Dimension: (277, 55)
y_test Dimension: (277,)
```

```
#Model Building and Training Data
model_los = GaussianNB()
model_los.fit(X_train, y_train)
```

```
GaussianNB()
```

```
#accuracy of the training data
model_los.score(X_train, y_train)
```

```
0.6759927797833934
```

```
#accuracy of the test data
model_los.score(X_test, y_test)
```

```
0.7003610108303249
```

```
#predictions
y_pred_los = model_los.predict(X_test)
y_pred_los
```

```
array([0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0], dtype=int64)
```

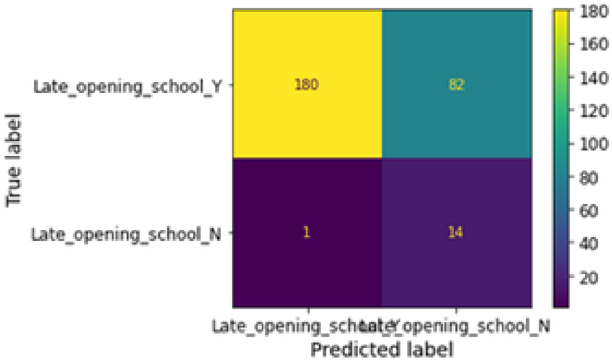
The overall model accuracy score is 70.04%. This means that the model is a good model as it correctly classified the test data for about 70.04%. The model has an F1-score of 61.84% indicating the precision and recall performance of the model. Based on the confusion matrix, the model has 180 True Positives, 14 True Negatives, 82 True Negatives, and 1 False Negatives . This goes to show that the model is relatively good.

```
model_los_acc_score = accuracy_score(y_pred_los, y_test) #Model Accuracy
f1_score_los = f1_score(y_pred_los, y_test, average="weighted") #precision + recall

print("Model Accuracy: ", model_los_acc_score)
print("Model F1 score: ", f1_score_los)
```

```
Model Accuracy:  0.7003610108303249
Model F1 score:  0.6184269037313997
```

```
#confusion matrix
labels = ['Late_opening_school_Y', "Late_opening_school_N"]
cm_los = confusion_matrix(y_test, y_pred_los)
display = ConfusionMatrixDisplay(confusion_matrix=cm_los, display_labels=labels)
display.plot();
```




```
print(classification_report(y_test, y_pred_los))
```

```
              precision    recall  f1-score   support

           0       0.99      0.69      0.81       262
           1       0.15      0.93      0.25        15

    accuracy                           0.70       277
   macro avg       0.57      0.81      0.53       277
weighted avg       0.95      0.70      0.78       277
```

To cross validate the accuracy of the dataset and to understand how the model is performing on the training data, the GaussianNB classifier performed training and evaluated the training data five times. The accuracy scores for each fold were 68.46%, 63.96%, 66.67%, 66.51% and 70.59% respectively which means that the model's accuracy goes up and down. It is not consistent. The average accuracy of the model when added and divided by 5 is 67.24% which means that the model is relatively good but needs more improvement. For the ROC curves, the predicted probabilities were used to plot a Receiver Operating Characteristic Curve and compute the Area Under the Curve.

```
#Cross Validation Accuracy of the data set
cross_val_score(GaussianNB(), X_train, y_train, cv=5)
```

```
array([0.68468468, 0.63963964, 0.66666667, 0.66515837, 0.70588235])
```

```
#ROC Curves
y_pred_prob_los = model_los.predict_proba(X_test)
y_pred_prob_los
```

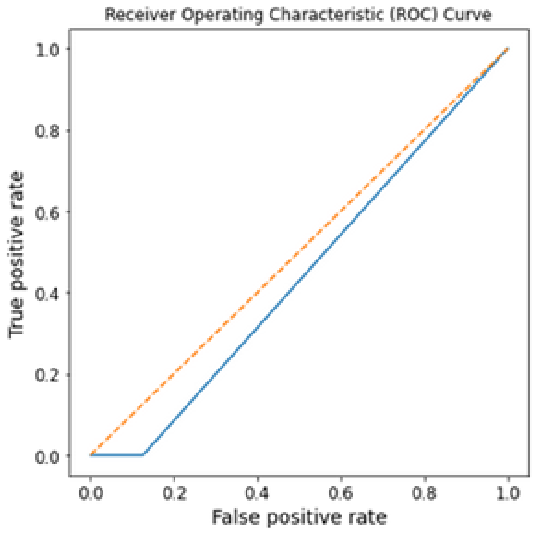
```
array([[1.00000000e+000, 0.00000000e+000],
       [2.84679260e-128, 1.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [2.64368362e-160, 1.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [4.04348448e-128, 1.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [7.81040739e-130, 1.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
       [1.00000000e+000, 0.00000000e+000],
```

```
# class 'Y' as the positives
probs_los = y_pred_prob_los[:,1]
probs_los
```

```
array([0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0.,
       0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0.,
       0., 1., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 1.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0.,
       0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0.,
       1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1.,
       0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 0.,
       1., 0., 0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0.,
       1., 0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1.,
       0., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 1., 0., 0., 1.,
       0., 1., 0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1., 1.,
       0., 0., 1., 0., 0.])
```

The model has an ROC AUC score of 81.02% which means that the model can highly differentiate the Late_opening_school_Y and Late_opening_school_N. The positive rate is higher than the false positive rate. All in all, this is a good model.

```
# use the roc_curve class to calculate the necessary values
fper, tper, thresholds = roc_curve(y_test, probs)
# the plotting
plt.figure(figsize=(6,6))
plt.plot(fper, tper)
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.show()
```



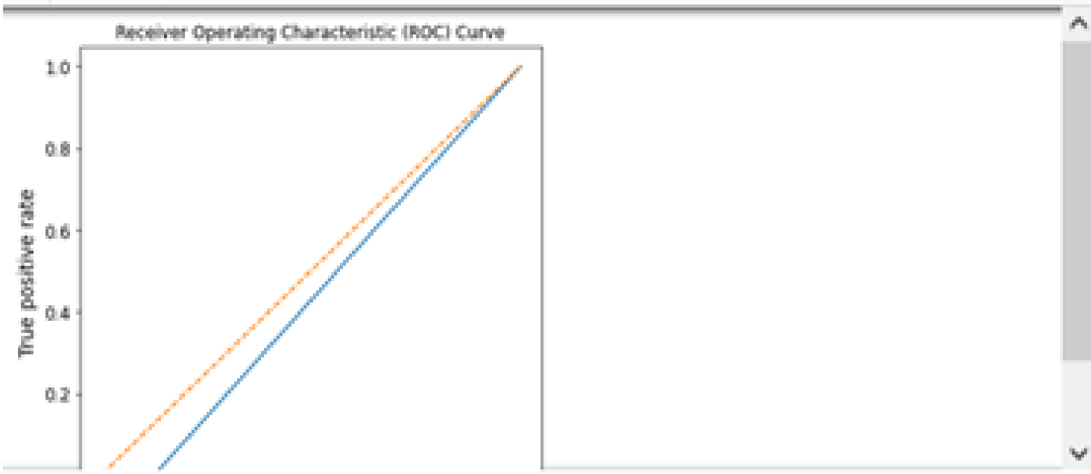

```
# area under the curve
roc_auc_score(y_test, probs_los)
```

```
0.8101781170483461
```

Several target variables were also used to choose the best target variable to be used for the Naïve Bayes model. Opportunity_class, Preschool_ind, Intensive_English_centre, and Late_opening_school were used. Out of the 4 target variables, Intensive_English_centre had the highest roc_auc_score of 94.54% but the 'Late_opening_school' variable was chosen to be the target variable for this model to be consistent with the target variable used for Neural Networks and SVM. The roc_auc_scores of the 4 models using Naïve Bayes with different target variables can be seen below:

Naive Bayes for 'Opportunity Class'

```python
# use the roc_curve class to calculate the necessary values
fper, tper, thresholds = roc_curve(y_test, probs)
# the plotting
plt.figure(figsize=(6,6))
plt.plot(fper, tper)
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.show()
```

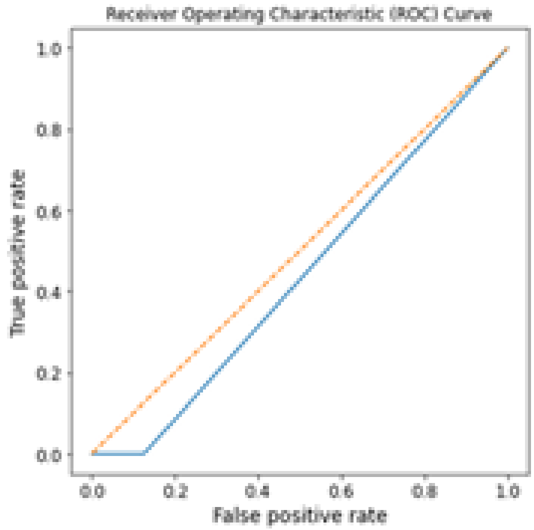



```python
# area under the curve
roc_auc_score(y_test, probs_oc)
```

```
0.6469109195402298
```

Naive Bayes: Late_opening_school

```
# use the roc_curve class to calculate the necessary values
fper, tper, thresholds = roc_curve(y_test, probs)
# the plotting
plt.figure(figsize=(6,6))
plt.plot(fper, tper)
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.show()
```

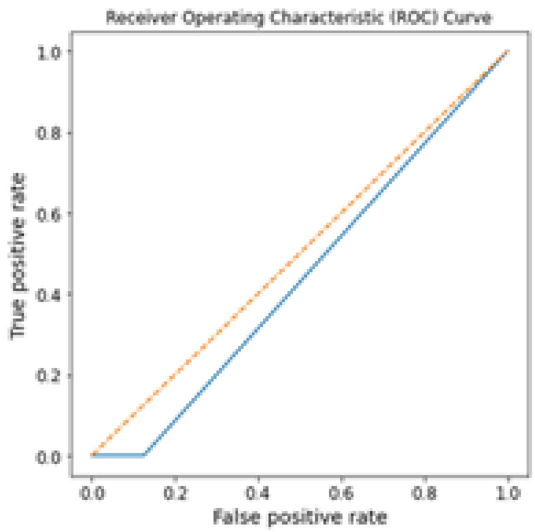



```
# area under the curve
roc_auc_score(y_test, probs_los)
```

0.8101781170483461

Naive Bayes for 'Pre-school_ind

```
# use the roc_curve class to calculate the necessary values
fper, tper, thresholds = roc_curve(y_test, probs)
# the plotting
plt.figure(figsize=(6,6))
plt.plot(fper, tper)
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.show()
```

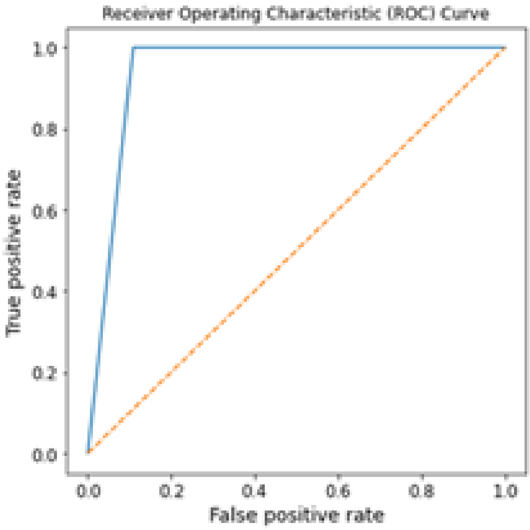



```
# area under the curve
roc_auc_score(y_test, probs_pi)
```

```
0.5726981351981352
```

Naive Bayes for 'Intensive_english_centre

```
#roc_curve class to calculate the necessary values
fper, tper, thresholds = roc_curve(y_test, probs_iec)
# the plotting
plt.figure(figsize=(6,6))
plt.plot(fper, tper)
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.show()
```




```
# area under the curve
roc_auc_score(y_test, probs_iec)
```

```
0.9454545454545454
```

ChatGPT queries

Yes, you can use Naive Bayes with data that has categorical variables converted to numerical variables using one-hot encoding. In fact, Naive Bayes can be applied to datasets with binary or numerical features, including those that have undergone one-hot encoding. Here's how it works:

1. **Binary Features (0 or 1)**: In cases where one-hot encoding results in binary features (0 or 1), you can use the Bernoulli Naive Bayes variant of the Naive Bayes classifier. Bernoulli Naive Bayes is designed for binary data, where features represent the presence (1) or absence (0) of a specific attribute.
2. **Multiple Categories (Multinomial Features)**: If one-hot encoding creates multiple categories (0 or 1 for each category), you can use the Multinomial Naive Bayes variant. Multinomial Naive Bayes is well-suited for count-based data, making it a good choice for text classification problems, where features represent word frequencies or counts.
3. **Independent Features Assumption**: Naive Bayes, including the Bernoulli and Multinomial variants, assumes that features are conditionally independent given the class label. This independence assumption simplifies the probability calculations, and while it may not hold strictly in all cases, Naive Bayes can still perform well in practice.
4. **Text Classification**: Naive Bayes, especially Multinomial Naive Bayes, is commonly used for text classification tasks. In these tasks, one-hot encoding often results in binary or count-based features, making it a suitable choice for analyzing text data.

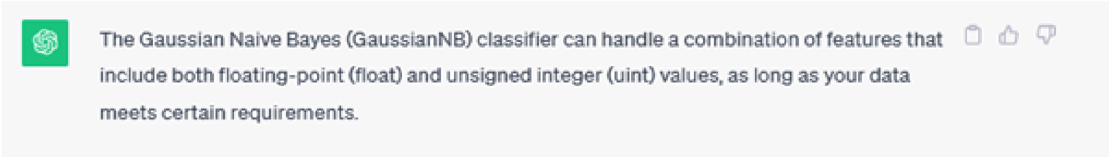
Here's a general workflow for using Naive Bayes with one-hot encoded data:

1. Preprocess your data, including one-hot encoding your categorical variables.
2. Split your dataset into a training set and a testing set.
3. Choose the appropriate variant of Naive Bayes (e.g., Bernoulli or Multinomial) based on your data and classification task.
4. Train the Naive Bayes model using the training data, specifying the target variable and the features created through one-hot encoding.
5. Evaluate the model's performance using the testing data, calculating metrics such as accuracy, precision, recall, and F1-score.

Regenerate

In summary, Naive Bayes can be used effectively with data that has categorical variables converted to numerical variables using one-hot encoding, provided that the variant of Naive Bayes aligns with the nature of your data (binary or count-based). It's a particularly good choice for text classification problems and can be a valuable tool in your machine learning toolkit.

Categorical variables were converted to numerical variables using OneHotEncoding



GaussianNB classifier was used for model building

Comparison

Neural Network is generally better in capturing complicated datasets. It is better than SVM for this because of the high dimensional data and too many columns. The number of true negatives captured by SVM was too low, therefore it may not be appropriate model for this. Naïve Bayes is not a good model for this as it has this independent assumption quality where it assumes all the variables are independent which is not true. It was not able to capture the correlated variables.

References

[1] https://data.cese.nsw.gov.au/data/dataset/nsw-public-schools-master-dataset

[2] WANG, R. 2023. Module 10 - Naive Bayes

[3] OpenAI's ChatGPT. (2023). Code for label encoding in SVM machine learning. Retrieved from ChatGPT conversation. [19/10/2023]

[4] OpenAI's ChatGPT. (2023). Code for mean accuracy score in SVM machine learning. Retrieved from ChatGPT conversation. [19/10/2023]

Appendix

GitHub Link: https://github.com/WSU-Data-Science/assignment-2-pa-23-group-10