Online Shoppers Purchase Intention Prediction

Harichandana Neralla,
HXN210336
The University of Texas at Dallas

Abstract

In our project, the model was designed for a binary classification problem computing the shopper's intention whether a user will make the transaction or not after visiting the shopping website. We trained our classification model using Decision Tree, SVM with our own implementation and analysis using Scikit's Multi-Layer Perceptron Logistic Regression, random forest, gradient boosting, SVM, SVM-rbf kernel, SVRegression, XGBoost and decision tree. The dataset was designed so that each session over the course of a year would belong to a distinct user in order to remove any tendency to a specific campaign, special day, or user profile. While implementing classification problems the structure of the data is very important. The dataset that we are using is imbalanced. Hence, we have used both undersampling and oversampling to know which one works better and compared the results. Data preprocessing contains feature selection, scaling, and shuffling of the data reducing the generalization. We also did hyperparameter training using a grid search and found the better-performing parameters. After training the model with different sets of data, analysis is made with the help of ROC, accuracy, precision, and recall parameters. Cross-validation is also done for SVM for understanding our model.For SVR, as it is the regression after the cross-validation is done, analyzed using metrics like Mean Suqare Error(MSE), Mean Absolute Error(MAE), Root Mean Square Error(RMSE), R2 score and plotted predictions according to the batch data.

Problem Statement

In today's business environment, small-scale industries face stiff competition from e-commerce giants like Amazon, resulting in lower sales volume and decreased revenue. In an effort to revitalize their businesses, there is a desire to better understand the changing consumer preferences and behaviors to inform marketing strategies that will increase purchase intent. This project seeks to leverage customer data to develop a predictive model that can accurately anticipate customers' purchasing intentions at small-scale industry stores. The model will inform marketing strategies and help drive revenue growth for these industries.

Dataset Description

Our model was developed using the UCI ML public repository of Online Shoppers Purchasing Intention Dataset which consists of feature vectors of 12,330 sessions where each session belongs to a different user purchasing history in one year. A total of 12,330 sessions were included in the dataset, of which 10,422 were negative class samples that resulted in not purchasing and the remaining 1908 were positive class samples that resulted in purchasing. Initial data type columns: bool(2), float64(7), int64(7), object(2)

Dataset Preparation

After encoding the categorical data to numerical values, we have divided the training and testing datasets, with 20 percent of the test data. Further, we have divided the training dataset into 4 batch samples manually, where we have taken data in the ratio of 1:1, 1:2, 1:3, 1:4.So, for every one positive sample to 1,2,3,4 samples of negative examples. We have calculated the prediction value while training the dataset with the models to compare the results.We have more negative samples in our dataset

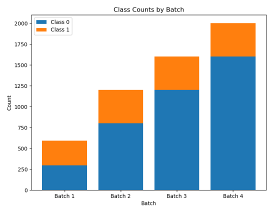


Figure 1: Oversampling before and after of classes 0 and 1

Correlation Matrix

Feature selection assists in avoiding overfitting and dimensional disaster issues by attempting to optimize model performance and minimizing the number of features in the model. In this approach, feature selection offers a model interpretation as an additional benefit. The output model gets simpler and easier to understand with fewer features, making it simpler to believe the model's predictions. We have used SelectKBest from scikit to select best features for our model training

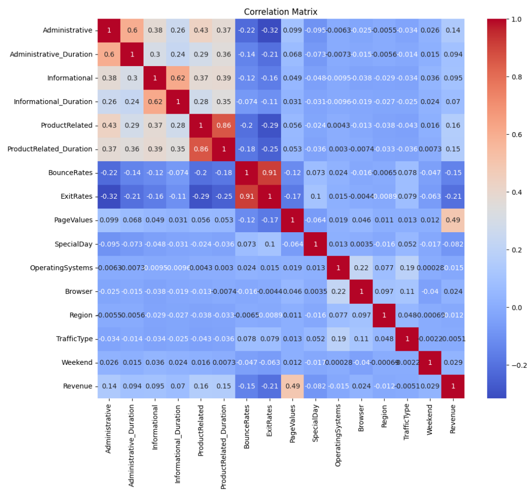


Figure 2: correlation matrix

Implementation

After, sampling the data we have trained the dataset using a decision tree, SVM-linear/gaussian, from our own implementation from scratch. And later we used 3 additional scikit algorithms for analysis purposes.

Decision Trees

Decision tree is a supervised learning algorithm that is frequently employed for categorization tasks. The tree structure is employed to model choices and potential outcomes. A test on a specific feature is represented by each internal node of the tree, the test's result is represented by each branch, and the class label is represented by each leaf node. By building a tree out of the training data and selecting the most appropriate feature to divide at each node, decision trees learn by repeatedly splitting the data into smaller subsets. Pruning and defining a limit depth for the tree are two strategies that can be used to produce a tree that generalizes effectively to any unknown data. We have used ID3 algorithm for implementation.

Batch 1:3					
Depth	Test Error	Accuracy	Precision	Recall	F1 Score
1	12.77	87.22	58.63	79.31	67.42
2	12.77	87.22	58.63	79.31	67.42
3	12.77	87.22	58.63	79.31	67.42
4	12.00	87.99	63.33	66.42	64.84
5	12.73	87.26	58.96	77.61	67.01
6	11.84	88.15	64.06	65.93	64.98
7	12.90	87.10	58.69	76.39	66.38
8	11.68	88.32	65.26	63.99	64.61
9	12.12	87.87	62.01	70.31	65.90
10	11.76	88.24	64.57	65.20	64.89

We generated the ROC curve for the batch that obtained the highest performance, which was the 1:3 batch dataset. After running 10 iterations for each depth, we found the highest accuracy was achieved by the decision tree with a

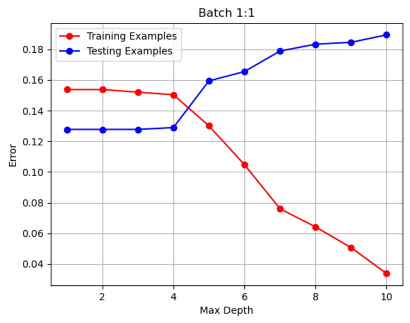
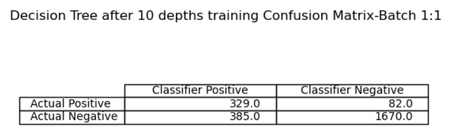


Figure 3: Train and Test Errors of Batch 1:1 using Decision Tree


Decision Tree after 10 depths training Confusion Matrix-Batch 1:1

	Classifier Positive	Classifier Negative
Actual Positive	329.0	82.0
Actual Negative	385.0	1670.0

Figure 4: Confusion of Batch 1:1 using Decision Tree

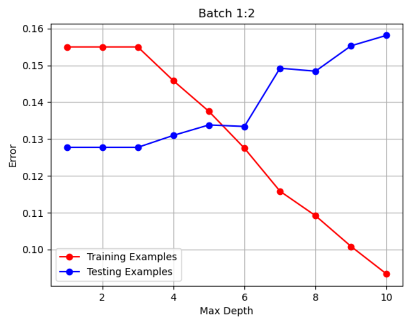
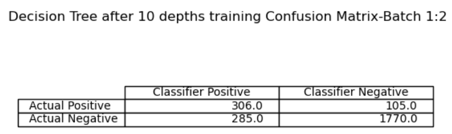


Figure 5: Train and Test Errors of Batch 1:2 using Decision Tree


Decision Tree after 10 depths training Confusion Matrix-Batch 1:2

	Classifier Positive	Classifier Negative
Actual Positive	306.0	105.0
Actual Negative	285.0	1770.0

Figure 6: Confusion Matrix of Batch 1:2 using Decision Tree

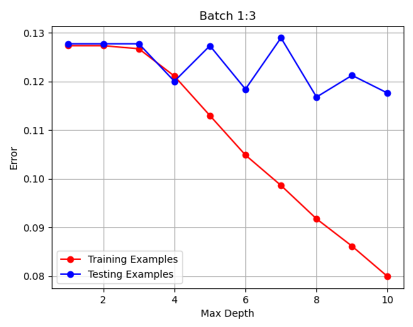
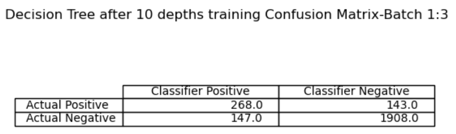


Figure 7: Train and Test Errors of Batch 1:3 using Decision Tree



Decision Tree after 10 depths training Confusion Matrix-Batch 1:3

	Classifier Positive	Classifier Negative
Actual Positive	268.0	143.0
Actual Negative	147.0	1908.0

Figure 8: Confusion Matrix Errors of Batch 1:3 using Decision Tree

depth of 8. We selected this tree and calculated its true positive rate and false positive rate to plot the ROC curve. The resulting ROC curve is shown below.

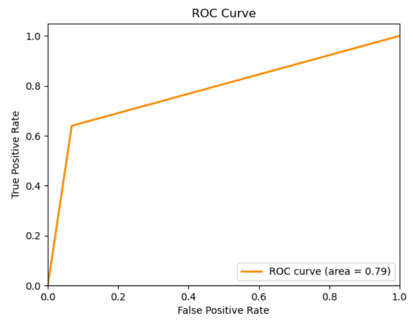


Figure 9: Confusion Matrix of Batch 1:3 using Decision Tree

Why decision tree?

Main reason is that decision trees can handle non-linear relationships between features and target variables well, which might be present in our dataset. We found that the Decision Tree algorithm provided the best results, particularly at depths 1 to 4, while deeper trees resulted in overfitting. Therefore, I recommend using Decision Trees with a depth of 4 or less for this dataset.

The dataset was trained using the decision tree algorithm, batch-wise, with each batch trained for 10 depths. Analysis of the results indicated that, with an increase in depth, the test error for Batch 1:1 increased while the accuracy decreased. It was observed that depths 1 to 4 provided better results when sampling the features from one negative sample to a positive sample. Increasing the depth beyond 4 resulted in decreased accuracy.

In contrast to batch 1:1, batch 1:2 exhibited superior accuracy. However, increasing the tree depth from 1 to 10 resulted in increased test errors and decreased accuracy. Despite this decline, accuracy remained higher than that of the 1:1 batch. Thus, using a ratio of 1 negative sample to 2 positive samples resulted in improved accuracy.

Batch 1:3 demonstrated significant improvements in comparison to the previous batches. As the model was trained from depths 1 to 10, there was a decrease in test error, although there were a few instances where it increased. However, at a depth of 10, the accuracy improved significantly. This suggests that using a ratio of 3 positive samples to 1 negative sample proved to be effective in enhancing the model's performance.

After increasing the ratio of positive to negative samples, we further increased it to 1:4. However, upon training the batch, we observed that although the results were similar to batch 1:3, the test error increased and the accuracy decreased. This indicates that the model has become more complex and may have overfit the data.

In conclusion, the decision tree model was evaluated on all batches, from 1:1 to 1:4, and proved to be accurate. Our model is susceptable to class imbalance but that affect didn't influence decision tree much. The 1:3 batch was found to be the most suitable dataset for model training. To compare our model's performance with that of scikit-learn's in-built decision tree model, we ran the tests on all batches and present the results below.

Batch 1:1	
Batch	Accuracy
1:1	79.64
1:2	81.18
1:3	84.95
1:4	85.23

Upon analyzing the table presented above, we can observe an increase in accuracy as we move from one batch to another, indicating an improvement in the model's performance as we balance the dataset gradually. However, our model yielded slightly higher accuracy when comparing our model's results with Scikit's decision tree model. The confusion matrix and graphviz visualization for each batch are attached to the code for reference.

Support Vector Machine

It operates by identifying the hyperplane that best divides the data into various classes. The margin is maximized by selecting the hyperplane, we should also make sure that we don't overfit. The kernel trick is a method that SVM can employ to transfer data into a higher-dimensional space, where it could be simpler to locate a separating hyperplane.

Why SVM?

Our dataset was certainly not linearly separable as it was the pattern of human shopping-browsing behavior. We realized that our model has a lot of non-linear relationships between the features and the target variable (purchasing intention). Kernel, can capture these non-linear relationships and create flexible decision boundaries. And also our dataset has a high number of features so, SVM would project it in high dimensional feature space. Additionally, SVM is also robust to outliers.

Linear-Kernel: A linear kernel in SVM defines the decision boundary as a linear hyperplane that separates the classes. But this didn't work in our case as it is a very simple model to our dataset which wouldn't separate non-linear data and led to accuracy of around 40% and even lower precision and recall.

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$$

where: $\mathbf{x}$ and $\mathbf{y}$ are the input vectors.

Gaussian-Kernel: In SVM it is a non-linear kernel that maps the data into a high-dimensional space and defines the decision boundary as a non-linear surface. The kernel function measures the similarity between two data points in the high-dimensional space and is based on the radial basis function (RBF).
For our dataset, after categorical encoding, the number of features available while loading the dataset is 29. This is transformed into the higher dimension (592 from the Gaussian formula). Then gaussian kernel is applied to classify the input in higher dimensions and parameters are chosen to generalize the model well.

$$K(x, y) = \exp\left(-\frac{||x-y||^2}{2\sigma^2}\right)$$

Here, x and y are data points or vectors in the input space, diff in the numerator represents the squared Euclidean distance between the vectors and the parameter in the denominator controls the width or spread of the kernel.
As we decrease the width, the accuracy of the model decreases. But even though higher width of 0.1 leads to 99.7% accuracy, it overfits the model. That's why we chose the width of 0.01 with 94.7% for our model.

Loss function: In the SVM model, it measures the error or cost during the training process. It consists of two terms: a misclassification term and a regularization term. The misclassification term penalizes the model for incorrect predictions, while the regularization term helps prevent overfitting we used 20 for C in our training but the effect of C didn't change much.

$$Loss = \frac{C}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \hat{y}_i) + \frac{\mathbf{w}^2}{2}$$

Prediction The prediction function in SVM is used to predict the class labels for new input data. The prediction is made based on the sign of the dot product between the input data and the weights, plus the bias term.

$$Pred(\mathbf{x}) = \{\, 1\ , if \quad \mathbf{x} \cdot \mathbf{w} + b > 0 \quad -1, otherwise$$

Data processing-pre:

Online shopping Intention doesn't have missing values and has 19 features. To train the data, we need to encode categorical variables, scale, and normalize numerical features. This is done using LabelEncoder() for 'Month', 'VisitorType', and 'Weekend' to encode and StandardScaler() from the scikit library on all input numerical features to get the data in a fixed range of values. Also, to reduce bias, and improve generalization we shuffled the data using the shuffled library from scikit.

Feature selection:

To reduce the irrelevance of a few features due to a large number of input features in the dataset, we used the SelectKBest function from the scikit library. This reduced the dimensionality of the dataset by selecting only the most informative features.
As we decreased the number of features to select, accuracy increased till a point but later started to overfit. For a feature size of 20, testing accuracy was 87%, for 12 features, it changed to 91% with a precision of 0.49 but for 5 features, precision changed to 0.16. This meant that significant information was lost with columns. So, we have to find the right balance. In our data set, we used 12 features throughout.

Hyperparameters:

In the case of our model, we used 4 hyperparameters to tune the model.

- Lr:0.01 As we increased the learning rate, the loss function over several iterations went steep which meant the learning was fast but the effect was least seen in model evaluation metrics as accuracy, and precision were almost the same for [0.01, 0.1, 1].
- C:0.1 Regularization constant controls the trade-off between maximizing the margin and minimizing the training error. In our case higher C values tends to overfit the data and lead to lower precision values. The C value that worked well for us is 0.1(similar results for 0.01, 1) against 30, 50
- epochs:10Changing the number of iterations alone didn't change the performance of the model
- width:0.01 It controls the spread or width of the Gaussian kernel in the decision boundary that we plotted after using PCA after projecting 29 features to 2 features.

Prediction

SVM gaussian model avg observations

lr=0.01, C=0.1, epochs=10, width=0.01

Batch	Train accuracy	Test Accuracy	Precision	Recall
1:3	96.56	88.91	25.90	55.32

Cross validation-observations

We used scikit's KFold method to perform the k fold cross validation on out dataset. Our model is susceptable to class imbalance and hence didn't perform as expected. Average

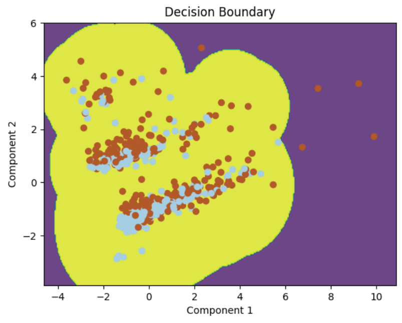


Figure 10: TDecision boundary in 2d using PCA

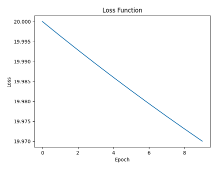


Figure 11: Loss values against number of epochs

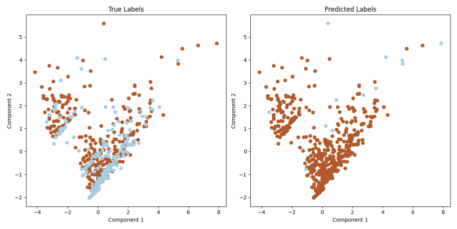


Figure 12: True values vs Predicted values of 2d features using PCA

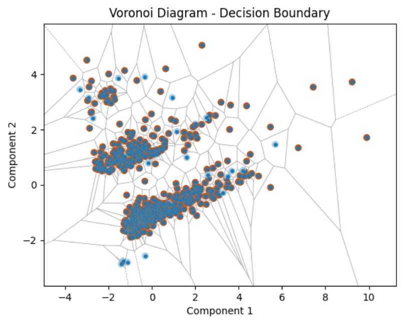


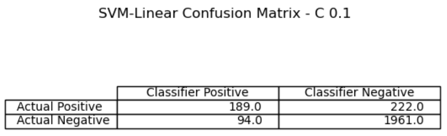
Figure 13: Vernoi decision boundary based on proximity

Accuracy: 0.487, Average Precision: 0.460, Average Recall: 0.591. Our model also overfit the data because complex kernel and more number of redundant data even after feature selection. Few of the features didn't help train the model well. We also used stratified k fold but didn't impact much in few of the folds

Kfold = 4			
Fold	Accuracy	Precision	Recall
1	0.988	1	0.987
2	0.643	0.423	0.624
3	0.458	0.22	0.448
4	0.277	0.42	0.43

Scikit-observations

Batch	Accuracy	Precision	Recall	F1 Score
1:3	87.26	65.90	48.90	56.14



SVM-Linear Confusion Matrix - C 0.1

	Classifier Positive	Classifier Negative
Actual Positive	189.0	222.0
Actual Negative	94.0	1961.0

Figure 14: Confusion Matrix of Batch 1:3 using SVM-scikit

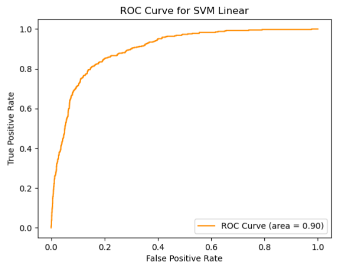


Figure 15: Learning Curve of Batch 1:3 using SVM-Scikit

Experimental Analysis on scikit models

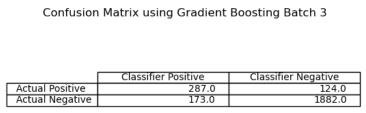
We utilized the scikit learn package's Logistic Regression, Gradient Boosting, and RandomForest classifier models to examine our dataset and assess the performance of various algorithms.

GradientBoosting

Gradient Boosting is a machine learning algorithm that combines the predictions of several individual models in a process known as ensemble learning. The model is built sequentially and at each stage, it focuses on the data points that the previous model failed to predict accurately. Gradient boosting works by optimizing a cost function that measures the difference between the predicted and actual output. It is a powerful algorithm that has proven to be effective in solving a wide range of problems, including classification, regression, and ranking tasks.

Gradient Boosting Results of 4 Batches				
Batch	Accuracy	Precision	Recall	F1 Score
1:1	82.89	49.18	80.29	61.00
1:2	85.28	54.15	76.15	63.30
1:3	87.96	62.39	69.82	65.90
1:4	87.88	64.21	61.55	62.86

Based on the table shown above, batch 1:3 has a higher accuracy score compared to the other batches. Therefore, we have presented its corresponding confusion matrix.


Confusion Matrix using Gradient Boosting Batch 3

	Classifier Positive	Classifier Negative
Actual Positive	287.0	124.0
Actual Negative	173.0	1882.0

Figure 16: Confusion Matrix of Batch 1:3 using Gradient Boosting

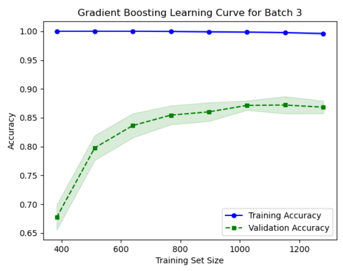


Figure 17: Train and Test Errors of Batch 1:3 using Gradient Boosting

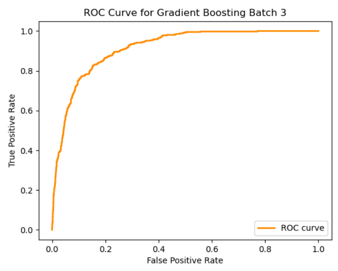
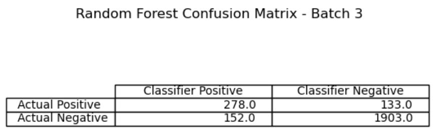


Figure 18: Confusion Matrix of Batch 1:3 using Gradient Boosting

Random Forest

Random Forest is a type of ensemble learning method that combines multiple decision trees to generate more accurate and stable predictions. In Random Forest, each decision tree is built independently using a random sample of the training data and a random subset of the features. The final prediction is made by aggregating the predictions of all the trees in the forest. Random Forest is a powerful algorithm that can handle large datasets and high-dimensional feature spaces, and it is widely used in various applications such as classification, regression, and feature selection.

Random Forest Results of 4 Batches				
Batch	Accuracy	Precision	Recall	F1 Score
1:1	83.70	50.65	85.40	63.59
1:2	86.98	58.52	75.18	65.81
1:3	88.44	64.65	67.63	66.11
1:4	88.77	67.18	63.74	65.42


Random Forest Confusion Matrix - Batch 3

	Classifier Positive	Classifier Negative
Actual Positive	278.0	133.0
Actual Negative	152.0	1903.0

Figure 19: Confusion Matrix of Batch 1:3 using Random Forest

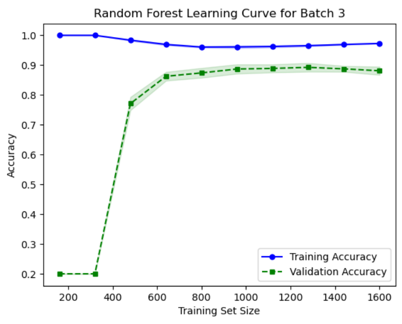
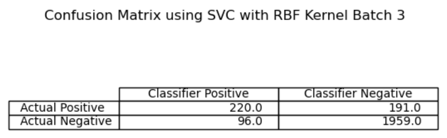


Figure 20: Train and Test Errors of Batch 1:3 using Random Forest

SVM-rbf Kernel

Support Vector Machines (SVM) with the Radial Basis Function (RBF) kernel is a powerful classification algorithm widely used for solving complex classification problems. SVM-RBF finds the optimal hyperplane that separates data points of different classes by transforming them into a higher-dimensional space using the RBF kernel. This kernel measures the similarity between samples based on their distance, allowing SVM to capture non-linear relationships in the data.

SVM with rbf Kernel Results of 4 Batches				
Batch	Accuracy	Precision	Recall	F1 Score
1:1	81.71	47.07	78.10	58.74
1:2	87.47	62.56	61.80	62.18
1:3	88.36	69.62	53.52	60.52
1:4	88.16	70.88	49.14	58.05


Confusion Matrix using SVC with RBF Kernel Batch 3

	Classifier Positive	Classifier Negative
Actual Positive	220.0	191.0
Actual Negative	96.0	1959.0

Figure 21: Confusion Matrix of Batch 1:3 using SVM-rbf

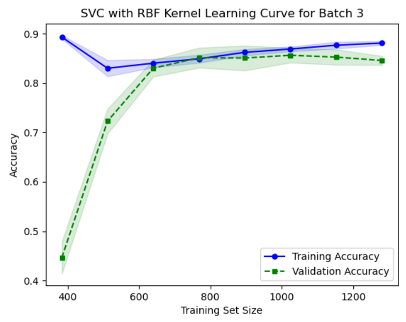


Figure 22: Train and Test Errors of Batch 1:3 using SVM-rbf

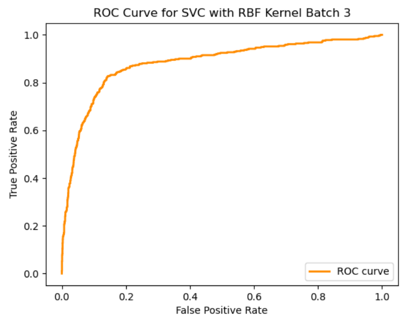


Figure 23: Learning Curve of Batch 1:3 using SVM-rbf

SVR

Support Vector Regression (SVR) is a machine learning algorithm used for solving regression problems. It is an extension of Support Vector Machines (SVM) and utilizes the same principles of finding an optimal hyperplane. SVR aims to find a function that approximates the relationship between input variables and the target variable, allowing for accurate predictions of continuous values. SVR uses a kernel function, such as the Radial Basis Function (RBF) kernel, to map the input data into a higher-dimensional space, enabling the algorithm to capture non-linear relationships. By controlling parameters like the regularization parameter (C) and the kernel parameter (gamma), SVR can balance between model complexity and generalization.

SVR with rbf Kernel Results of 4 Batches				
Batch	MAE	MSE	RMSE	R2 Score
1:1	26.08	13.10	36.19	5.60
1:2	21.55	10.37	3.20	25.35
1:3	18.97	9.12	30.20	34.34
1:4	19.22	9.30	30.49	33.05

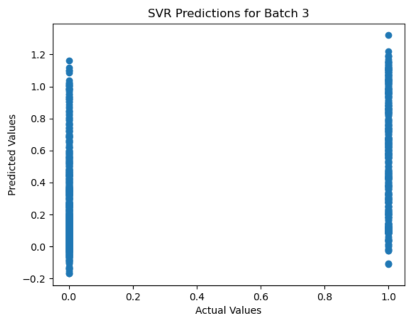


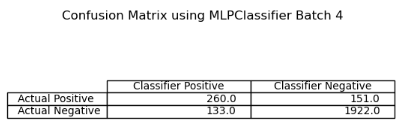
Figure 24: Predictions plots of Batch 1:3 using SVR-rbf

Though it's a regression model I have tried to analyse the dataset by this model as well. As my problem statement suggests this is not only a prediction report but also analysis of the datset. I have added only the batch-3 1:3 dataset predictions as that one is giving optimal results compared to other dataset batches. Overall, the results suggest that the SVR model with rbf kernel may not be performing optimally for the classification problem. The MAE, MSE, and RMSE values indicate that the model's predictions have some level of deviation from the actual values. The R2 scores indicate that the model explains a modest amount of variance in the target variable, but there is room for improvement. To further evaluate more, I have added multi layer perceptron and XGBoost algorithms to better analyse the data.

Multi-Layer Perceptron

A multi-layer perceptron (MLP) is a type of artificial neural network that is capable of learning complex patterns and relationships in data. It consists of multiple layers of interconnected neurons, with information flowing from the input layer through hidden layers to the output layer. Through a process called backpropagation, MLPs adjust the weights and biases of the neurons to minimize a loss function during training.

MultiLayer Perceptron Results of 4 Batches				
Batch	Accuracy	Precision	Recall	F1 Score
1:1	80.05	44.37	77.61	56.46
1:2	85.48	54.90	72.26	62.39
1:3	88.24	64.51	65.45	64.98
1:4	88.48	66.16	63.26	64.68


Confusion Matrix using MLPClassifier Batch 4

	Classifier Positive	Classifier Negative
Actual Positive	260.0	151.0
Actual Negative	133.0	1922.0

Figure 25: Confusion Matrix of Batch 1:3 using MultiLayer Perceptron

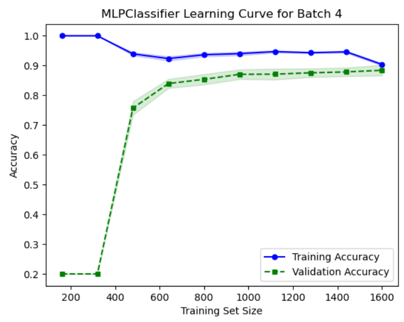


Figure 26: Train and Test Errors of Batch 1:3 using Multi-Layer Perceptron

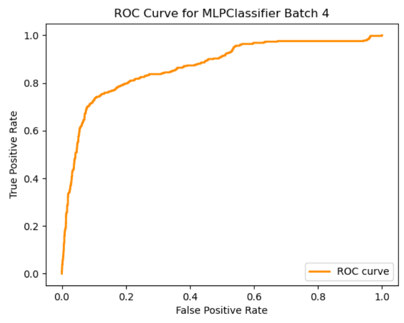
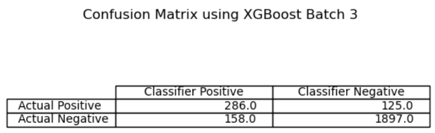


Figure 27: Learning Curve of Batch 1:3 using MultiLayer Perceptron

Extreme Gradient Boosting(XGboost)

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm that belongs to the gradient boosting family. It is widely recognized for its exceptional performance in various data science competitions and real-world applications. XGBoost combines the strengths of gradient boosting and regularized learning to deliver accurate and efficient models. It utilizes a boosted ensemble of decision trees, where each tree is constructed to correct the mistakes made by the previous trees. XGBoost incorporates techniques such as tree pruning, regularization, and parallel processing to optimize model performance and prevent overfitting. With its ability to handle large datasets, feature importance analysis, and built-in handling of missing values, XGBoost has become a popular choice for tasks like classification, regression, and ranking. It has proven effective in domains such as finance, healthcare, and online advertising, making it a valuable tool in the data scientist's toolkit.

XGBoost Results of 4 Batches				
Batch	Accuracy	Precision	Recall	F1 Score
1:1	83.01	49.41	81.50	61.52
1:2	86.37	56.76	76.64	65.22
1:3	88.52	64.41	69.58	66.90
1:4	88.24	65.39	62.53	63.93


Confusion Matrix using XGBoost Batch 3

	Classifier Positive	Classifier Negative
Actual Positive	286.0	125.0
Actual Negative	158.0	1897.0

Figure 28: Confusion Matrix of Batch 1:3 using XGBoost

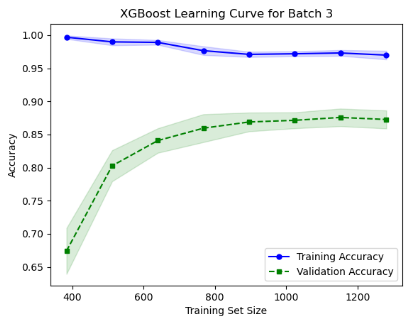


Figure 29: Learning Curve of Batch 1:3 using XGBoost

Overall Observations

Up to this point, we have applied decision tree and SVM with Gaussian kernels to the dataset with different class distributions and discovered that the 1:3 (pos:neg) batch consistently performed the best with each algorithm, yielding the highest accuracy, precision, and recall. Based on these results, the decision tree model appears to be the most effective for our trained datasets. In SVM, class imbalance plays a major role and hence produces low performance on our dataset when we reduce the size of the input while doing cross-validation. Since we also project the data into higher dimensions it is difficult to view the decision boundary. Also with the current dataset, SVM with a linear model is very simple and has a very high bias but SVM with a Gaussian kernel is more complex and tends to overfit the data. Because of this precision is low in SVM gaussian implentation. This means that there might be false positives. But having low precision is not a worrying factor for our problem as we can afford to have more promotions. So we would not recommend using SBM for the current problem, rather use a decision tree maybe with bagging and boosting if required. Additionally, we have utilized Scikit-learn algorithms such as decision tree, SVM-Linear, gradient boosting, logistic regression, and random forest classifier to demonstrate accuracy computation and compare each model to determine which one is optimal for our dataset. Among these, random forest performs better, and would recommend using it.

After all the above observations and additional Scikit-learn models such as multi layer perceptron, SVM-rbf Kernel, XGBoost and SVR to evaluate the metrics for the classification for the demonstration of the accuracy. Among all the previously evaluated algorithms and we can conclude that XGboost(Extreme Gradient Boosting) has the better performance than others and we can recommend this algorithms for this dataset to use.

References

- https://link.springer.com/article/10.1007/s00521-018-3523-0
- https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset
- https://en.wikipedia.org/wiki/Decision tree
- https://en.wikipedia.org/wiki/Support-vector machine
- https://ieeexplore.ieee.org/abstract/document/9038521
- https://towardsdatascience.com/predict-loan-eligibility-using-machine-learning-models-7a14ef904057
- https://ieeexplore.ieee.org/document/9076521